PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Modelling input texts: from Tree Kernels to Deep Learning

Aliaksei Severyn

Advisor:

Prof. Alessandro Moschitti

Università degli Studi di Trento

April 2015

Acknowledgements

Writing a PhD dissertation requires a considerable effort, willingness and motivation to explore new research topics and, of course, the right mindset to go on when facing the unavoidable failures, dead end ideas and paper rejections. I can hardly imagine finishing this work without the help of others. I would like to greatly thank the following people:

- My advisor who inspired me to pursue a PhD in the first place and who has greatly helped me to perform at my best through the numerous technical discussions, suggestions and tips on carrying out research and academic writing.
- The current and former members of our group, and in particular Olga Uryupina, Barbara Plank, Massimo Nicosia, and Kateryna Tymoshenko for a fruitful collaboration on several joint papers.
- My supervisors Katja Filippova and Ioannis Tsochandaridis during my internship at Google Zurich, and Anastasios Kakalis at Google London for their great assistance through my industry experience while taking extra effort to make sure it was aligned with my research interests as much as it was possible.
- To Ryan McDonald for an interesting discussion that sparked my interest in Deep Learning methods and Nal Kalchbrenner who provided tips on relevant papers and sharing his experience.

For two years I was supported by the Google Doctoral Fellowship which gave me the possibility to focus on my research. I am very grateful for this gift from Google.

I would also like to thank the students of an MSc course on NLP and IR (for which I was a TA) who helped me appreciate the joy of teaching, and my friends Alexandre Kandalintsev, Iryna Haponchyk, and Andrey Bogomolov who helped me keep a better balance taking time outside of research and enjoy the life in Trento.

Finally, I would like to thank my family for their love, endless support and encouragement to help me get through the hardships of a PhD path and especially my wife Ia who is always happy.

Abstract

One of the core questions when designing modern Natural Language Processing (NLP) systems is how to model input textual data such that the learning algorithm is provided with enough information to estimate accurate decision functions.

The mainstream approach is to represent input objects as feature vectors where each value encodes some of their aspects, e.g., syntax, semantics, etc. Feature-based methods have demonstrated state-of-the-art results on various NLP tasks. However, designing good features is a highly empirical-driven process, it greatly depends on a task requiring a significant amount of domain expertise. Moreover, extracting features for complex NLP tasks often requires expensive pre-processing steps running a large number of linguistic tools while relying on external knowledge sources that are often not available or hard to get. Hence, this process is not cheap and often constitutes one of the major challenges when attempting a new task or adapting to a different language or domain. The problem of modelling input objects is even more acute in cases when the input examples are not just single objects but pairs of objects, such as in various learning to rank problems in Information Retrieval and Natural Language processing.

An alternative to feature-based methods is using kernels which are essentially non-linear functions mapping input examples into some high dimensional space thus allowing for learning decision functions with higher discriminative power. Kernels implicitly generate a very large number of features computing similarity between input examples in that implicit space. A well-designed kernel function can greatly reduce the effort to design a large set of manually designed features often leading to superior results.

However, in the recent years, the use of kernel methods in NLP has been greatly underestimated primarily due to the following reasons: (i) learning with kernels is slow as it requires to carry out optimizaiton in the dual space leading to quadratic complexity; (ii) applying kernels to the input objects encoded with vanilla structures, e.g., generated by syntactic parsers, often yields minor improvements over carefully designed feature-based methods.

In this thesis, we adopt the kernel learning approach for solving complex NLP tasks and primarily focus on solutions to the aforementioned problems posed by the use of kernels. In particular, we design novel learning algorithms for training Support Vector Machines with structural kernels, e.g., tree kernels, considerably speeding up the training over the conventional SVM training methods. We show that using the training algorithms developed in this thesis allows for trainining tree kernel models on large-scale datasets containing millions of instances, which was not possible before.

Next, we focus on the problem of designing input structures that are fed to tree kernel functions to automatically generate a large set of tree-fragment features. We demonstrate

that previously used plain structures generated by syntactic parsers, e.g., syntactic or dependency trees, are often a poor choice thus compromising the expressivity offered by a tree kernel learning framework. We propose several effective design patterns of the input tree structures for various NLP tasks ranging from sentiment analysis to answer passage reranking. The central idea is to inject additional semantic information relevant for the task directly into the tree nodes and let the expressive kernels generate rich feature spaces. For the opinion mining tasks, the additional semantic information injected into tree nodes can be word polarity labels, while for more complex tasks of modelling text pairs the relational information about overlapping words in a pair appears to significantly improve the accuracy of the resulting models.

Finally, we observe that both feature-based and kernel methods typically treat words as atomic units where matching different yet semantically similar words is problematic. Conversely, the idea of distributional approaches to model words as vectors is much more effective in establishing a semantic match between words and phrases. While tree kernel functions do allow for a more flexible matching between phrases and sentences through matching their syntactic contexts, their representation can not be tuned on the training set as it is possible with distributional approaches. Recently, deep learning approaches have been applied to generalize the distributional word matching problem to matching sentences taking it one step further by learning the optimal sentence representations for a given task. Deep neural networks have already claimed state-of-the-art performance in many computer vision, speech recognition, and natural language tasks.

Following this trend, this thesis also explores the virtue of deep learning architectures for modelling input texts and text pairs where we build on some of the ideas to model input objects proposed within the tree kernel learning framework. In particular, we explore the idea of relational linking (proposed in the preceding chapters to encode text pairs using linguistic tree structures) to design a state-of-the-art deep learning architecture for modelling text pairs. We compare the proposed deep learning models that require even less manual intervention in the feature design process then previously described tree kernel methods that already offer a very good trade-off between the feature-engineering effort and the expressivity of the resulting representation. Our deep learning models demonstrate the state-of-the-art performance on a recent benchmark for Twitter Sentiment Analysis, Answer Sentence Selection and Microblog retrieval.

Keywords
Tree Kernels; Fast Kernel Methods; Semantic Textual Similarity; Youtube comments; Question Answering; Microblog Retrieval; Twitter Sentiment Analysis; Deep Learning

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
  - 1.1.1 The problem of modelling input texts . . . 2
  - 1.1.2 Research goals . . . 4
- 1.2 Contributions and structure of the thesis . . . 6

2 Preliminaries: Supervised Learning with Structural Kernels **13**
- 2.1 Supervised Learning . . . 13
- 2.2 Supervised learning problems in NLP . . . 15
  - 2.2.1 Classification . . . 16
    - Problem formulation . . . 16
    - Example tasks . . . 16
  - 2.2.2 Re-ranking . . . 17
    - Problem formulation . . . 18
    - Learning to Rank approaches . . . 18
    - Example tasks . . . 19
  - 2.2.3 Structured Prediction Problems in NLP . . . 19
    - Problem formulation . . . 20
    - Example tasks . . . 20
- 2.3 Binary Support Vector Machines . . . 23
  - 2.3.1 Primal formulation . . . 23
  - 2.3.2 The dual . . . 24
  - 2.3.3 The kernel trick . . . 26
- 2.4 Structural Kernels . . . 27
  - 2.4.1 String Kernels . . . 27
  - 2.4.2 Convolutional Tree Kernels . . . 28
    - Counting shared fragments . . . 28
    - Syntactic Tree Kernel (STK) . . . 29
    - Partial Tree Kernel (PTK) . . . 30
- 2.5 Summary . . . 31

3 Fast Support Vector Machines with Structural Kernels **33**
3.1 Overview . . . 34
3.1.1 Approximate Cutting Plane Algorithm with model compression . . . 34
3.1.2 Fast linearization of tree kernels . . . 36
3.1.3 Our contributions . . . 37
3.2 Preliminaries: Cutting Plane Algorithm with Sampling . . . 39
3.2.1 Cutting-plane algorithm (primal) . . . 39
3.2.2 Cutting-plane algorithm (dual) . . . 40
3.3 Fast CPA for Structural Kernels . . . 42
3.3.1 Compacting cutting plane models using DAGs . . . 42
3.3.2 DAG tree kernels . . . 42
3.3.3 Fast Computation of the CPM on Structural Data . . . 44
3.4 Implementation of the DAG Kernel . . . 45
3.4.1 DAG construction . . . 46
3.4.2 DAG kernel computation . . . 47
3.4.3 Parallelization . . . 48
3.5 Handling Class-Imbalanced data . . . 48
3.5.1 Cost-proportionate sampling . . . 49
3.5.2 Theoretical Analysis of the Algorithm . . . 51
3.6 Linearization of DAG models . . . 52
3.6.1 Feature enumeration aka feature hashing . . . 52
3.6.2 Reverse Kernel Engineering . . . 53
3.7 Experiments . . . 54
3.7.1 Experimental setup . . . 54
3.7.2 Data . . . 55
3.7.3 Comparative speedup analysis of DAG-based models . . . 56
Learning speedups . . . 57
Classification experiments . . . 59
3.7.4 Tuning up Precision and Recall . . . 61
3.7.5 Parallelization . . . 62
3.7.6 Linearization experiments . . . 63
Data and task . . . 63
Setup . . . 63
Feature-based learning . . . 64
Learning with kernels . . . 65
Tree Kernel Linearization . . . 67
Distributed linearization . . . 68
3.8 Related work . . . 69
3.9 Summary . . . 71

4 Modelling YouTube comments with Shallow Syntactic Tree Structures 75
4.1 Overview . . . 75
4.1.1 Our contributions . . . 76
4.2 Representations and models . . . 77
4.2.1 Feature Set . . . 77
4.2.2 Structural model . . . 78
4.2.3 Learning . . . 79
4.3 YouTube comments corpus . . . 81
4.4 Experiments . . . 82
4.4.1 Task description . . . 82
4.4.2 Data . . . 83
4.4.3 Results . . . 84
In-domain experiments . . . 84
Learning curves . . . 85
Cross-domain experiments . . . 85
4.4.4 Discussion . . . 87
4.5 Related work . . . 87
4.6 Summary . . . 89

5 Modelling text pairs with syntactic trees 91
5.1 Overview . . . 92
5.1.1 Semantic Textual Similarity . . . 92
5.1.2 Microblog Retrieval . . . 93
5.1.3 Our contributions . . . 94
5.2 Semantic Textual Similarity . . . 95
5.2.1 Learning . . . 95
5.2.2 Syntactic models of text pairs . . . 96
5.2.3 Pairwise similarity features . . . 97
5.2.4 Experiments . . . 98
Experimental setup . . . 98
Results . . . 99
5.3 Microblog Retrieval . . . 99
5.3.1 Learning . . . 100
5.3.2 A syntactic model for tweets . . . 100
5.3.3 Experiments . . . 101
Experimental setup . . . 101
Results . . . 102
5.4 Summary . . . 104

6 Modelling Question-Answer Pairs **107**
6.1 Overview . . . 108
6.2 Structural models of question-answer pairs . . . 110
6.2.1 Basic structural representations . . . 110
Shallow tree structures . . . 110
Soft matching for relational linking . . . 111
6.2.2 Refining relational tag using question focus and question category . 113
Question focus detection . . . 113
Question classification . . . 113
Semantic linking . . . 114
6.2.3 Tree kernels for automatic feature engineering . . . 114
6.3 QA tasks . . . 115
6.3.1 Answer passage reranking . . . 115
QA architecture . . . 115
Pairwise reranking with kernels . . . 116
6.3.2 Answer Sentence Selection . . . 117
QA pair classification with tree kernels . . . 117
6.3.3 Answer Sentence Extraction . . . 118
6.4 Feature Vector Representation . . . 119
6.4.1 Basic features . . . 119
6.4.2 Advanced features . . . 120
6.5 Experiments . . . 121
6.5.1 Answer Passage Reranking . . . 121
Experimental setup . . . 121
Data . . . 122
Search engines . . . 123
Relational learning on TREC . . . 123
Soft Word Matching . . . 124
Relational learning using Question Focus and Question Class . . . 125
Learning cross-domain pairwise structural relationships . . . 126
Error Analysis . . . 126
6.5.2 Answer Sentence Selection . . . 128
Experimental setup . . . 128
Results . . . 129
6.5.3 Answer Extraction . . . 130
Recovering answers . . . 130
Best Answer Selection . . . 131
Discussion and Error Analysis . . . 131
6.6 Related Work . . . 133

6.7 Summary . . . 134

7 Deep Learning models of input texts **137**

7.1 Overview . . . 138
7.1.1 Modelling tweets . . . 138
7.1.2 Modelling text pairs . . . 139
7.1.3 Our contributions . . . 141
7.2 Preliminaries: Convolutional Neural Networks . . . 142
7.2.1 Sentence matrix . . . 142
7.2.2 Convolution feature maps . . . 142
7.2.3 Activation units . . . 144
7.2.4 Pooling . . . 144
7.2.5 Hidden layers . . . 145
7.2.6 Softmax . . . 145
7.2.7 Training . . . 146
7.2.8 Regularization . . . 146
7.3 Modelling tweets . . . 146
7.3.1 Network architecture . . . 146
7.3.2 Phrase-level sentiment analysis . . . 147
7.3.3 Initializing the model parameters . . . 147
7.4 Modelling text pairs . . . 148
7.4.1 Sentence model . . . 148
7.4.2 Encoding overlapping words . . . 149
7.4.3 Our architecture for matching text pairs . . . 149
7.4.4 Matching query and documents . . . 149
7.4.5 The information flow . . . 150
7.5 Experiments . . . 151
7.5.1 Twitter Sentiment Analysis . . . 151
Experimental setup . . . 151
Pre-training the network . . . 151
Official rankings . . . 152
7.5.2 Answer Sentence Selection . . . 153
Experimental setup . . . 153
Word embeddings . . . 154
Training and hyperparameters . . . 155
Size of the model . . . 156
Results and discussion . . . 156
7.5.3 TREC Microblog Retrieval . . . 158
Experimental setup . . . 159

Results and discussion 160
7.6 Related Work 162
7.7 Summary 163

8 Summary and Future Work **165**
8.1 Fast SVMs with structural kernels 166
8.2 Syntactic structures for modelling input texts: Youtube comments 167
8.3 Syntactic structures for modelling text pairs 167
8.4 Semantic syntactic structures for Question Answering 168
8.5 Deep Learning architectures for modelling input texts and text pairs 169

Bibliography **171**

List of Tables

3.1 Comparing speedups: STK . . . 58
3.2 Comparing speedups: PTK . . . 59
3.3 Classification speedups . . . 60
3.4 Large-scale experiment on SRL . . . 60
3.5 Handling class-imbalance problem . . . 61
3.6 Results of feature enumeration on SRL . . . 65
3.7 Feature overlap with SVM model . . . 66
3.8 Runtimes for kernelized models. . . . 66
3.9 Accuracy of RKE models . . . 68
3.10 Large-scale learning with RKE . . . 69

4.1 Summary of YouTube comments data . . . 83
4.2 In-domain experiments . . . 84
4.3 Cross-domain experiment . . . 86

5.1 Results on STS-2012 . . . 98
5.2 Comparison with the Uva L2R model . . . 103
5.3 System performance on the top 30 runs from TMB2012 . . . 104
5.4 Results of improving system runs . . . 105

6.1 Example of LDA topics from Aquaint corpus . . . 112
6.2 Mapping from question classes to named entity types. . . . 114
6.3 Search Engine baselines on TREC QA. . . . 123
6.4 Answer passage reranking on TREC QA . . . 124
6.5 Results on TREC QA using various relational tagging schemes . . . 125
6.6 Answer passage reranking on Answerbag. . . . 126
6.7 Testing model robustness: training on Answerbag and testing on TREC QA. 126
6.8 Summary of the TREC data for answer sentence selection. . . . 129
6.9 Answer sentence reranking on TREC 13 . . . 130
6.10 Results on answer extraction . . . 132
6.11 Results on finding the best answer with voting. . . . 132

7.1 Semeval-2015 data . . . 152
7.2 Results on the progress test sets from Semeval-2015 . . . 152
7.3 Results on Semeval-2015 for phrase and tweet-level subtasks . . . 153
7.4 Summary of TREC QA datasets for answer reranking. . . . 154
7.5 Results on TRAIN and TRAIN-ALL from TREC QA using only similarity score and also including the distributional representation of question and answer sentences. . . . 157
7.6 Results on TREC QA when augmenting the deep learning model with relational information about overlapping words. . . . 158
7.7 Survey of the results on the TREC QA answer selection task. . . . 158
7.8 Summary of TREC Microblog datasets. . . . 159
7.9 System performance on the top 30 runs from TMB2012 . . . 161
7.10 Averaged relative improvements on TMB2012 . . . 162

List of Figures

1.1 Shallow syntactic tree representation of the YouTube comment: *"iPad 2 is better. the superior apps just destroy the xoom."* . . . 5
1.2 Excerpt of tree fragment features automatically generated by the tree kernel function for the example shown in Figure 1.1. . . . 5

2.1 Example syntactic trees. . . . 23
2.2 Example tree fragments generated by Tree Kernels. . . . 29

3.1 Three syntactic trees and the resulting DAG. . . . 35
3.2 Comparing asymptotic algorithm efficiency on SRL . . . 56
3.3 Speedups due to parallelization . . . 63
3.4 Learning curves on SRL . . . 67

4.1 Example shallow tree representation of a Youtube comment . . . 78
4.2 In-domain learning curves . . . 86
4.3 Learning curves for the cross-domain setting . . . 86

5.1 A phrase-dependency based structural representation of a text pair . . . 96
5.2 Example shallow tree model of a query-tweet pair . . . 101

6.1 Shallow chunk tree for the q/a pair . . . 111
6.2 Relational tagging of q/a pairs . . . 113
6.3 Some of the tree fragments generated from a q/a pair . . . 115
6.4 Kernel-based answer passage reranking system . . . 116
6.5 Basic CH representation of the example SV pair. . . . 126
6.6 Structural representation of the answer from the SV pair with the relational focus tag. . . . 127
6.7 Incorrect answer with the relational focus tag. . . . 128
6.8 Example q/a pair from SVs not activated after using typed relation focus tag. . . . 128

7.1 The architecture of our deep learning model for sentiment classification. . . . 147

7.2 Our deep learning architecture for reranking short text pairs. 150

Chapter 1

Introduction

1.1 Motivation

Perhaps the most prominent results in developing modern Natural Language Processing (NLP) systems are largely due to the application of statistical machine learning methods. Prior attempts to tackle language-processing tasks were extremely labour intensive typically requiring to hand-engineer a large set of manual rules. For example, early spell-checkers and automatic machine translation systems required to define and encode thousands of rules leading to overly complicated systems that were hard to maintain and adapt to new languages or domains. In the past decades, the manual rule engineering approaches have been successfully replaced with statistical learning methods able to automatically extract and learn such rules through the analysis of large collections of texts.

For example, Google Translate, an automatic machine translation system developed by Google, can translate written text from one language into another supporting over 90 languages and is being used by millions users on a daily basis. Another example of a system where advanced statistical learning methods play an indispensable role to teach a machine to understand language at the level of a human is a Question Answering system developed by IBM Watson. Watson is an open domain question answering system whose core components are primarily based on advanced natural language processing, information retrieval, knowledge representation, automated reasoning, and machine learning technologies.

Given that the amount of textual information generated daily is growing at an increasing rate, building intelligent NLP systems able to adapt to new domains and languages without methods able to automatically extract and capture salient patterns in the data is not possible.

A large class of algorithms used in NLP to tackle various tasks are supervised learning methods, which learn to extract knowledge from labeled training data. Each instance in the training set is a pair consisting of an input object, e.g. a document, sentence, or a

word, and a desired output value, e.g., document category, sentence class or a word type. The goal is to infer a decision function by analysing the training data, which can be used for mapping unseen test examples to their output labels. This process is called *training* and is typically set up to minimize some sort of an error (loss) function computed on the training set. The quality of the trained model is then estimated by how well it can generalize (predict) to the previously unseen test examples.

Hence, before approaching a given supervised learning problem it is generally required to: (i) decide on the type of the input data and construct a labeled training set; (ii) define a model to represent the input objects that will be fed to a learning algorithm; (iii) depending on the nature of the output labels and the class of functions that can be inferred from the training set, select a corresponding learning algorithm; and (iv) train the model and test it on a held out set to assess the model quality.

While there are many possible factors to consider when tackling a supervised learning task, e.g., the size of the training set, the choice of the learning algorithm and its hyper-parameters, etc., one of the key decisions to consider is how to represent the input objects. In fact, the accuracy of the learned model is greatly determined by how well we can model the input data to capture its important features helping to discriminate between examples with different labels. Typically, the input object is transformed into what is called a feature vector, which is essentially a vector of numbers that encode certain aspects of the object it represents.

1.1.1 The problem of modelling input texts

Among the first and most simple ideas to model the input textual information was to treat words as atomic units mapping them to arbitrary integers (drawn from a finite-sized set corresponding to the size of the word vocabulary). This representation is called bag-of-words, whose name stems from the fact that we treat texts as sets of words where the word order information is completely ignored. For example, a sentence `"Mike wants to eat a cake"` and its shuffled variant `"cake eat to a Mike wants"` are absolutely identical from the viewpoint of a learning algorithm that relies on the bag-of-words model. Despite its simplicity, such crude approximation of language, which, as we know, is inherently structured, results in surprisingly accurate models for many applications in Information Retrieval (IR) and NLP.

One straightforward fix to improve the representation power of the bag-of-words model is to also consider word sequences up to a certain size, e.g., n-grams. It has been shown to improve the model quality helping to capture better the local context of individual words. Nevertheless, using n-grams of higher order may result in extremely sparse representations as even using tri-grams produces feature spaces of many millions of dimensions. Additionally, longer range dependencies between words that appear useful to model in more complex tasks are rather problematic to capture using this approach.

For a wide range of more complex NLP tasks the ability to capture the syntactic/semantic relations between words and phrases in a sentence is essential to train models with better generalization. In this case, using a bag-of-word model to represent input texts may lead to rather disappointing results. To increase the expressivity of the model to represent input objects, NLP researchers typically follow the path of encoding additional feature types that capture lexical, syntactic and semantic information.

An even more challenging problem than dealing with input texts is to represent text pairs, which is a common scenario in various IR and NLP applications, e.g., Retrieval, Question Answering, Paraphrasing, Textual Entailment, etc. The most widely used approach is, again, to encode input text pairs using many complex lexical, syntactic and semantic features and then compute various similarity measures between the obtained representations. Typically, input texts are represented using various lexical, syntactic and semantic units such as words, n-grams, part-of-speech tags, dependency chains, predicate-argument relations, etc. These units are then translated into feature vectors using one-hot encoding scheme. Having mapped the input texts into a joint feature space, their similarity can be easily computed by an inner product between their feature vectors. The obtained similarity scores computed over various representations encode the input text pairs into feature vectors which are fed to a learning algorithm. Many state-of-the-art approaches that deal with text pairs follow that schema. For example, in answer passage reranking, Surdeanu et al. [157] employ complex linguistic features, modelling syntactic and semantic information as bags of syntactic and semantic role dependencies and build similarity and translation models over these representations.

Hence, a large effort in the design of NLP systems is devoted to engineering features in an attempt to encode various aspects of input objects that can help to learn better discriminative functions. However, the choice of representations and features is largely an empirical process, driven by the intuition, experience and domain expertise.

An alternative to tedious manual feature engineering required by the feature-based methods is using kernels. Kernels are essentially non-linear functions mapping input examples into some high dimensional space where the inner product between them can be efficiently computed. Thus, defining an expressive kernel function that automatically generates a very large number of features from the input objects can greatly ease the feature engineering problem, while allowing for learning decision functions with higher discriminative power. In fact, a well-designed kernel function is an important step towards automatic feature engineering greatly reducing the effort to design a large set of manual features to train accurate models. Additionally, extracting features from more complex structures such as trees is somewhat problematic for feature-based approaches, as it requires to define exact procedures to encode substructures that can be useful for the task. On the other hand, structural kernels can naturally deal with the inputs that are sequences, trees, graphs, and have been shown very effective on a large number of tasks

in NLP, bio-informatics, data mining, etc.

However, in the recent years the use of structural kernel methods in NLP has been greatly under-estimated primarily due to the following reasons:

- Learning with kernels scales quadratically in the number of training examples thus limiting their application to small datasets;
- Applying structural kernels to input objects represented using vanilla syntactic structures, e.g. syntactic parse trees, results in models that demonstrate minor improvements over carefully designed feature-based methods.

1.1.2 Research goals

In this thesis we adopt the tree kernel learning approach for solving complex NLP tasks and focus on the aforementioned problems posed by the use of kernels: (i) slow training and (ii) designing more expressive structural representations of input texts going beyond plain structures produced by syntactic parsers.

To address the first problem, we design novel algorithms for training Support Vector Machines with structural kernels, e.g., tree kernels, considerably speeding up the training over the conventional SVM training methods. We show that using the novel training algorithms developed in this thesis it is possible to train tree kernel models on large-scale datasets containing millions of instances, which was not possible before.

Next, we focus on the problem of designing input structures that are fed to tree kernel functions to automatically generate a large set of tree-fragment features. We demonstrate that previously used plain structures generated by syntactic parsers, e.g., syntactic or dependency trees, are often a poor choice thus compromising the expressivity offered by a tree kernel learning framework.

To give a flavour of the structural tree representations of texts we propose in this thesis, consider the problem of building a sentiment classifier and the following example comment from a YouTube video about a Motorola Xoom smartphone:

`iPad 2 is better.  the superior apps just destroy the xoom.`

The comment contains one *negative* and two *positive* tokens (for example, identified by a simple lookup in one of the available sentiment lexicons). This would strongly bias a typical feature-based sentiment classifier based on n-grams to assign a *positive* label to the comment. However, the polarity of this comment is *negative* with respect to the target product.

The advantage of the structured representation over the features-based models comes from its ability to encode powerful contextual syntactic features in the form of tree fragments. An example shallow syntactic tree structure (introduced in Chapter 4) to represent our example comment is shown in Fig. 1.1. In contrast to the feature-based models that

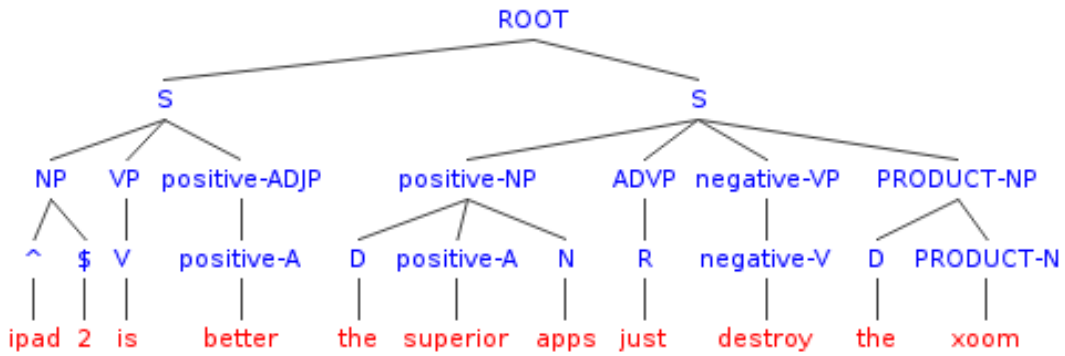


Figure 1.1: Shallow syntactic tree representation of the YouTube comment: *"iPad 2 is better. the superior apps just destroy the xoom."*

typically have difficulties to model syntactic contexts of words, our structural representation encodes the fact that the negative word, *destroy*, refers to the product *xoom*, where the latter is in object relation to the verb *destroy*. Note that we augment the basic syntactic structure (composed of only part-of-speech and chunk tags) with additional semantic labels plugged directly into the tree nodes to encode the target product (motorola *xoom*) and sentiment-bearing words *(better, superior, destroy)*. Consider the excerpt of tree fragments generated by the tree kernel shown in Figure 1.2 (in fact, the kernel function generates many more subtrees). The tree fragment on the left is a strong feature that helps the classifier to discriminate in such difficult cases. In general, tree kernels generate all possible subtrees, thereby producing generalized (back-off) features. For instance, the middle and right subtree shown in Figure 1.2 generalize the one on the left.

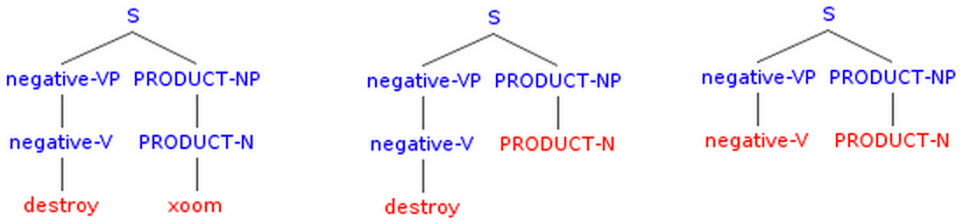


Figure 1.2: Excerpt of tree fragment features automatically generated by the tree kernel function for the example shown in Figure 1.1.

Hence, this thesis explores the idea of modelling input texts with syntactic structural representations augmented with additional semantic information. We propose several effective design patterns of the input tree structures for various NLP tasks ranging from sentiment analysis to answer passage reranking. The central idea is to inject additional semantic information relevant for the task directly into the tree nodes and let the ex-

pressive kernels generate rich feature spaces. For example, for the opinion mining tasks, the additional semantic information injected into tree nodes can be word polarity labels and target products (as we have just seen in Fig. 1.1), while for more complex tasks of modelling text pairs the relational information about overlapping words in a pair appears to significantly improve the accuracy of the resulting models.

Finally, we observe that both feature-based and kernel methods typically treat words as atomic units where matching different yet semantically similar words is problematic. Conversely, the idea of distributional approaches to model words as vectors is much more effective in establishing a semantic match between words and phrases. While tree kernel functions do allow for a more flexible matching between phrases and sentences through matching their syntactic contexts, their representation can not be tuned on the training set as with distributional approaches. Recently, deep learning approaches have been applied to generalize the distributional word matching problem to matching sentences taking it one step further by learning the optimal sentence representations for a given task. Deep neural networks have already claimed state-of-the-art performance in many computer vision, speech recognition, and natural language tasks.

Following this trend, this thesis also explores the virtue of deep learning architectures for modelling input texts and text pairs where we build on some of the ideas to model input objects proposed within the tree kernel learning framework. In particular, we explore the idea of relational linking (proposed in the preceding chapters to encode text pairs using linguistic tree structures) to design a state-of-the-art deep learning architecture for modelling text pairs. We compare the proposed deep learning models that require even less manual intervention in the feature design process then previously described tree kernel methods that already offer a very good compromise between the feature-engineering effort and the expressivity of the resulting representation. Our deep learning models demonstrate the state-of-the-art performance on a recent benchmark for Twitter Sentiment Analysis, Answer Sentence Selection and Microblog retrieval.

In the following we provide an overview of the thesis structure and highlight the contributions made in this thesis.

1.2 Contributions and structure of the thesis

In Chapter 2 we review the general framework for learning with structural kernels. We touch on the most important concepts from the supervised learning theory to give a general description of the learning setup adopted in this thesis. The provided definitions are also handy to give a more formal description of the NLP tasks we consider in this thesis. Then, as a particular instance of the supervised learning methods, we introduce the main machinery behind the kernelized Support Vector Machines that we use to approach most of the NLP problems tackled in this thesis. We also include a formal definition of

some of the most important structural kernels widely used in NLP and that are of central focus in this thesis.

Having introduced the general framework for learning with structural kernels, Chapter 3 describes several solutions to the **first problem** posed by the use of kernel methods: slow training times.

Contribution 1 (Chapter 3) *Novel algorithms for speeding up the training of SVMs with structural kernels.*

We propose the use of Directed Acyclic Graphs (DAGs) to compactly represent trees in the training algorithm of Support Vector Machines (SVMs). In particular, we use DAGs for each iteration of the cutting plane algorithm (CPA) to encode the model composed by a set of trees. This enables DAG kernels to efficiently compute Tree Kernel similarity between the current model and a given training tree. Consequently, the amount of total computation is reduced by avoiding redundant evaluations over shared substructures. We provide theory and algorithms to formally characterize the above idea. In addition, we propose an alternative sampling strategy within the CPA to address the class-imbalance problem, which coupled with fast learning methods provides a viable tree kernel learning framework for a large class of real-world applications. Finally, we use the fast training algorithms with DAG compression to define a novel linearization framework such that the optimization can be performed entirely in the linear space thus closing the gap between the tree kernel learning and learning with the feature-based methods.

In the following Chapters 4, 5, and 6 we focus on the **second problem**—designing syntactic/semantic structures to model input texts within the tree kernel learning framework. The central idea is to inject additional semantic information relevant for the task directly into the tree nodes and let the expressive kernels generate rich feature spaces. For the opinion mining tasks, the additional semantic information injected into tree nodes can be word polarity labels, while for more complex tasks of modelling text pairs the relational information about overlapping words in a pair appears to significantly improve the accuracy of the resulting models.

Contribution 2 (Chapter 4) *Novel shallow syntactic models for opinion mining of YouTube comments.*

We define a systematic approach to Opinion Mining (OM) on YouTube comments by (i) modelling classifiers for predicting the opinion polarity and the type of comment and (ii) proposing robust shallow syntactic structures for improving model adaptability. We rely on the tree kernel technology to automatically extract and learn features with better generalization power than bag-of-words. An extensive empirical evaluation on our manually annotated YouTube comments corpus shows a high classification accuracy and highlights the benefits of structural models in a cross-domain setting.

Contribution 3 (Chapter 5 and 6) *Novel structural relational models for modelling text pairs.*

Effective methods for computing semantic textual similarity of text pairs is at the cornerstone of many NLP applications. We develop several types of relational structural representations and demonstrate their effectiveness on the following NLP tasks: Semantic Textual Similarity, passage reranking in Question Answering, answer sentence selection and answer extraction, and Microblog Retrieval. Different from the majority of approaches, where a large number of pairwise similarity features are used to represent a text pair, the proposed model features the following: (i) it directly encodes input texts into relational syntactic structures; (ii) relies on tree kernels to handle feature engineering automatically; (iii) combines both structural and feature vector representations in a single scoring model; and (iv) performs on the par or delivers significant improvements over strong feature-based models.

Contribution 4 (Chapter 6) *Relational and semantic structures to model question-answer pairs for answer passage reranking, answer sentence selection and answer extraction.*

In this chapter we elaborate further on the effectiveness of the relational linguistic structures for modelling text pairs and demonstrate their effectiveness on a set of QA tasks: answer passage reranking, answer sentence selection, and answer extraction. Again, the proposed approach relies on a kernel-based learning framework, where structural kernels, e.g., tree kernels, are used to greatly ease the feature engineering effort. It is enough to specify the desired type of structures, e.g., shallow, constituency, dependency trees, representing question and its candidate answer sentences and let the kernel learning framework learn to use discriminative tree fragments for the target task. An important feature of this approach is that it can effectively combine together different types of syntactic and semantic information, also generated by additional automatic classifiers, e.g., focus and question classifiers. We augment the basic structures with additional relational and semantic information by introducing special tag markers into the tree nodes.

In the final Chapter 7 we focus on the deep learning methods that treat words as vectors and have the benefit of making it possible to learn their optimal representation directly from the training data.

Contribution 5 (Chapter 7, Sec. 7.3) *State-of-the-art deep learning system for Twitter Sentiment Analysis*

In this chapter we demonstrate a deep learning architecture that obtains state-of-the-art results on Twitter Sentiment Analysis task. The core contribution is a process to initialize

the parameter weights of the convolutional neural network, which is crucial to train an accurate model while avoiding the need to inject any additional features. Briefly, we propose to use an unsupervised neural language model to initialize word embeddings that are further tuned by a deep learning model on a distant supervised corpus. At a final stage, the pre-trained parameters of the network are used to initialize the model which is then trained on the supervised training data from Semeval-2015. According to results on the official test sets, my model ranks 1st in the phrase-level subtask A (among 11 teams) and 2nd on the message-level subtask B (among 40 teams). Interestingly, computing an average rank over all six test sets (official and five progress test sets) puts our system 1st in both subtasks A and B.

Contribution 6 (Chapter 7, Sec. 7.4) *Deep learning architectures for modelling short text pairs.*

We present a deep learning architecture for modelling short text pairs: query-tweet and question-answer pairs. We use convolutional neural networks to learn the optimal representation of text pairs to establish their similarity. In addition to the input texts we feed the network with the information about words that overlap in a pair. Effectively, the word embeddings are augmented with additional parameters (dimensions) to represent the fact that words in a pair are related. These parameters are then tuned by the network. Intuitively, this idea is similar to the relational linking strategy that we explore when designing syntactic structures to model question-answer pairs within the tree-kernel framework. However, one important distinction is that treating overlapping words as an additional input to the network allows for tuning its representation on the training set. We empirically verify that modelling overlapping words in a pair allows the network to better capture the interactions between questions and answers resulting in a significant boost in the accuracy over the previous state-of-the-art approaches including previously proposed deep learning models.

Finally, Chapter 8 summarizes the contributions of this thesis and discusses potential avenues for the future work.

Publications

Research presented in this dissertation resulted in the following publications:

1. Aliaksei Severyn and Alessandro Moschitti. *Twitter Sentiment Analysis with Deep Convolutional Neural Networks.* Under submission, 2015

2. Aliaksei Severyn and Alessandro Moschitti. *Learning to Rank Short Text Pairs with Deep Convolutional Neural Networks.* Under submission, 2015

3. Aliaksei Severyn and Alessandro Moschitti. *Modelling Question-Answer Pairs with Convolutional Neural Networks.* Under submission, 2015
4. Aliaksei Severyn and Alessandro Moschitti. *UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification.* In Semeval 2015 (to appear)
5. Aliaksei Severyn and Alessandro Moschitti. *On the Automatic Learning of Sentiment Lexicons.* In NAACL 2015 (to appear)
6. Aliaksei Severyn, Alessandro Moschitti, Olga Uryupina, Barbara Plank, and Katja Filippova. *Multi-lingual Opinion Mining on YouTube.* In Information Processing Management (IPM) journal, 2015 (to appear)
7. Aliaksei Severyn, Alessandro Moschitti, Manos Tsagkias, Richard Berendsen and Maarten de Rijke. *A Syntax-Aware Re-ranker for Microblog Retrieval.* In SIGIR 2014
8. Aliaksei Severyn, Alessandro Moschitti, Olga Uryupina, Barbara Plank, and Katja Filippova. *Opinion Mining on YouTube.* In ACL 2014
9. Olga Uryupina, Barbara Plank, Aliaksei Severyn, Agata Rotondi, and Alessandro Moschitti. *SenTube: A Corpus for Sentiment Analysis on YouTube Social Media.* In LREC 2014
10. Kateryna Tymoshenko, Alessandro Moschitti and Aliaksei Severyn. *Encoding Semantic Resources in Syntactic Structures for Passage Reranking.* In EACL 2014
11. Aliaksei Severyn and Alessandro Moschitti. *Automatic Feature Engineering for Answer Selection and Extraction.* In EMNLP 2013
12. Aliaksei Severyn and Massimo Nicosia and Alessandro Moschitti. *Building Structures from Classifiers for Passage Reranking.* In CIKM 2013
13. Aliaksei Severyn and Massimo Nicosia and Alessandro Moschitti. *Learning Adaptable Patterns for Passage Reranking.* In CoNLL 2013
14. Aliaksei Severyn and Massimo Nicosia and Alessandro Moschitti. *Learning Semantic Textual Similarity with Structural Representations.* In ACL 2013
15. Aliaksei Severyn and Massimo Nicosia and Alessandro Moschitti. *iKernels-Core: Tree Kernel Learning for Textual Similarity.* In *SEM 2013
16. Aliaksei Severyn and Alessandro Moschitti. *Fast Linearization of Tree Kernels over Large-Scale Data.* In IJCAI 2013
17. Aliaksei Severyn and Alessandro Moschitti. *Structural Relationships for Large-Scale Learning of Answer Re-reranking.* In SIGIR 2012

18. Aliaksei Severyn and Alessandro Moschitti. *Fast Support Vector Machines for Convolution Tree Kernels.* In Data Mining and Knowledge Discovery (DMKD), Volume 25 (special issue), 2012

19. Aliaksei Severyn and Alessandro Moschitti. *Fast Support Vector Machines for Structural Kernels.* In ECML/PKDD 2011 [**Best Student Paper Award**]

20. Aliaksei Severyn and Alessandro Moschitti. *Large-Scale Learning with Structural Kernels for Class-Imbalanced Datasets.* In Communications in Computer and Information Science (CCIS), Springer 2011.

21. Aliaksei Severyn and Alessandro Moschitti. *Large-Scale Support Vector Learning with Structural Kernels.* In ECML/PKDD 2010

Competitions

The methods and techniques described in this thesis have been applied to develop the systems participating in the following competitions and challenges:

1. Twitter Sentiment Analysis (Task 10) at Semeval 2015 [129]
   Problem: 3-class Twitter Sentiment classification
   Results:

   Subtask A: rank **1st** out of 11 teams

   Subtask B: rank **2nd** out of 40 teams

2. Partly Sunny with a Chance of Hashtags at Kaggle 2013 [71]
   Problem: multi-output classification for Twitter Sentiment Analysis.
   Results: rank **1st** out of 259 teams

3. Semantic Textual Similarity shared task of *SEM 2013 [3]
   Problem: predict a real-valued score reflecting semantic similarity between text pairs
   Results: rank **21st** out of 88 submissions

Chapter 2

Preliminaries: Supervised Learning with Structural Kernels

In this chapter we introduce some of the most important concepts that are heavily used throughout this thesis and the reader is assumed to be familiar with them before proceeding to the next chapters. We briefly define the general problem of supervised learning starting from the definition of empirical risk minimization. We also formally define the classification, sequence labelling and reranking tasks and follow up with examples from NLP tasks that are directly related to this thesis. Next, we give a formal definition of an optimization problem solved by binary SVMs, and show how they can be kernelized to allow for learning non-linear decision boundaries. Finally, we define some of the most important tree kernel functions used in this thesis.

2.1 Supervised Learning

In this section we briefly overview some of the main concepts in supervised learning: specifying the decision function, the inference problem to assign labels to output variables, empirical risk minimization and a loss function.

In supervised learning, given a labeled training set $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$ the goal is to learn a decision function $h \in \mathcal{H}$ from some hypothesis space $\mathcal{H}$, that given an input $x \in \mathcal{X}$ assigns an output label $y \in \mathcal{Y}$: $h : \mathcal{X} \rightarrow \mathcal{Y}$.

Inference. The process of assigning a label to output variables can be formalized in the most general way as follows:

$$h(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} f_{\mathbf{w}}(\mathbf{x}, \mathbf{y}), \tag{2.1}$$

where $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a discriminant function parametrized by a vector of weights $\mathbf{w}$ that assigns a numerical score to the input-output variables. It can be also thought of as a compatibility function measuring the degree to which input and output variables agree. Typically, the discriminant function f is chosen to be linear in the vector of model parameters $\mathbf{w}$:

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle, \tag{2.2}$$

where $\langle \cdot, \cdot \rangle$ computes a dot product and $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ computes a joint feature representation of the input example pairs $(\mathbf{x}, \mathbf{y})$ by mapping them into some feature space $\mathbb{R}^d$.

Empirical Risk Minimization. Having fixed the hypothesis space $\mathcal{H}$ of possible decision functions to have a linear form (as defined by the scoring function from Eq. 2.2), the remaining question is how to estimate the model parameters $\mathbf{w}$ from the training data.

In supervised learning we consider machine learning methods that perform estimation of the decision function by minimizing a task-dependent loss function evaluated on the training data. This is formalized by the Empirical Risk Minimization principle (regularized) [140]:

$$h^* = \operatorname*{arg\,min}_{h \in \mathcal{H}} R(f) = \operatorname*{arg\,min}_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(\mathbf{x}_i), \mathbf{y}_i) + \lambda \Omega(\mathbf{w}) \tag{2.3}$$

where L is task-dependent loss function which computes the cost of predicting $\mathbf{y}$ when the correct label is $\mathbf{y}^*$ and Ω is a regularization term whose goal is to penalize complex model to prevent overfitting. One of the most common choice for the regularization term is the squared L_2-norm:

$$\Omega(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2. \tag{2.4}$$

Loss function. Loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ specifies how much we need to penalize incorrect predictions. Amongst the most commonly used loss functions is a hinge loss, which is an upper-bound approximation of the standard zero-one loss used in classification. In its most general form it can be formulated as follows:

$$L_{\text{hinge}}(\mathbf{y}^*, \mathbf{y}) = \max\Big(0, \max_{\mathbf{y} \neq \mathbf{y}^*} (\Delta(\mathbf{y}, \mathbf{y}^*) + \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle) - \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}^*) \rangle\Big), \tag{2.5}$$

where $\Delta(\cdot, \cdot)$ measures the discrepancy between the predicted label $\mathbf{y}$ and the true output $\mathbf{y}^*$. It encodes the requirement that the true output $\mathbf{y}^*$ should score higher than any other predicted value $\mathbf{y}$ by at least $\Delta(\mathbf{y}, \mathbf{y}^*)$.

Hence, to fully specify a supervised learning problem it is required to define:

1. A joint feature map $\Psi(\cdot, \cdot)$ that maps input-output variables into some joint feature space $\mathbb{R}^d$. In other words, it is a feature engineering process which plays a crucial role in the design of machine learning systems. The expressiveness of the feature sets that encode input examples directly affects the accuracy of the learned models. It is largely empirical-driven, labour-intensive and a highly task-dependent process. In the later sections we will see how kernels may automate this process by generating implicit feature spaces of high-dimensionality and letting the learning algorithm select the relevant features.

2. An inference process of assigning output labels $\mathbf{y}$ to the inputs $\mathbf{x}$ (Eq. 2.1). In simple cases where the cardinality of the output space $\mathcal{Y}$ is small we can simply enumerate all possible $\mathbf{y}$ choosing the one that results in the maximum score. However, in cases when $\mathcal{Y}$ is more complex it often becomes computationally intractable to enumerate all possible output values as the output space can be exponentially large. Hence, more efficient inference procedures are required. Typically, in cases when $\mathcal{Y}$ is complex, to make the inference problem tractable the feature map $\Psi()$ is chosen to decompose into a set of local features.

3. A task-dependent loss function L that estimates how much we need to penalize incorrect predictions. To make the inference problem tractable, the $\Delta(\cdot, \cdot)$ is required to also decompose in the same fashion as the feature map $\Psi()$.

4. The form of a regularizer $\Omega(\cdot)$.

5. An optimization algorithm to estimate the model parameters $\mathbf{w}$ by minimizing the regularized empirical risk from Eq. 2.3.

All of the above choices lead to different learning algorithms. Yet a single factor that drives all of the above decisions and actually defines the type of the learning problem is the structure and the size of the output space $\mathcal{Y}$. In the following we consider the main types of supervised learning problems exemplifying with tasks from NLP that are directly related to this thesis.

2.2 Supervised learning problems in NLP

In this section we briefly overview a set of problem classes that are most common in NLP covering classification, reranking and structured output prediction. For each of the problem classes we also briefly describe specific tasks that are either directly targeted in this thesis, or, otherwise, used as downstream components in our pipeline (primarily structured output prediction). Our primary goal is not on the solutions to these problems but rather on giving a brief description of their nature and framing them into a corresponding class of machine learning problems. For the tasks that are directly tackled in

this thesis, their description and our solutions will be described in much more detail in the corresponding chapters.

2.2.1 Classification

Tasks that can be treated as a classification problems are extremely common in NLP whether it's a simple document classification problem or a more complex task of learning to match text pairs in Semantic Textual Similarity.

Problem formulation

In binary classification the output domain is simply $\mathcal{Y} = \{-1, +1\}$ and for multi-class classification $\mathcal{Y} = \{1, .., K\}$ for small number of classes K. The inference problem can be solved by exhaustive enumeration of output variables y and choosing the one that results in the maximum score. The joint feature map for binary classification is simply $\Psi(\mathbf{x}) = y\Phi(\mathbf{x})$ where $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ denotes an arbitrary feature representation of inputs. For multi-class $\Psi(\mathbf{x}) = \Lambda^k \otimes \Phi(\mathbf{x})$, where Λ^k refers to the binary encoding of the k-th label y and $\otimes$ is the tensor product which forms all products between the two argument vectors. A typical choice for the loss function in classification tasks is the hinge loss, whose general form is given by Eq. 2.3. For binary classification it simplifies to the following: $\max(0, 1 - yf(\mathbf{w}, \mathbf{x}))$, and $\max(0, 1 + \max_{y \neq y^*} f(\mathbf{w}_y, \mathbf{x}) - f(\mathbf{w}_{y^*}, \mathbf{x}))$ for multi-class classification.

Hence, in these type of tasks to learn accurate models that produce decision function with high discriminative power the main effort lies in how well we can model the structure of the input data.

Example tasks

Question Classification. In factoid Question Answering knowing the question category, i.e., what is being asked, is essential to direct the focus of the information retrieval component to recall relevant documents. Given a question, the goal of QC task is to classify it in one of the several predefined question categories, e.g., Descriptions (e.g., definitions or explanations), Human (e.g., group or individual), Location (e.g., cities or countries), Numeric (e.g., amounts or dates), etc. These categories can be used to determine the Expected Answer Type for a given question and find the appropriate entities found in the candidate answers. Imposing such constraints on the potential answer types greatly reduces the search space in the downstream QA pipeline. We consider the problem of QC using tree kernel learning framework in a larger context of more involved QA reranking tasks in Ch. 6.

Focus Classification. The question focus is typically a simple noun representing the entity or property being sought by the question. It can be used to search for semantically compatible candidate answers in document passages, thus greatly reducing the search space. For example, in a question: *"What is the real Mark Twain's name?"* the question category is *Human*, while the word *name* is the question focus and helps to further specify the type of the answer phrase the correct answer candidates should contain. We build a focus identification classifier as a submodule of our QA pipeline in Ch. 6.

Sentiment Analysis. Sentiment Analysis is being increasingly used to automatically recognize opinions about products in natural language texts. The goal of sentiment analysis is to identify a sentiment expressed by a user in a given text. It can be performed at various levels of granularity, e.g., classifying the sentiment expressed by a document, sentence, or individual words or phrases. A user might be interested not only in identifying the polarity of a text but also to identify the subject (who is expressing an opinion) and its object (what item or product is the focus of the expressed opinion). The target of the prediction can be binary, e.g., *positive/negative*, or multi-class, e.g. including additional classes, such as *neutral* or even introducing additional classes to make the *positive/negative* more fine grained, etc.

We tackle the problem of Sentiment Analysis of YouTube comments using tree kernels in Ch. 4 and design a state-of-the-art deep learning system for identifying sentiment polarity of tweets in Sec. 7.3.

Semantic Textual Similarity. The goal of the Semantic Textual Similarity (STS) task is to learn a scoring model that given a pair of two texts returns a similarity score that correlates with human judgement of how semantically close these texts are. Hence, the key aspect of having an accurate STS framework is the design of features that can adequately represent various aspects of the similarity between texts, e.g., using lexical, syntactic and semantic similarity metrics. We address the STS problem in Sec. 5.2.

2.2.2 Re-ranking

In this section we formally describe the problem of reranking text pairs which encompasses a large set of tasks in IR and Question Answering, e.g., reranking candidate answer passages, answer sentence selection, microblog retrieval, etc. While several effective solutions to the reranking problems exist, we argue that deriving an efficient representation of query-document pairs that are then passed to a learning to rank algorithm often plays a far important role in training an accurate model.

Problem formulation

The most typical setup in supervised learning-to-rank tasks is as follows: we are given a set of retrieved lists, where each query $\mathbf{q}_i \in \mathbf{Q}$ comes together with its list of candidate documents $\mathbf{D}_i = \{\mathbf{d}_{i1}, \mathbf{d}_{i2}, \ldots, \mathbf{d}_{in}\}$. The candidate set comes with their relevancy judgements $\{y_{i1}, y_{i2}, \ldots, y_n\}$, where, relevant documents have labels equal to 1 (or higher to reflect the degree of relevancy) and 0 otherwise. The goal is to build a model that for each query $\mathbf{q}_i$ and its candidate list $\mathbf{D}_i$ generates an optimal ranking $\mathbf{R} = (r_{i1}, r_{i2}, \ldots, r_{in})$, s.t. relevant documents appear at the top of the list. Each r_{ij} specifies the position of the document d_{ij} in the reranked list.

More formally, the task is to learn a ranking function in the general form:

$$h(\mathbf{w}, \Phi(\mathbf{q}_i, \mathbf{D}_i)) \rightarrow \mathbf{R}, \tag{2.6}$$

where function $\Phi(\cdot)$ maps input query-document pairs to a feature vector representation where each component reflects a certain type of their relatedness, e.g., lexical, syntactic, and semantic. The weight vector $\mathbf{w}$ is a parameter of the model and is learned during the training.

Learning to Rank approaches

There are three most common approaches to learn the ranking function h referred to as pointwise, pairwise and listwise methods.

Pointwise approach is perhaps the most simple way to solve the reranking problem. It is treated as a binary classification problem, where each triple $(\mathbf{q}_i, \mathbf{d}_{ij}, y_{ij})$ represents a training instance and it is enough to train a classifier: $h(\mathbf{w}, \Phi(\mathbf{q}_i, \mathbf{d}_{ij})) \rightarrow y_{ij}$.

The decision function $h(\cdot)$ typically takes a linear form simply computing a dot product between the model weights $\mathbf{w}$ and a feature representation generated by $\Phi(\cdot)$. At test time, the learned model is used to classify unseen pairs $(\mathbf{q}_i, \mathbf{d}_{ij})$, where the raw scores are used to establish the global rank $\mathbf{R}$ of the documents in the retrieved set. This approach is widely used in practice because of its simplicity and effectiveness.

Pairwise approach. A more advanced approaches to reranking, is pairwise, where the model is explicitly trained to score correct pairs higher than incorrect pairs with a certain margin:

$$\langle \mathbf{w}, \Phi(\mathbf{q}_i, \mathbf{d}_{ij}) \rangle \geq \langle \mathbf{w}, \Phi(\mathbf{q}_i, \mathbf{d}_{ik}) \rangle + \epsilon,$$

where document $\mathbf{d}_{ij}$ is relevant and $\mathbf{d}_{ik}$ is not. Conceptually similar to the pointwise method described above, pairwise approach exploits more information about the ground truth labeling of the input candidates. However, it requires to consider a larger number of training instances (potentially quadratic in the size of the candidate document set) than the pointwise method, which may lead to slower training times. However, both pointwise

and pairwise approaches ignore the fact that ranking is a prediction task on a list of objects.

Listwise approach. The third method, referred to as a listwise approach [24], treats a query with its list of candidates as a single instance in learning, thus able to capture considerably more information about the ground truth ordering of input candidates. However, the inference process is more computationally expensive.

While listwise approach claim to yield better performance, it is more complicated to implement and less effective train. Most often, producing a better representation $\Phi()$ that encodes various aspects of similarity between the input query-document pairs plays a far more important role in training an accurate reranker than choosing between different ranking approaches.

Example tasks

Discriminative reranking has a long standing history in NLP. Perhaps its most successful application across numerous NLP tasks was the idea to rerank the k-best output of probabilistic generative systems thus boosting their accuracy. This type of reranking has proven especially useful in developing accurate syntactic parsers for Named Entity Recognition, Semantic Role Labeling, Constituency parsing, Machine Translation, etc.

Another popular scenario of applying reranking stems from IR applications, where the output of retrieval components, e.g., search engines, is refined by learning to rank modules. Improving the accuracy of Question Answering systems using discriminative reranking is perhaps one of the best examples of blending retrieval and supervised machine learning.

In this thesis we focus on applications of reranking for Question Answering and Microblog Retrieval in Chapter 6. Given a query with its associated list of candidates our goal is to learn a decision function (Eq. 2.6) that scores input candidates that are relevant higher than those that are not relevant.

We adopt *pointwise* (Sec. 5.3, 6.5.3, 7.4) and *pairwise* (Sec. 6.5) ranking methods and focus primarily on modelling rich representations of text pairs.

2.2.3 Structured Prediction Problems in NLP

Although not directly addressed in this thesis, the problem of structured output prediction is of key importance to NLP. In fact, the resulting syntactic parsers are used to derive various syntactic representations of the input examples that are heavily exploited in this thesis.

In the following, our goal is not to describe the solutions to various structured output prediction tasks in NLP but to mainly introduce the resulting syntactic structures that encode our knowledge of how the language is structured. In particular, we briefly describe two types of structured output prediction problem types where the output is a sequence

of labels or a tree structure. To exemplify the former, we consider part-of-speech tagging, shallow syntactic parsing (chunking), Named Entity Recognition (NER) and Semantic Role Labeling (SRL). The latter problem type where the output structures are trees is primarily addressed by dependency and constituency parsing.

Problem formulation

In sequence labeling problem, for a given input sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ of size m the goal is to predict a label sequence $\mathbf{y} = (y_1, \ldots, y_m)$. Denoting Σ to be a set of possible labels for each y_i the output space of all possible label assignments to our input sequence is $\mathcal{Y} = \Sigma^m$. As we can see the size $\mathcal{Y}$ is exponential in the size of the input sequence, thus rendering the possibility of treating it as a multi-class classification problem impractical. While it is possible to predict each individual label y_i independently, thus effectively reducing this problem to a simple classification problem, we are loosing the ability to capture interactions between the output variables, which appears to be quite important to train accurate models. The importance of modelling interactions between the output variables has been verified by the state-of-the-art systems for most of the sequence labeling tasks, e.g., chunking, named-entity-recognition, semantic role labeling, etc.

Tasks where the output structure is a tree, hence the output space $\mathcal{Y}$ is a set of all valid trees for considered inputs, represent an even more complex problem class, where effective inference methods are crucial to render the problem computationally tractable.

Learning to output complex structures in NLP, e.g., sequences and trees, have been tackled using a wide variety of approaches ranging from rule-based to discriminatively trained machine learning models, among which some of the most widely used in NLP are Hidden Markov Models [119], Conditional Random Fields [81], Perceptron [32], Maximum Entropy models [17], SVMs and Structured SVMs [166], and Max Margin Markov Networks [160], etc.

In the following we briefly describe part-of-speech tagging, shallow syntactic parsing and Named Entity Recognition (NER), Semantic Role Labeling (SRL), constituency and dependency based trees.

Example tasks

Part-of-speech tagging is the process of assigning a part-of-speech tag to each word in a sentence such as noun, verb, adjective, preposition, etc. For example, given an input sentence: *"heat water in large vessel"* the POS-tagger produces the following tagged output: $\texttt{[heat]}_{\texttt{verb}}$ $\texttt{[water]}_{\texttt{noun}}$ $\texttt{[in]}_{\texttt{prep}}$ $\texttt{[a]}_{\texttt{det}}$ $\texttt{[large]}_{\texttt{adj}}$ $\texttt{[vessel]}_{\texttt{noun}}$. One of the biggest problems in POS tagging is the word ambiguity, i.e., depending on the context words may act as different parts of speech. In the previous example, the word `heat` is a verb, but can be a noun in another context, e.g., *today the sun heat is intense.* The set

of tags assigned by a POS tagger typically contains tens to hundreds of tags. The most commonly used set of POS tags for English is the Penn TreeBank tagset which includes 36 tags.

Current state-of-the-art part-of-speech taggers can process many thousands of tokens per second with per-token accuracies around 97%. The information provided by POS tagging is useful in many applications, e.g., Information Retrieval, Speech processing, Word Sense Disambiguation, Machine Translation, Question Answering, etc. There is a large number of freely available POS tagger.[1]

In this thesis, the POS tags are used ubiquitously to build syntactic representations that are the focus of Chapters 4, 5, and 6.

Shallow syntactic parsing or *chunking* seeks to divide a text into segments which correspond to certain syntactic units such as noun (NP), verb (VP) or prepositional phrases (PP). The idea is to identify syntactically related words and group them into non-overlapping syntactic units. An example output of the chunker for the following sentence: `[Mike]`$_{\texttt{NP}}$ `[would join]`$_{\texttt{VP}}$ `us [this evening]`$_{\texttt{NP}}$`.`

Chunks are represented as groups of words denoted by the squared brackets and are associated with a tag indicating their syntactic category. The chunking task is typically framed as a sequence labeling problem using BIO scheme, i.e., each word in a sentence is labeled with a tag starting with `B-` denoting the start of a chunk, `I-` if the word is inside a previously detected chunk, and `O` for words outside of any chunks.

Chunking is an intermediate step towards full parsing, being much faster and more robust. The information provided by a chunker is useful in many NLP tasks, such as information extraction, text summarization, machine translation, etc. According to the results on test set from CoNLL chunking shared task in 2000 [163], the state-of-the-art accuracy is about 94% in F1-score.

In this thesis, the output of the chunker is one of the key components of our syntactic tree representations described in Ch. 4, 5, and 6.

Named Entity Recognition is the task of locating a word or a phrase that references a particular entity. Typical types of detected named entities are the names of persons, organizations and locations.

For example the following output of a NER tagger: `[U.N.]`$_{\texttt{ORG}}$ `official [Ekeus]`$_{\texttt{PER}}$ `heads for [Baghdad]`$_{\texttt{LOC}}$ contains three named entities: `Ekeus` is a person, `U.N.` is a organization and `Baghdad` is a location.

Similar to chunking, the task is typically framed as a sequence labeling problem using BIO scheme. Features extracted from POS tags and chunking information are heavily exploited by NER systems.

[1] a comprehensive list of POS taggers at http://nlp.stanford.edu/links/statnlp.html#Taggers

Named entity recognition is an important subtask of information extraction systems and is widely used in Information Retrieval, Question Answering. According to the results on test set from NER task at CoNLL-2003 [164], the state-of-the-art F1 score for NER taggers is about 90%.

In this thesis, we use the NER tagger to build our relational syntactic tree models for Question Answering in Chapter 6.

Semantic Role Labeling or *shallow semantic parsing* annotates phrases of a sentence with semantic roles with respect to a target predicate (verb). Typical semantic arguments are Agent, Patient, Instrument, etc. For example, the output of an SRL tagger: `[John]`$_{\texttt{A0}}$ `[ate]`$_{\texttt{V}}$ `[the apple]`$_{\texttt{A1}}$, where `ate` is a verb, while `John` is an argument of type 0 meaning it is an acceptor and `the apple` is tagged with `A1` (thing accepted).

Recognizing and labeling semantic arguments is a key task in Information Extraction, Question Answering, Summarization, many other NLP tasks where semantic interpretation is needed. SRL is a far more complex task, and according to the results on the test set from CoNLL-2005 SRL shared task [25], the state-of-the-art accuracy is about 78% in F1-score.

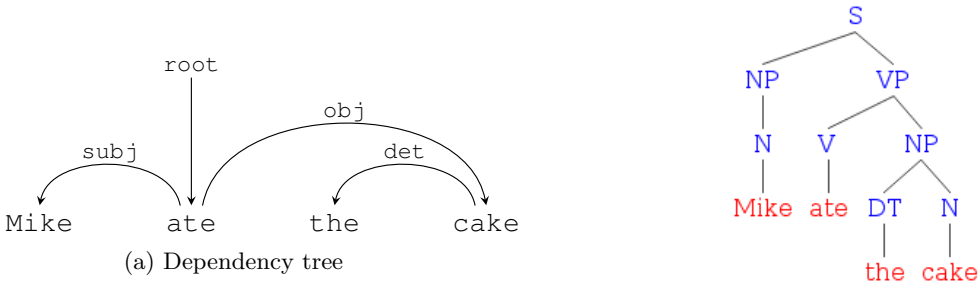
While representing a sequence labeling task, state-of-the-art approaches to SRL often treat it as a two-stage problem: first identifying boundaries of predicate arguments and then classifying them with a corresponding semantic role. State-of-the-art SRL approaches rely on the output of syntactic constituency parsers to identify argument boundaries. Once the predicate arguments are identified assigning them with a semantic role is essentially a classification problem.

In this thesis we address the first step of boundary identification of predicate arguments for SRL and show how tree kernel approaches can achieve state-of-the-art accuracy (Chapter 3).

Constituency parsing. The idea behind generating constituency parse trees is to hierarchically organize words in a sentence into nested constituents. To model the constituency structure parsers rely on a set of rules specified by a context-free grammar. The constituency parse tree conveys several layers of information: POS tags of individual words, the grouping of words into phrases, interactions between constituents which captures their syntactic/semantic role.

Given that using constituency tree parsers is a rather computationally expensive preprocessing step (typical complexity is $O(n^3)$), in this thesis we use it to build a syntactic tree model only for Semantic Textual Similarity in Sec 5.2.

Dependency parsing. A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between head words and words which modify those



(a) Dependency tree

(b) Constituency tree

Figure 2.1: Example syntactic trees.

heads. An example of labeled dependency parse tree is shown in Fig. 2.1a. The arrows indicate the dependency relations between words (starting from the head word going to the modified word) while the labels specify the exact nature of the dependency. For example, in the Fig. 2.1a word `ate` is the head word of the word `Mike` linked by the `subj` dependency type.

In the subsequent section we introduce a kernelized binary SVM classifier that can handle classification, regression and ranking problems that are considered in this thesis. In turn, the syntactic parsers described in this section are used to construct syntactic structures used in the SVM tree kernel framework for automatically generating rich feature spaces to ease the tedious task of manual feature engineering.

In this thesis we consider the use of dependency structures for Semantic Textual Similarity task in Sec 5.2.

2.3 Binary Support Vector Machines

One of the most prominent classes of supervised learning algorithms are Support Vector Machines that enable the use of kernels. SVMs are based on the margin maximization principle and enable the use of high-dimensional feature spaces via the "kernel trick". SVMs come with strong generalization guarantees and have demonstrated state-of-the-art performance in many tasks including document categorization, optical character recognition, image classification and many more. As previously mentioned, binary SVMs can handle classification, regression and ranking problems.

In the following, we provide a brief formulation of an SVM optimization problem and show how it can be kernelized to allow for learning non-linear decision boundaries.

2.3.1 Primal formulation

In fact, SVMs are a class of learning algorithms instantiated from the more general supervised learning framework described in Sec. 2.1.

Binary SVMs deal with the output space that is $\mathcal{Y} = \{-1, +1\}$. The joint feature map reduces to a product of the output label and the input feature map: $\Psi(\mathbf{x}, y) = y\Phi(\mathbf{x})$. For the moment we consider a simple case of linear SVMs where the input feature map is an identity $\Phi(\mathbf{x}) = \mathbf{x}$. Hence, binary SVMs estimate the following linear discriminant function[2]:

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle, \tag{2.7}$$

where $\mathbf{w} \in \mathbf{R}^d$ is a vector of model parameters. In this case, the inference procedure is simply $y^* = sign(\langle \mathbf{w}, \mathbf{x} \rangle)$. One of the distinctive characteristics of SVMs is the maximum-margin principle derived from the geometric intuition that examples of different classes should be separated by a decision function with a large margin. In the formalism of general supervised learning from Sec. 2.1 this aspect is captured by using the L2-norm regularizer.

The hinge loss defined in Eq. 2.5 for binary classification takes the form: $L(y) = \max(0, 1 - y f_{\mathbf{w}}(\mathbf{x})))$. Hence, in lieu of Empirical Risk Minimization principle (Eq. 2.3, SVM estimates the weight vector $\mathbf{w}$ by solving the following optimization problem, e.g., [140]:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \max(0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle) \tag{2.8}$$

where the first term in the objective function is a regularizer encoding the maximum-margin principle and the second term represents the empirical loss incurred on the training set. The margin trade-off parameter C controls the balance between the regularization term and the empirical loss. Equivalently, Eq. 2.8 can reformulated as the following constrained optimization problem:

$$\begin{aligned} \underset{\mathbf{w},\xi}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \xi_i \\ \text{subject to} \quad & y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{2.9}$$

The constraints enforce the requirement to classify the training examples with a minimum margin. The slack variables ξ_i allow for violations in classification, which is essential in practice when dealing with noisy data.

2.3.2 The dual

The SVM optimization problem is often more convenient to solve in the dual, which also, as we will see later, allows for using kernels. To derive the dual formulation, the

[2] note that the bias b can be integrated into a feature vector $\mathbf{x}$ that is always on

Lagrangian of the primal problem (3.1) is computed as:

$$L_P(\mathbf{w}, \xi, \alpha, \lambda) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi - \sum_i \alpha_i(1 - y_i\langle \mathbf{w}, \mathbf{x}_i\rangle - \xi_i) - \sum_i \lambda_i \xi_i, \tag{2.10}$$

where α and λ are the Lagrange multipliers.

Using the fact that both gradients of L with respect to $\mathbf{w}$ and ξ vanish at the optimum:

$$\begin{aligned} \frac{\delta L_P}{\delta \mathbf{w}} = 0 \quad &\Rightarrow w = \sum_i \alpha_i y_i \mathbf{x}_i \\ \frac{\delta L_P}{\delta \xi} = 0 \quad &\Rightarrow \alpha_i + \lambda_i - C = 0, \end{aligned} \tag{2.11}$$

and by substituting variables from (2.11) into (2.10), we obtain the dual Lagrangian:

$$L_D(\alpha, \lambda) = \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle + \sum_i \alpha_i \tag{2.12}$$

Now we can state the dual variant of the optimization problem (3.1):

$$\begin{aligned} \underset{\mathbf{a}\geq 0}{\text{maximize}} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \end{aligned} \tag{2.13}$$

Using the first equation from (2.11), we get the connection between the primal and dual variables:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \tag{2.14}$$

The classifier decision function is now expressed through the dual variables:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x}\rangle \tag{2.15}$$

There a few key observations we can make about Eq. 2.15:

- First, we note that the form of the decision function in Eq. 2.15 that we derived directly from the SVM optimization problem using Lagrangian multipliers is a direct result of a particular instance of Representer's theorem [140]. It postulates that for a large class of algorithms minimizing a sum of an empirical risk term and a regularization term the optimal solutions can be written as kernel expansions in terms of training examples.
- Compared to the primal form of the equivalent decision function from Eq. 2.7 we replaced $\mathbf{w}$ (whose dimensionality is equal to the dimensionality of the input space

$\mathcal{X}$) with dual variables α_i that are independent of the dimensionality of the input space and are used to weight training examples.

- Examples with $\alpha_i \geq 0$ are called support vectors, hence the name Support Vector Machines. In practice, it is often the case that a large portion of the α_i are zeros, hence the produced solution is often sparse.
- The training instances $\mathbf{x}$ in both optimization problem (Eq. 2.13) and decision function (Eq. 2.15) appear only in the form of the inner products.
- While the decision function has a linear form, it is straight-forward to make it non-linear by transforming input examples from their original feature space into another other space via a non-linear feature map $\phi(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^{d'}$. To increase the discriminative capacity of the classifier the dimensionality of the transformed feature space d' is chosen to be $d' \gg d$ and can be potentially infinite.

The last point above is the key to making SVMs able to learn non-linear decision boundaries via non-linear feature transforms. The kernel trick is described next.

2.3.3 The kernel trick

As previously noted SVMs use the data only through inner products. Because of this, they can be made non-linear in a very general way via feature maps $\phi(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^{d'}$:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle$$

In fact, the non-linear map $\phi(\cdot)$ doesn't have to be computed explicitly. Instead of mapping our data via $\phi(\cdot)$ and computing the inner product, we can do it in one operation, leaving the mapping completely implicit. This is a direct result of the Mercer's Theorem (for example, see [140]), which states the following: a symmetric function $K(\mathbf{x}_i, \mathbf{x}_j)$ can be expressed as an inner product

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \tag{2.16}$$

for some $\phi()$ if and only if $K(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite, i.e.

$$\sum_{i,j} K(\mathbf{x_i}, \mathbf{x}_j) c_i c_j \geq 0, \forall \mathbf{c}$$

This means that we do not need to know an explicit form of $\phi()$, as all we need to know is how to compute the inner product. A function that takes as input pairs of objects and outputs an inner product between them in some feature space, while satisfying the conditions of the Mercer's theorem, is called a kernel function.

Hence, the decision function of the kernelized SVMs can be expressed as a kernel expansion:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \tag{2.17}$$

The choice of a kernel function K depends on the task and the type of input data and is up to the user. Some examples of widely used kernel functions are:

- Polynomial of degree p: $K(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^p$
- Sigmoid: $K(\mathbf{x}, \mathbf{y}) = \tanh(a\langle \mathbf{x}, \mathbf{y} \rangle + b)$
- Gaussian RBF: $K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2)$

So far we considered cases where $\mathbf{x}$ is a vector of fixed dimensionality, i.e., is a list of real-valued numbers. However, given that a kernel function is only required to provide an inner product between objects in some implicit feature space, kernels can be easily applied to other types of input objects, e.g., histograms, strings, trees, graphs, etc. In the next section we define kernels useful for NLP tasks that need to operate on textual inputs, e.g., strings and tree structures.

2.4 Structural Kernels

Structural kernels have been successfully applied to ease the design of machine learning systems in diverse domains, ranging from bioinformatics [78,82,130] and data-mining [10, 30,139,162,173,181,184,191] to Natural Language Processing (NLP) [23,34,37,73,79,142]. They free oneself from tedious manual feature engineering by automatically generating huge feature spaces assuming that the learning algorithm will end up using the most relevant features for a given task. This is especially useful when designing models for domains where no expert knowledge is easily available with respect to which features are most useful for a given problem.

In this section we introduce some of the most general types of structural kernels used in NLP and throughout this thesis: string kernels (SKs) [140], the Syntactic Tree Kernels (STKs) [32] and the Partial Tree Kernels (PTKs) [93].

2.4.1 String Kernels

The String Kernels (SK) that we consider count the number of subsequences shared by two strings of symbols, s_1 and s_2. Some symbols during the matching process can be skipped. This modifies the weight associated with the target substrings as shown by the following SK equation:

$$K_{\text{SK}}(s_1, s_2) = \sum_{u \in \Sigma^*} \phi_u(s_1) \cdot \phi_u(s_2) = \sum_{u \in \Sigma^*} \sum_{\vec{I}_1 : u = s_1[\vec{I}_1]} \sum_{\vec{I}_2 : u = s_2[\vec{I}_2]} \lambda^{d(\vec{I}_1) + d(\vec{I}_2)} \tag{2.18}$$

where, $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ is the set of all strings, $\vec{I}_1$ and $\vec{I}_2$ are two sequences of indexes $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |s|$, such that $u = s_{i_1}..s_{i_{|u|}}$, $d(\vec{I}) = i_{|u|} - i_1 + 1$ (distance between the first and last character) and $\lambda \in [0, 1]$ is a decay factor.

It is worth noting that: (a) longer subsequences receive lower weights; (b) some characters can be omitted, i.e., gaps; (c) gaps determine a weight since the exponent of λ is the number of characters and gaps between the first and last character; and (c) the complexity of the SK computation is $O(mnp)$ [140], where m and n are the lengths of the two strings, respectively and p is the length of the largest subsequence we want to consider.

SK applied to a sequence can derive useful dependencies between its elements. For example, for the following sentence: *"Is movie theater popcorn vegan?"*, we can define the sequence `[[is] [movie] [theater] [popcorn] [vegan]]`, which generates the subsequences, `[[is] [movie]]`, `[[is] [theater] [vegan]]`, `[[is] [vegan]]`, `[[ movie] [popcorn] [vegan]]` and so on. Note that this corresponds to a language model over words using skip n-grams.

2.4.2 Convolutional Tree Kernels

A tree kernel function detects if a tree subpart (common to both trees) belongs to the feature space that we intend to generate. For such purpose, the desired fragments need to be described and efficiently computed. We consider two important convolutional tree kernel functions: the syntactic tree kernel (STK) and the partial tree kernel (PTK).

Counting shared fragments

Convolution TKs compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. For this purpose, let the set $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ be the space of substructures and $\chi_i(n)$ be an indicator function, equal to 1 if the target t_i is rooted at a node n and equal to 0 otherwise. A tree-kernel function over T_1 and T_2 is

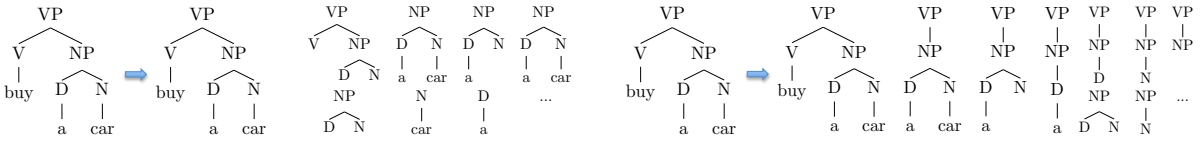
$$K_{\text{TK}}(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2), \tag{2.19}$$

where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{T}|} \chi_i(n_1)\chi_i(n_2). \tag{2.20}$$

which computes the number of common fragments rooted in the n_1 and n_2 nodes.

Of course, the number above depends on how fragments are defined. Indeed, there are



(a) SubTree (ST) fragments

(b) Partial Tree fragments (PTFs)

Figure 2.2: Example tree fragments generated by Tree Kernels.

different types as described in [93]. We consider three important characterizations: (i) the *subtrees*, (ii) the syntactic tree fragments (STFs) and (iii) the partial tree fragments (PTFs). These three types of tree fragments determine three different kernel functions.

A *subtree* rooted in a node n of a target tree T is a substructure that includes n with all of its descendants. A generalization of subtrees are STFs that do not necessarily include all the descendants of n, although each of its nodes contain exactly the same edges of T. For example, Figure 2.2a shows 10 STFs (out of total 17) of the subtree rooted in `VP` (of the left tree). In phrase structure syntactic trees, the constraint on the edges, is equivalent to impose that grammatical rules cannot be broken[3]. For example, `[VP [V NP]]` is an STF, which has two non-terminal symbols, `V` and `NP`, as leaves whereas `[VP [V]]` is not an STF, i.e. the rule `VP->V NP` can not be split.

If we relax such constraint, we obtain more general substructures called PTFs. These can be generated by the application of partial production rules of the grammar, consequently `[VP [V]]` and `[VP [NP]]` are valid PTFs. More in general, nodes in PTFs can have any subset of edges that had in T. This means that PTFs are not constrained to any grammar and can be applied to any tree structure of any domain. Figure 2.2b shows that the number of PTFs derived from the same tree as before is higher (i.e. 30 PTs).

Syntactic Tree Kernel (STK)

The Δ function depends on the type of fragments that we consider as *basic* features. To evaluate the number of STFs, we can use the following algorithm:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminal symbols) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j)) \tag{2.21}$$

[3]Any tree can be seen as generated by an underlying grammar where the production rules are given by a node (left hand side) and its children (right hand side).

where $nc(n_1)$ is the number of children of n_1 and c_n^j is the j-th child of the node n. Note that, since the productions are the same, $nc(n_1) = nc(n_2)$.

$\Delta(n_1, n_2)$ evaluates the number of STFs common to n_1 and n_2 as proved in [32]. Moreover, a decay factor λ can be added by modifying steps (2) and (3) as follows[4]:

2. $\Delta(n_1, n_2) = \lambda$,
3. $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$.

The computational complexity of STK is $O(|N_{T_1}| \times |N_{T_2}|)$ but as shown in [94], the average running time tends to be linear, i.e. $O(|N_{T_1}| + |N_{T_2}|)$, for natural language syntactic trees.

It should be noted that STK was devised for processing syntactic trees. Its main characteristic is that the production rules of the grammar used to generate the tree will not be *broken* to generate fragments. This corresponds to the restriction of not separating children in the related substructures (i.e. STFs). STK can be applied to compute the similarity measure between any trees (not necessarily syntactic parse trees) with a restriction that any node of the generated STF must still contain all (or none) of its children.

Partial Tree Kernel (PTK)

The computation of PTFs is carried out by the Δ function defined as follows:

1. if the node labels of n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. else:

$$\Delta(n_1, n_2) = 1 + \sum_{\mathbf{I}_1, \mathbf{I}_2, l(\mathbf{I}_1) = l(\mathbf{I}_2)} \prod_{j=1}^{l(\mathbf{I}_1)} \Delta(c_{n_1}(\mathbf{I}_{1j}), c_{n_2}(\mathbf{I}_{2j}))$$

where $\mathbf{I}_1 = \langle h_1, h_2, h_3, .. \rangle$ and $\mathbf{I}_2 = \langle k_1, k_2, k_3, .. \rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively. $\mathbf{I}_{1j}$ and $\mathbf{I}_{2j}$ point to the j-th child in the corresponding sequence, and, again, $l(\cdot)$ returns the sequence length, i.e. the number of children.

Furthermore, we add two decay factors: μ for the depth of the tree and λ for the length of the child subsequences with respect to the original sequence, which accounts for gaps. Hence, the expression for the Δ function for PTK derives as follows:

$$\Delta(n_1, n_2) = \mu \Big(\lambda^2 + \sum_{\mathbf{I}_1, \mathbf{I}_2, l(\mathbf{I}_1) = l(\mathbf{I}_2)} \lambda^{d(\mathbf{I}_1) + d(\mathbf{I}_2)} \prod_{j=1}^{l(\mathbf{I}_1)} \Delta(c_{n_1}(\mathbf{I}_{1j}), c_{n_2}(\mathbf{I}_{2j})) \Big), \tag{2.22}$$

where $d(\mathbf{I}_1) = \mathbf{I}_{1l(\mathbf{I}_1)} - \mathbf{I}_{11} + 1$ and $d(\mathbf{I}_2) = \mathbf{I}_{2l(\mathbf{I}_2)} - \mathbf{I}_{21} + 1$. This way, we penalize both larger trees and child subsequences with gaps. Eq. 2.22 is more general than Eq. 2.21.

[4] To have a similarity score between 0 and 1, we also apply the normalization in the kernel space: $TK_{norm}(T_1, T_2) = \frac{TK(T_1, T_2)}{\sqrt{TK(T_1, T_1) \times TK(T_2, T_2)}}$.

Indeed, if we only consider the contribution of shared subsequences containing all children of nodes, we actually obtain the STK kernel. The computational complexity of PTK is $O(p\rho^2|N_{T_1}| \times |N_{T_2}|)$ [93], where p is the largest subsequent of children that we want to consider and ρ is the maximal out-degree observed in the two trees. However, as shown in [93], the average running time again tends to be linear in the number of nodes for natural language syntactic trees.

2.5 Summary

In this section we gave the minimum required background for the reader to comprehend the material of the subsequent chapters. We started by briefly outlining the general learning setup in supervised learning providing definitions of such concepts as the discriminant function, inference problem, empirical risk minimization, loss functions, etc. Then we provided a brief overview of the main classes of supervised machine learning problems tackled in NLP together with example tasks addressed in this thesis. Finally, we introduced the main machinery behind the kernelized Support Vector Machines that we use in this thesis to approach classification and reranking NLP problems. We first gave a formal definition of SVMs and showed how they can be used together with kernels, thus avoiding the need to explicitly define a non-linear feature map for the input examples. We also included a formal definition of some of the most important structural kernels widely used in NLP and that are of central focus in this thesis.

In the following chapters we build on the structural kernel learning framework proposing a set of techniques to greatly speedup the training process in Chapter 3, and designing novel linguistic tree structures for encoding input texts in Chapters 4, 5, and 6 to benefit from the expressivity of the tree kernel functions.

Chapter 3

Fast Support Vector Machines with Structural Kernels

One of the major drawbacks of learning with kernels is that training typically requires a large number of kernel computations (quadratic in the number of training examples) between the model and training examples. However, when dealing with structural input, in practice, substructures often repeat in the training data which makes it possible to avoid a large number of redundant kernel evaluations.

In this chapter, we propose the use of Directed Acyclic Graphs (DAGs) to compactly represent tree structures in the training algorithm of Support Vector Machines (SVMs). In particular, we use DAGs for each iteration of the cutting plane algorithm (CPA) to encode the model composed by a set of trees. This enables DAG kernels to efficiently evaluate TKs between the current model and a given training tree. Consequently, the amount of total computation is reduced by avoiding redundant evaluations over shared substructures. We provide theory and algorithms to formally characterize the above idea, which we tested on several datasets. The empirical results confirm the benefits of the approach in terms of significant speedups over previous state-of-the-art methods. In addition, we propose an alternative sampling strategy within the CPA to address the class-imbalance problem, which coupled with fast learning methods provides a viable TK learning framework for a large class of real-world applications.

Having faster algorithms to train models with tree kernels still exhibits higher computational complexity than learning approaches using explicit feature vectors and performing optimization in the linear (primal) space. Hence, we derive a novel method that couples our approximate CPA training with DAG model compression and a principled approach to linearize tree kernel models relying on the idea of reverse kernel engineering. The approach gives the advantage of using tree kernels to automatically generate rich structured feature spaces, while remapping the learning problem into the linear space where training and testing is fast. We experimented with training sets up to 4 million examples from

argument boundary detection task for Semantic Role Labeling. The results show: (i) our approach is superior to feature hashing methods on a task where complex semantic features are essential to train an accurate model (ii) we can speed-up the training time of vanilla SVMs with tree kernels from weeks to less than 20 minutes.

3.1 Overview

Despite a great success of kernel methods in many application domains [78, 82, 130] and data-mining [10,30,139,162,173,181,184,191] and NLP in particular [23,34,37,73,79,142], their use has been restricted to relatively small datasets as the training becomes much slower compared to linear models. Indeed, the major drawback of kernel methods, Support Vector Machines (SVMs) in particular, is the necessity to carry out learning in the dual space, where training complexity is typically quadratic in the number of training instances. This is largely attributed to the fact that the model weight vector is represented as a linear combination of training examples (support vectors) that all lie in the implicit feature space spanned by a given kernel function. As the size of the training set increases the number of support vectors in the kernel expansion of the model also tends to grow linearly [154]. Thus, evaluating a dot product between a model and a given example entails a large number of kernel computations over the training examples in the model.

3.1.1 Approximate Cutting Plane Algorithm with model compression

Recently, a number of efficient methods to train SVMs based on the idea of the Cutting Plane Algorithm (CPA) have been proposed [46,69]. The CPA finds the model parameter vector by iteratively constructing cutting plane models that refine the estimation of the empirical risk. The optimal solution is a linear combination of such cutting planes. The linear-time behavior of the CPA again depends on the possibility to compact the model by summing up its constituent feature vectors such that the dot product can be computed efficiently. Unfortunately, again, for the reason briefly outlined above the method scales well only when linear kernels are used. To address slow learning with non-linear kernels, Joachims and Yu [67] propose to extract basis vectors to compactly represent cutting plane models, which speeds up both classification and learning. However, this requires to solve a non-trivial optimization problem, which renders intractable when considering discrete feature spaces generated by structural kernels. Finding a set of basis vectors in such high-dimensional spaces produced by arbitrary kernels, and in particular structural kernels, is an active research area.

Another approach of adapting CPA for non-linear kernels by reducing the number of kernel evaluations is studied in [188], where sampling is used to reduce the number of basis functions in the resulting kernel expansion. In [132], we showed that the same algorithm can be successfully applied to SVM learning with structural kernels on very

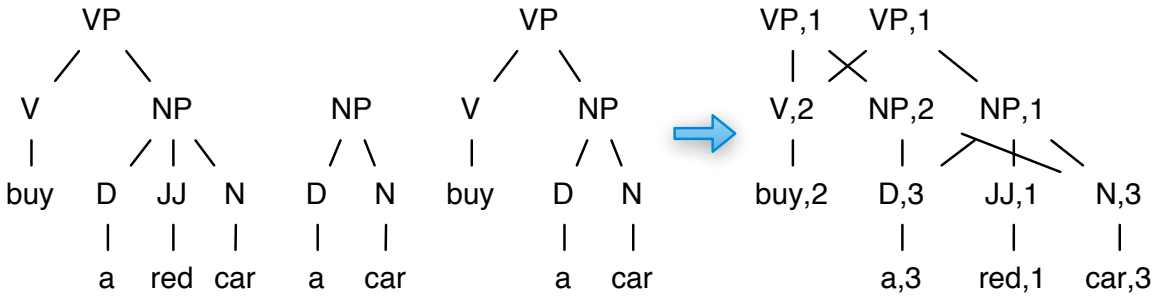


Figure 3.1: Three syntactic trees and the resulting DAG.

large data obtaining speedup factors up to 10 over conventional SVMs. The approach was rather general as we did not make any assumption on the data. In contrast, in [4, 5], we exploited a specific approach based on Directed Acyclic Graphs (DAGs) in the online learning setting to speed up the perceptron algorithm. DAGs were used to compactly represent tree forests given by the support vectors found by the learning algorithm. The approach reduced the number of expensive kernel evaluations, since DAGs provided the means to avoid redundant computations over shared substructures.

To give an intuition of the DAG approach for compact model representation studied in this article, consider an example of the tree-structured data, e.g. syntactic parse trees that are used extensively in Natural Language Processing. Fig. 3.1 shows a model consisting of only three syntactic trees[1] on the left and the resulting DAG on the right. As we can see, the subtree of the noun phrase `[NP [D a][N car]]` is repeated in two trees, thus the frequency of the corresponding node is updated to 2. Also smallest subtrees such as `[D a]` and `[D car]` are shared with a frequency of 3. The two subtrees rooted in `VP` are different and require different roots but they can still share some of their subparts, e.g. `[V buy]`.

The results show that (i) our approach defined in [131] generalizes to most of tree-based kernels; and (ii) the high speedup achieved in [131] was also due to the compactness of the model, which could better fit in the CPU cache, amplifying the benefit of our approach. Nevertheless, the results also demonstrate that there is still a significant speedup for any size of the data at hand. Additionally, when dealing with less sparse data, e.g., with unlexicalized trees, the impact of our approach is further amplified. In particular, on the INEX data, whose trees show much larger repetition of the sub-structures, the DAG methods deliver the speedups of about two orders of magnitude w.r.t. to plain CPA model. This reveals that the potential impact of our approach may be beyond those we have outlined here.

[1] node labels define syntactic categories: NP - noun phrase, D - determiner, V - verb, VP - verb phrase, N - noun, JJ - adjective

3.1.2 Fast linearization of tree kernels

As previously discussed, the scaling behaviour of kernel methods that perform optimization in the dual is in the order of $O(n^2)$. On the other hand, similar to linear-time algorithms with linear kernel, the optimization of kernel machines can also be performed in the primal [28], which, in turn, requires to precompute and keep the entire Gram matrix in the memory. Both methods pose computational problems for applying kernels on datasets with millions of examples. Consequently, several approaches that attempt to trade-off accuracy for speed have been proposed: (i) linearization of the kernel space, i.e., extraction of the most important features in the form of explicit feature vectors, e.g., [34, 79]. (ii) Feature selection in kernel spaces [80], where the implicit computation of feature weights allows one to avoid full computation of the kernel functions; (iii) hybrid approaches mixing kernel computations with on-line linearization of trees [74]; (iv) feature hashing [49, 146], which can highly decrease the size of the linearized space; (v) linearization using reverse-kernel engineering (RKE) approach [113], which exploits SVM models to extract the most important features from the kernel space; and (vi) recent advances in fast SVM learning with structural kernels [131] (SDAG), which, thanks to the approximate cutting plane algorithm and its model compression via directed acyclic graphs, provides significant speedups over conventional SVM training methods. While kernel-based learning have been shown to deliver more accurate models for complex NLP tasks, none of the methods above seem to be optimal when dealing with large-scale datasets without compromising the accuracy of their exact counterparts.

In Section 3.6, we study the latest linearization approaches to large-scale learning with convolution tree kernels (TKs) and derive the following findings: first, it is computationally prohibitive to naïvely enumerate all structural features, which is often mandatory for tackling high-level semantic tasks such as the extraction of predicate argument structures in Semantic Role Labeling (SRL). Indeed, (i) enumerating all possible sub-structures quickly becomes intractable as their number grows exponentially with the size of the input structure; (ii) hashed feature vectors tend to be very dense; and (iii) the feature collisions due to hashing cause the model to underperform w.r.t. structural kernels.

Secondly, we show that reverse-kernel engineering is a principled way to linearize exponentially large tree kernel spaces for tasks where models encoding complex structural features yield better accuracy. However, this approach inherits the computational burden of training an SVM model, which is required by its greedy mining procedure. We attack the major computational bottleneck of this approach by (i) replacing the slow SVM training with our much faster CPA training algorithm with DAG compression, which produces a more compact model in a form of a directed acyclic graph (DAG); and (ii) we define a linearization approach based on fragment extraction directly from the DAG model. This approach achieves the same accuracy as the traditional SVM learning with structural

kernels. We provide an explanation of such surprising result by showing that our SDAG and the SVM models are rather similar, as their most important features have a large overlap. Therefore, we can use fast algorithms to learn a tree kernel model and work in the linear space where learning and testing is fast.

Thirdly, we propose a distributed system to further speed up RKE on the DAG models, which includes: (a) splitting the training set into smaller subsets and (b) using multiple CPUs to learn individual SDAG models. In particular, we show that: (i) the merged feature space can be used to learn the final linear model whose accuracy is just few tens of a point lower than that of the traditional SVMs using tree kernels; and (ii) the entire process takes less than an hour. For example, on the 4 million examples from SRL dataset, the entire linearization process takes less than 20 minutes achieving an F1 of 84.5% vs. 84.8% achieved by SDAG.

Finally, we could carry out large-scale experiments, e.g., using the entire 4 million instances of the SRL dataset (boundary detection from CoNLL 2005 [25]), in a few hours. This does not produce any loss in accuracy w.r.t. the traditional SVMs.

3.1.3 Our contributions

The contributions of this chapter consider our two learning approaches:

1. Approximate CPA with model compression using DAGs:

- We model DAGs to encode the cutting plane models computed at each iteration of the CPA algorithm [131]. We present two different algorithms, which, by compressing the trees in the CPA model, deliver impressive speedups for both training and testing.
- We extend our approach to any tree kernel satisfying some properties, e.g., our approach can be applied to a more general tree kernel, namely, the Partial Tree Kernel (PTK) [93]. PTK not only enables the use of dependency syntactic trees and other different syntactic paradigms, but also allows for applying our fast approach to many other application domains, e.g., it can be applied to XML trees or any other tree-structured data. In contrast to Syntactic Tree Kernel (STK) used in the previous study, PTK takes on a more fine-grained approach by matching any subsequence of children nodes of a given node. This necessitates modifications in the organization of the DAG structure and also yields different levels of compression compared to the STK. We extensively study the effects of using PTK when compacting tree forests into DAGs.
- We investigate the speedup decrease observed when the training size increases by defining a new efficiency measure based on atomic kernel operations that in our case is the Δ function (i.e., the evaluation of the number of shared substructures rooted at two given nodes). This allows us to exactly verify the speedup independently of the hardware used for running the experiments.

- We demonstrate a simple way to parallelize our approach and a method for handling class-imbalanced datasets in the CPA algorithm, which has been previously missing. Plain CPA model previously studied in [132] simply learns a model that minimizes the error rate, which in the case of imbalanced datasets tends to produce models with low Recall. While, CPA can be used to train models that optimize various performance measures, e.g. F_1 score [66], this, however, entails the use of non-decomposable loss-functions, which, in turn, requires to compute the inner product over the entire training set when constructing cutting plane models at each iteration of the CPA. Hence, it prevents the use of sampling to speed up the learning with non-linear kernels. Conversely, we show that a simple cost-proportionate sampling technique is an elegant solution to extend the CPA to handle class-imbalanced data. We demonstrate that using an alternative sampling strategy within the CPA to build cutting planes at each iteration, indeed, provides an efficient way to tune up Precision and Recall of the obtained classifier. We also show that the original convergence bounds still apply to the modified algorithm.
- We carry out an extensive evaluation of our approaches on five datasets: (a) a large dataset of Semantic Role Labeling (SRL) that contains a collection of parse trees expressing predicate argument relationships; (b) a new dataset derived from the previous one by removing lexical information, i.e. words, (unlexicalized trees); (c) question classification dataset, i.e., a taxonomy of the question types used in question answering systems; (d) question and answer pairs from Yahoo! answers; and (e) a new dataset from INEX [165] 2005 competition that contains a collection of XML trees. We evaluated the speedup in terms of the training time and the number of Δ-iterations for both STK and the newly proposed PTK on DAGs.

2. Fast linearization of DAG models:

- We show that reverse-kernel engineering is a principled way to linearize exponentially large tree kernel spaces (compared to feature hashing approaches) for tasks where models encoding complex structural features yield better accuracy.
- We extend the RKE approach of [113] by addressing its major computational bottlenecks and replacing the slow SVM training with our much faster CPA algorithm with DAG compression, which produces a more compact model in a form of a directed acyclic graph (DAG).
- We define a linearization approach based on fragment extraction directly from the DAG model.
- We propose a distributed system to further speed up RKE of DAG models by splitting the training set into smaller subsets and using multiple CPUs to learn individual tree kernel models.

In the remainder of this chapter section 3.2 provides the reader with the required background on the CPA algorithm. Section 3.3 illustrates how the idea of compacting trees into a DAG can be used to speed up the CPA algorithm. Section 3.4 demonstrates our approach to construct DAGs from a tree forest, the computation of the DAG tree kernel and the parallelization of the resulting CPA algorithm. Section 3.5 illustrates our methods for dealing with class imbalance. Section 3.6 defines our procedure to linearize the learning with tree kernels. Section 3.7 reports on our empirical evaluations, while Section 3.8 reports on the related work.

3.2 Preliminaries: Cutting Plane Algorithm with Sampling

In this section, we start off from the problem formulation for binary SVMs (provided in Sec. 2.3 and present a re-elaborated version of the cutting plane method (originally proposed in the context of structural SVMs) for binary classification. Having considered the case for linear SVMs, we point out the main source of inefficiency for the case when non-linear kernels are used. Next, we present the idea of using sampling [188] to approximate cutting planes computed at each iteration of the CPA, which is shown to alleviate high training costs for SVMs with non-linear kernels.

3.2.1 Cutting-plane algorithm (primal)

First, we consider an equivalent formulation of the SVM training problem defined by Eq. 2.9, known as a 1-slack reformulation [69], to derive a more efficient version of the CPA for binary classification:

$$\begin{aligned} \underset{\mathbf{w},\xi\geq 0}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{subject to} \quad & \frac{1}{n}\sum_{i=1}^{n} c_i y_i \mathbf{w}\cdot\mathbf{x}_i \geq \frac{1}{n}\sum_{i=1}^{n} c_i - \xi,\ \forall \mathbf{c}\in\{0,1\}^n \end{aligned} \tag{3.1}$$

where a binary vector $\mathbf{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$ is an index into the training set and selects which training examples form a given constraint. Hence, each of such constraints is composed by a linear combination of the constraints of the form: $y_i(\mathbf{w}\cdot\mathbf{x}_i) \geq 1 - \xi_i$. The key idea behind this equivalent reformulation is to represent the empirical risk by only a single slack variable ξ shared across all the constraints. Even though the number of slack variables is reduced to only a single ξ, the number of constraints is 2^n (as defined by all possible values of $\mathbf{c}$). This prevents the application of off-the-shelf optimization methods to directly solve optimization problem in Eq. 3.1 (OP3.1). Nevertheless, it has been shown in [166] that the cutting plane algorithm applied to the OP3.1 uses only a small subset of active constraints that is independent of the size of the training set.

Algorithm 1 Cutting Plane Algorithm (primal)

1: Input: $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, C, ϵ
2: $S \leftarrow \emptyset; t \leftarrow 0$
3: **repeat**
4: $(\mathbf{w}, \xi) \leftarrow$ optimize OP3.1 over the constraints in S
5: **for** $i = 1$ to n **do**
6: $c_i^{(t)} \leftarrow \begin{cases} 1 & \textbf{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 1 \\ 0 & \textbf{otherwise} \end{cases}$
7: $d^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^n c_i^{(t)}$
8: $\mathbf{g}^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^n c_i^{(t)} y_i \mathbf{x}_i$
9: $S \leftarrow S \cup \{(d^{(t)}, \mathbf{g}^{(t)})\}$
10: $t \leftarrow t + 1$
11: **until** no CPMs are violated by more than ϵ
12: **return** $\mathbf{w}, \xi$

To solve OP3.1, an adaptation of the generic cutting plane algorithm [69] for binary classification problem has been shown to yield significant performance gains over conventional classifiers.

The CPA is presented in Alg. 1. It starts with an empty set of constraints S and computes the optimal solution to the OP3.1. Next, the algorithm forms a binary vector $\mathbf{c}$ that is merely an index into the training set and selects which training examples will form the next cutting plane model (CPM) (defined by an offset $d^{(t)} = \frac{1}{n} \sum_{i=1}^n c_i^{(t)}$ and a gradient $\mathbf{g}^{(t)} = \frac{1}{n} \sum_{i=1}^n c_i^{(t)} y_i \mathbf{x}_i$ (lines 5-8)). The cutting plane model encodes a constraint $\mathbf{w} \cdot \mathbf{g}^{(t)} \geq d^{(t)} - \xi$ that is violated the most by the current solution $\mathbf{w}$, which is then included in the set of active constraints S (line 9). This process is repeated until no CPMs are violated by more than ϵ (line 11), which is formalized by the following criteria $\mathbf{w} \cdot \mathbf{g}^{(t)} \geq d^{(t)} - \xi + \epsilon$.

3.2.2 Cutting-plane algorithm (dual)

While SVMs discussed in the previous section seek to build classifiers that are linear functions, one may achieve better accuracy by using the power of kernels to build highly discriminative non-linear decision boundaries. This is achieved by introducing a mapping function $\phi(\cdot)$ that projects the inputs into some high-dimensional feature space. However, the use of kernels requires to solve the OP3.1 in the dual space. Its solution $\mathbf{w}$ lies in the feature space defined by a kernel $K(\mathbf{x}_i, \mathbf{x}_k) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_i)$. Omitting the details, it can be verified (by deriving the dual from OP3.1) that the solutions of the primal and dual problems are connected via:

$$\mathbf{w} = \sum_{j=1}^{t} \alpha_j \mathbf{g}^{(j)}, \tag{3.2}$$

where α_i are dual variables, $\mathbf{g}^{(j)} = \frac{1}{n}\sum_{k=1}^{n} c_k^{(j)} y_k \phi(\mathbf{x}_k)$ denotes the gradient of the cutting plane model added at iteration j and t is the current iteration.

As one can see, with the use of kernels the gradient $\mathbf{g}^{(j)}$ represents a weighted sum of training examples that lie in the feature space spanned by $\phi(\cdot)$. This implies that a dot product between $\mathbf{w}$ and a given example $\mathbf{x}_i$ requires an explicit computation with each of its components encoded by $\mathbf{g}^{(j)}$, i.e., a common trick, to compact $\mathbf{w}$ into a single vector by simply summing up its n feature vectors is no longer possible. This prohibits to exploit linear-time training algorithms of SVMs with linear kernels and represents a major bottleneck of kernelized SVMs. We will address the problem of compact representation of the cutting plane models in Section 3.3.

Computing an inner product between the weight vector $\mathbf{w}$ and an example $\mathbf{x}_i$ involves the sum of kernel evaluations for each example $\mathbf{x}_k$ in the cutting plane model $\mathbf{g}^{(j)}$ over the set S. In particular, using the expansion of $\mathbf{w}$ from (3.2), the inner product required to find the next cutting plane model (steps 5-8 in the Alg. 1), renders as:

$$\mathbf{w} \cdot \phi(\mathbf{x}_i) = \sum_{j=1}^{t} \alpha_j \mathbf{g}^{(j)} \cdot \phi(\mathbf{x}_i) = \sum_{j=1}^{t} \alpha_j \Big(\frac{1}{n} \sum_{k=1}^{n} c_k^{(j)} y_k \Big) K(\mathbf{x}_k, \mathbf{x}_i), \tag{3.3}$$

The analysis of the inner product given by (3.3) reveals that the number of kernel evaluations is $O(tn)$. Indeed, the number of non-zero elements in each $\mathbf{g}^{(j)}$ is proportional to the number of support vectors which grows linearly with the training size n [154]. Performing the kernel evaluations for each cutting plane model $\mathbf{g}^{(j)}$ in the set S, we obtain the complexity of (3.3) is $O(tn)$. Since the inner product (3.3) needs to be computed for each training example (lines 5-6 in Alg. 1) we obtain the total $O(tn^2)$ scaling behavior for each iteration of the Alg. 1.

The obtained quadratic scaling in the number of examples makes cutting plane training for non-linear SVMs prohibitively expensive for even medium-sized datasets and no better than convenvional decomposition methods such as SMO or SVM-light. To address this limitation [188] proposed to construct approximate cuts by sampling r examples from the training set. The idea is to replace the expensive computation of the cutting plane (lines 5-7, Alg. 1) over all training examples n by a sum over a smaller sample r, s.t. the number of examples in $\mathbf{g}^{(j)}$ is reduced from $O(n)$ to $O(r)$. In this case the double sum of kernel evaluations in (3.3) reduces from $\sum_{i,j=1}^{n} K(\mathbf{x}_i, \mathbf{x_j})$ to a more tractable in practice $\sum_{i,j=1}^{r} K(\mathbf{x}_i, \mathbf{x_j})$. This reduces the complexity of each iteration of Alg. 1 from $O(tn^2)$ to $O(tr^2)$.

Using sampling to approximate cutting planes computed at each iteration introduces an additional parameter into the learning algorithm. Nevertheless, it has been shown in [188] that the resulting training and test set errors are stable with respect to changes in the sample size r. Additionally, [132] extensively studied the effects of the sample size

on the obtained runtime speedups within the context of SVMs with structural kernels. It has been shown that selecting smaller sample sizes r (as small as 100 examples) provides significant savings in the runtime while leading to only a small loss in accuracy. While there is no general strategy for selecting the sample size that gives the best choice for every possible scenario, in Section 3.7 we provide the reader with some simple guidelines on the choice of the sampling size that we found to work well in practice.

3.3 Fast CPA for Structural Kernels

In this section we present an approach to significantly speed up the approximate CPA for structural kernels described in Section 3.2. We observe that for convolution structural kernels that are defined in terms of its substructures, the cutting plane model can be compactly represented as a Directed Acyclic Graph (DAG), where each unique substructure is stored only once. This helps to speed up both the training and classification as the repeating kernel evaluations over shared substructures can be avoided. Most interestingly, this approach can be parallelized during training, thus, making structural kernel learning practical on larger datasets.

3.3.1 Compacting cutting plane models using DAGs

In the previous section we have seen that computing a cutting plane model (CPM) at each iteration involves a quadratic number of kernel evaluations. Using sampling to approximate the cutting plane helps to reduce the number of kernel evaluations.

Here, we explore another method to reduce the number of kernel computations when convolution structural kernels are used. Indeed, when applied to structural data such as sequences, trees or graphs, we can take advantage of the fact that many examples share common sub-structures. Hence, we can use a compact representation of a cutting plane model to avoid redundant computations over repeating sub-structures. In particular, when dealing with tree-structured data, a collection of trees can be compactly represented as a DAG [5]. In the following we briefly introduce the idea behind using DAGs to compactly represent a tree forest and then show how it applies to speed up the learning algorithm.

3.3.2 DAG tree kernels

A DAG can efficiently represent a set of trees (a forest F) by including only unique subtrees and accounting for the frequency of the repeated substructures. Given the DAG representation previously presented in Fig. 3.1, we can define tree kernel functions between a DAG and a tree, which compute exactly the same kernel, with a relevant speedup. The support for this algorithm is given by the following:

Theorem 3.3.1 *Let* $\mathbf{D}$ *be a DAG representing a tree forest* F *and* $K_{dag}(\mathbf{D}, T_2) = \sum_{n_1 \in N_{\mathbf{D}}} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta(n_1, n_2)$ *then*

$$\sum_{T1 \in F} TK(T_1, T_2) = K_{dag}(\mathbf{D}, T_2), \tag{3.4}$$

where $f(n_1)$ *is the frequency associated with* n_1 *in* $\mathbf{D}$*, TK is any tree kernel function that can be factorized with Eq. 6.3 and a* $\Delta(n_1, n_2)$ *function, which counts the number of shared subtrees rooted in* n_1 *and* n_2*.*

Proof Let $\mathcal{S}(F)$ be the set of possible *subtrees* (see the definition in Sec. 2.4.2) of F, i.e., the substructures whose leaves coincide with those of the original tree (in general $\mathcal{T} \neq \mathcal{S}$), then

$$\sum_{T_1 \in F} TK(T_1, T_2) = \sum_{T_1 \in F} \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) = \sum_{T_1 \in F} \sum_{\substack{n_1 : t \in \mathcal{S}(T_1) \\ n_1 = r(t)}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2),$$

where $r(t)$ is the root of the subtree t. The last expression is equal to:

$$\sum_{\substack{n_1 : t \in \mathcal{S}(F) \\ n_1 = r(t)}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2)$$

. Let $\mathcal{S}'$ be the unique subtrees of $\mathcal{S}$, we can rewrite the above equation as:

$$\sum_{\substack{n_1 : t \in \mathcal{S}'(F) \\ n_1 = r(t)}} f(n_1) \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) = \sum_{\substack{n_1 : t \in \mathbf{D} \\ n_1 = r(t)}} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta(n_1, n_2) = \sum_{n_1 \in N_{\mathbf{D}}} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta(n_1, n_2) \square$$

Remark It should be noted that no assumption is made on $\Delta(n_1, n_2)$, thus our approach is valid for a vast set of tree kernels defined by a specific form of its Δ function. Since STK and PTK are based on Eq. 6.3 and their $\Delta(n_1, n_2)$ function computes the number of substructures rooted in n_1 and n_2 the following holds:

Corollary 3.3.2 *Given a DAG* $\mathbf{D}$ *and a forest* F*, let us define*

1. $STK_{dag}(\mathbf{D}, T_2) = \sum_{n_1 \in N_{\mathbf{D}}} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta_{STK}(n_1, n_2)$ *and*
2. $PTK_{dag}(\mathbf{D}, T_2) = \sum_{n_1 \in N_{\mathbf{D}}} \sum_{n_2 \in N_{T_2}} f(n_1)\Delta_{PTK}(n_1, n_2)$

then such DAG kernels exactly compute $\sum_{T1 \in F} STK(T_1, T_2)$ *and* $\sum_{T1 \in F} PTK(T_1, T_2)$*, respectively, where* Δ_{STK} *is Eq. 2.21 and* Δ_{PTK} *is Eq. 2.22.*

Remark It is easy to prove that convolution kernels [57] can be factorized with Eq. 6.3 and a generic $\Delta(n_1, n_2)$. Therefore, our approach at least applies to such large class of kernels.

3.3.3 Fast Computation of the CPM on Structural Data

Having introduced the DAG tree kernel, we redefine the most computationally expensive part of the CPA, i.e. the inner product in Eq. 3.3 required to compute the CPM by compacting $\mathbf{g}^{(j)}$ into a single DAG model $\mathbf{D}^{(j)}$:

$$\mathbf{w} \cdot \phi(\mathbf{x}_i) = \sum_{j=1}^{t} \alpha_j \Big(\frac{1}{n} \sum_{k=1}^{n} c_k^{(j)} y_k \Big) K(\mathbf{x}_k, \mathbf{x}_i) = \frac{1}{n} \sum_{j=1}^{t} \alpha_j K_{dag}(\mathbf{D}^{(j)}, \mathbf{x}_i) \tag{3.5}$$

Unlike Eq. 3.3, where each cutting plane $\mathbf{g}^{(j)}$ is an arithmetic sum of training examples, here we take advantage of the fact that a collection of trees can be efficiently put into an equivalent DAG $\mathbf{D}^{(j)}$. As shown in Th. 3.3.1 computing a kernel $K_{dag}(\cdot, \cdot)$ between an example and a DAG that represents a collection of trees yields an exact kernel value. The benefit of such representation comes from the efficiency gains obtained by speeding up kernel evaluations over the sum of examples compacted into a single DAG.

Alternatively, to benefit even more from the compact representation offered by DAGs, we can put all the cutting planes from the active set S into a single DAG model $\widehat{\mathbf{D}}$, such that the inner product in Eq. 3.5 is reduced to a single kernel evaluation:

$$\mathbf{w} \cdot \phi(\mathbf{x}_i) = \frac{1}{n} \sum_{j=1}^{t} \alpha_j K_{dag}(\mathbf{D}^{(j)}, \mathbf{x}_i) = \frac{1}{n} K_{dag}(\widehat{\mathbf{D}}^{(t)}, \mathbf{x}_i), \tag{3.6}$$

where $\widehat{\mathbf{D}}^{(t)}$ at iteration t is built by compacting all $\mathbf{D}^{(j)}$ together with their corresponding dual variable α_j. This ensures that a single K_{dag} evaluation over the full DAG model makes Eq. 3.6 equivalent to computing a weighted sum of K_{dag} using individual $\mathbf{D}^{(j)}$ in Eq. 3.5. Henceforth, we call the approach to compute the inner-product in Eq. 3.3 using more efficient expressions in Eq. 3.5 and Eq. 3.6 as SDAG and SDAG+ respectively. A more detailed explanation of the DAG structure implementation along with the pseudocode of algorithms for building a DAG from a set of trees, compacting a set of DAGs into a single DAG and computing K_{dag} are given in the forthcoming section.

Now we are ready to present the new cutting plane algorithm (Alg. 2) adapted for the use of tree kernels with DAGs. Different from Alg. 1 here we use a set of r examples uniformly sampled from the original training set to approximate the CPM computed at each iteration of the CPA, s.t. computing the inner product over all n examples in the training set is reduced to a much smaller sample r. This also reduces the size of the resulting model weight vector, since each CPM includes maximum only r training points. To compute the CPM we can use either SDAG or SDAG+ approach to form a binary vector $\mathbf{c}$, which then defines the training examples that are further inserted into the DAG to represent the CPM at iteration t.

As one can see, while using SDAG+ approach provides better compression, since all

Algorithm 2 Cutting Plane Algorithm (dual) using DAGs

1: Input: $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, r, C, ϵ,
2: $S \leftarrow \emptyset$; $\mathbf{D} \leftarrow \emptyset$; $t \leftarrow 0$;
3: **repeat**
4: **Update the Gram matrix G with a new CPM**
5: $\alpha \leftarrow$ optimize Wolfe dual of OP3.1
6: $I \leftarrow$ index set of r examples uniformly sampled from the training set X
7: $\mathbf{c}^{(t)} \leftarrow$ find CPM using SDAG (Alg. 3) or SDAG+ (Alg. 4)
8: $\mathbf{D}^{(t)} \leftarrow \text{buildDAG}(\mathbf{x}_{c^{(t)}}, \mathbf{y}_{c^{(t)}})$
9: $d^{(t)} \leftarrow \frac{1}{r}\sum_{i=1}^{r} c_i^{(t)}$
10: $S \leftarrow S \cup \{(d^{(t)}, \mathbf{D}^{(t)})\}$
11: $t \leftarrow t + 1$
12: **until** no CPMs are violated by more than ϵ
13: **return** $\mathbf{w}, \xi$

Algorithm 3 Find CPM with SDAG

Input: $\mathbf{x}, S, \alpha, I$
for each $i \in I$ **do**

$$c_i^{(t)} \leftarrow \begin{cases} 1 & \textbf{if } y_i/r \sum_{j=1}^{t} \alpha_j K_{dag}(\mathbf{D}^{(j)}, \mathbf{x}_i) \leq 1 \\ 0 & \textbf{otherwise} \end{cases}$$

return $\mathbf{c}^{(t)}$

CPM models $\mathbf{D}^{(j)}$ are compacted into a single $\widehat{\mathbf{D}}^{(t)}$, it, however, needs to be re-built at each iteration to accommodate an update in vector α after re-solving the dual of OP3.1. Nevertheless, the time to construct $\widehat{\mathbf{D}}$ is linear in the number of nodes in the model and imposes negligible computational overhead in practice. Another computational drawback of using full DAG model compared to the set of $\mathbf{D}^{(j)}$ is that in the former case we need to compute the update of the Gram matrix column (line 4 in Alg.2) $G_{it} = \mathbf{D}^{(i)} \cdot \mathbf{D}^{(t)}$ for $1 \leq i \leq t$, while in the latter case it is obtained automatically from computing Eq. 3.5.

Even though the worst-case complexity of computing CPMs at each iteration using both variants of DAGs is still $O(r^2)$, in practice we can observe much better scaling behavior, since in real datasets examples tend to share many common substructures. As verified by the extensive experiments in section 3.7, this greatly speeds up both training and classification by avoiding redundant kernel computations. Finally, it is important to note that the obtained Alg. 2 preserves all theoretical benefits of the approximate CPA with sampling, since the kernel computations remain the same, while in practice greatly reducing the number of expensive kernel evaluations to compute the CPM.

3.4 Implementation of the DAG Kernel

The implementation of the DAG kernel requires two different algorithms for: (i) efficiently inserting trees into a DAG and (ii) computing a kernel between a DAG and a given tree. Additionally, the DAG kernel computation can be parallelized.

Algorithm 4 Find CPM with SDAG+

Input: $\mathbf{x}, S, \alpha, I$
$\widehat{dag} \leftarrow \text{compactDAG}(S, \alpha)$
for each $i \in I$ **do**
$c_i^{(t)} \leftarrow \begin{cases} 1 & \textbf{if } y_i / rK_{dag}(\widehat{dag}, \mathbf{x}_i) \leq 1 \\ 0 & \textbf{otherwise} \end{cases}$
return $\mathbf{c}^{(t)}$

3.4.1 DAG construction

There are various methods to efficiently build a DAG corresponding to a collection of trees, F, see for example [5]. Given a tree, $T \in F$, we need to insert its nodes in the DAG, where the uniqueness of each node is defined by its corresponding subtree. The latter criteria is crucial, since the DAG representation has to conform to the recursive nature of the tree kernel functions computed over the DAG, i.e. two nodes which corresponding subtrees differ in only one element will have to be inserted separately in the DAG. For this purpose, to insert tree nodes $n \in T$, we need to be able to efficiently check if the subtree rooted at n is already in the DAG. The key of the node n can simply by a serialized string representation of the subtree rooted at this node. In case the node is already present in the DAG we only need to update its associated weight.

The node weights are defined as $y_j/\sqrt{|T|}$ or $y_j \cdot \alpha_j/\sqrt{|T|}$ for SDAG and SDAG+ respectively[2], such that the DAG kernel produces an equivalent tree kernel value when computing the inner product in Eq. 3.5 or Eq. 3.6. Hence, each element in the DAG is a pair of two items: a node (we only need to keep a pointer) and its weight. This is enough to obtain a compact model representation, where repeating tree sub-structures are accounted by their corresponding weights and can be uniquely stored in the DAG. This construction ensures that one obtains the same TK values.

As we have seen from section 3.3.2 a DAG tree kernel is defined as the sum of Δ-function evaluations over the node pairs sharing some common property, i.e. production rules (STK) or node labels (PTK) are the same. For this purpose, it is convenient to maintain a second associative array indexed by either the production rule (STK) or node labels (PTK) in the DAG structure, such that for a given node $n \in T$ we can retrieve a list of all matching nodes N in the DAG in constant time.

Hence, we design the DAG data structure D to contain two associative arrays: *nodes* and *productions*. The former is used to perform constant time membership checks when inserting a new node, such that two nodes with identical subtrees are stored only once. The latter allows for the constant time retrieval of a list of node pairs in a DAG matching the production rule (node label) of a given node n, which is used for computing a DAG tree kernel. In this way we can efficiently insert trees into a DAG and enumerate all

[2]here, we consider the normalized version of the tree kernel, i.e. $K(T1,T2) = K(T1,T2)/(\sqrt{K(T1,T1)} * \sqrt{(K(T1,T1))}$, hence, we need to introduce the tree norm $|T|$ into the node weight.

the candidate substructures sharing the same production (STK) or node label (PTK) to compute the tree kernel between a DAG and a given tree. The aforementioned implementation ideas are formalized in Algorithms 5 and 8 which provide the pseudocode for inserting trees into a DAG.

More specifically, to insert a tree T into a DAG D we proceed as follows: we first define the weight of the nodes to be inserted into a DAG as the tree label y_i divided by the tree norm $|T|$, such that we effectively compute the inner product as defined in Eq. 3.5. Computing the tree norm for each tree in the input data is linear in the tree size (number of nodes) and is done at pre-processing time. Next, for each node in the input tree, we compute the nodes id[3] and use it to check if the node is already present in the DAG. If it is present, we simply update its weight, otherwise we insert the node together with its weight into both associative arrays: *nodes* and *productions*. For the latter, we use nodes production (label) to retrieve a list of matching nodes and append the new element. To give a concrete example of tree insertion, we refer back to the constructed DAG in Fig. 3.1 (right) that compactly represents a collection of 3 trees (left). Suppose, we would like to insert another tree T': `[NP [D a] [N car]]` into our DAG D. T' consisting of 3 non-terminal and 2 leaf nodes with labels {`NP, D, N`} and {`a, car`} respectively. Since the node weights and node ids for each node $n \in T'$ have been already precomputed at the pre-processing time, we simply iterate over nodes checking for their membership in D. For example, to insert the root node `NP` we query the array *nodes* in DAG data structure using its node id as a key. Node id is simply a fingerprint (unique hash value) of the string `(NP(D(a))(N(car)))`, i.e. a serialization of the full subtree rooted at this node. Since, such node already exists in D, we simply update its associated node weight in D.

Finally, Alg. 8 shows how a set of DAGs $\mathbf{D}^{(j)}$ are compacted into a single $\widehat{D}$, which allows for a reduction of an inner product in Eq. 3.5 to a single kernel evaluation as shown in Eq. 3.6. Note that, to account for individual α_j of each $\mathbf{D}^{(j)}$ when inserting its node into a $\widehat{\mathbf{D}}$, we simply make it as an additional factor of the node weight, i.e. $weight \cdot \alpha_j$ (line 7).

3.4.2 DAG kernel computation

Having discussed the particular implementation of the DAG data structure that allows for efficient DAG construction from a collection of trees, we now consider how one can compute a tree kernel over a given tree and the constructed DAG D. The associative array *productions* allows us to efficiently compute TKs by retrieving (in constant time) a list of nodes in the DAG matching a given node $n \in T$ to evaluate Δ for each pair of nodes. In particular, computing a tree kernel between a DAG and a tree (see Alg. 11) simply requires (i) looping over nodes in a tree (line 3), (ii) retrieving a list of nodes in

[3]Computing node ids requires to serialize subtrees rooted at each node which is linear in the tree size and is also performed at the preprocessing stage.

Algorithm 5 Build a DAG from a collection of trees

```
1: Input: sample of I, dag D
2: for i ∈ I do
3:    weight ← y_i / sqrt(|T_i|)
4:    for each node ∈ T_i do
5:       addNode(D, weight, node)
```

Algorithm 6 Insert a tree node into a DAG

```
1: procedure addNode(D, weight, node)
2: key ← id(node)
3: K ← all keys in D.nodes
4: if key ∈ K then
5:    updateWeight(D.nodes[key], weight)
6: else
7:    newDagElement ← construct a new pair (weight, node)
8:    D.nodes[key] ← newDagElement
9:    dagElementsList ← D.productions[node.production]
10:   append(dagElementsList, newDagElement)
```

the DAG matching node n (line 7) and (iii) summing up the product between the weight of the considered node in the DAG and the value returned by a call to a Δ function (line 10). Note that we return the final value of the *sum* divided by a tree norm $|T|$, s.t. evaluating K_{dag} over a DAG and a set of trees yields a normalized TK value. This procedure is similar for both STK and PTK kernels with the difference that for PTK we form the matching node pairs using node labels instead of production rules.

3.4.3 Parallelization

The modular nature of the CPA suggests easy parallelization. In fact, in our experiments, we observed that at each iteration 95% of the total learning time is spent on computing the CPM (steps 3-9, Alg. 2). This involves computing Eq. 3.5 over the set of individual DAGs or Eq. 3.6 using full DAG model for the sample of r training examples. Using p processors the complexity of this pre-dominant part can be brought down from $O(r^2)$ to $O(r^2/p)$.

3.5 Handling Class-Imbalanced data

Having considered a set of techniques to speed up the training of SVMs with tree kernels, we now turn to addressing another important problem of dealing with class-imbalanced data. This problem often arises in situations when we have to deal with datasets where the number of negative examples largely outnumbers the number of positive examples. On such datasets, a typical classifier that is minimizing a mis-classification rate is likely

Algorithm 7 Compact a set of CPMs into a single DAG

Input: set of CPMs S, vector of dual variables α
$\widehat{D} \leftarrow newDag()$
for $j = 1$ to $length(S)$ **do**
  $K \leftarrow$ all keys in $D^{(j)}.nodes$
  for each $key \in K$ **do**
    $weight, node \leftarrow D^{(j)}.nodes[key]$
    addNode$(\widehat{D}, weight \cdot \alpha_j, node)$
return $\widehat{D}$

Algorithm 8 Compute $K_{dag}(\mathbf{D}, \mathbf{T})$

Input: tree T, dag
$sum \leftarrow 0$
for each $node \in T$ **do**
  $key \leftarrow$ production rule of $node$
  $P \leftarrow$ all production rules in $D.productions$
  if $key \in P$ **then**
    $dagElementsList \leftarrow D.productions[key]$
    for each $dagElement \in dagElementsList$ **do**
      $weight, dagNode \leftarrow dagElement$
      $sum \leftarrow sum + weight \cdot \Delta(node, dagNode)$
return $sum/\sqrt{|T|}$

to learn a model that will tend to label all examples as negative. Hence, it will will do poorly in terms of Precision and Recall.

Thus, in this section, we extend the theory of the cutting-plane algorithm to tackle class-imbalance problem. Our approach is based on an alternative sampling strategy, e.g. cost-proportionate sampling, that is effective for tuning up Precision and Recall on class-imbalanced data. Typically, cost-proportionate sampling is used to alter the distribution of the original training set to make the proportion of positive and negative examples balanced, such that the classifier can be trained on a balanced data. However, the CPA operates differently as it iteratively draws samples to build an approximation of the cutting plane models at each step. Hence, it is important to verify that altering the distribution of the examples in the sample used to build CPM at each step, indeed, allows for an effective way to tune up Precision and Recall of the final classifier. We also demonstrate that the same convergence bounds hold when cost-proportionate sampling is applied within the CPA.

3.5.1 Cost-proportionate sampling

Conventional SVM problem formulation allows for natural incorporation of example dependent importance weights into the optimization problem. We can modify the objective

function to include example dependent cost factors:

$$\begin{aligned} \underset{w,\xi_i \geq 0}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i}^{n} z_i \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w} \cdot \mathbf{x_i}) \geq 1 - \xi_i,\ 1 \leq i \leq n \end{aligned} \tag{3.7}$$

where z_i is the importance weight of example i and $\frac{1}{n}\sum_i^n z_i\xi_i$ serves as an upper bound on the total cost-sensitive empirical risk. This problem formulation where there is an individual slack variable ξ_i for each example is typically referred to as "n-slack" formulation.

In the dual space, the example-dependent costs captured by cost factors z_i translate into the box constraints imposed on each dual variables: $0 \leq \alpha_i \leq z_i C,\ 1 \leq i \leq n$ such that the $z_i C$ sets an upper bound on the values of α_i. This feature to integrate importance weights z_i in the problem formulation is implemented in SVM-light software.

This natural modification of the quadratic problem, is, however, difficult to incorporate in the case of 1-slack formulation (OP3.1). Indeed, in the case of 1-slack formulation we have a single slack variable ξ that is shared among all the constraints. More importantly, moving to the dual space, the box constraints $0 \leq \alpha_i \leq C$ are no longer for each individual dual variable but for a sum: $\sum_i \alpha_i$. This makes the 1-slack problem formulation difficult to incorporate importance weights directly. Nevertheless, the idea of approximating the cutting plane model at each iteration via sampling suggests a straightforward solution.

Indeed, we can extend the original CPA to the case of cost-sensitive classification. A straight-forward way to do this is instead of using uniform sampling to build an approximation to the CPM at each iteration in Alg. 2), we can draw examples according to their importance weights using the cost-proportionate rejection sampling technique (Alg. 9).

Algorithm 9 Cost-proportionate rejection sampling

1: Pick example $(\mathbf{x_i}, y_i, z_i)$ at random
2: Flip a coin with bias z_i/Z
3: **if** *heads* **then**
4: keep the example
5: **else**
6: discard it

Here z_i is the importance weight of the i-th example and Z is an upper bound on any importance value in the dataset. This process is repeated until we sample the required number of examples r. This modification enables the control over the proportion of examples from different classes that will form a sample used to compute the CPM.

Unlike the conventional approaches for addressing the class-imbalance problem, that either under-sample the majority class or over-sample the minority class from the training data, the rejection sampling coupled with CPA does not completely discard examples from the training set. At each iteration it forms a sample according to the pre-assigned

importance weights for each example, such that examples from both the majority and minority classes enter the sample in the desired proportion. This process is repeated until the algorithm converges. Thus, the learner has the chance to incorporate relevant information present in the data over a number of iterations before it converges. This way, the method preserves the global view on the dataset and no relevant information is lost during the iterative optimization process unlike in the "one-shot" sampling methods.

Another benefit of this approach is that by increasing the importance weight of the minority class, we give its examples more chance to end up in CPM and hence, become support vectors. This way the imbalanced support-vector ratio is automatically tuned to include more examples from the minority class, which gives more control over the class-imbalance problem. Proving this property could be an interesting theoretical result.

3.5.2 Theoretical Analysis of the Algorithm

Cost proportionate rejection sampling allows for natural extension of the binary classification to importance weighted binary classification. It achieves this task by re-weighting the original distribution of examples D according to the importance weights of examples such that the training is effectively carried out under the new distribution $\hat{D}$.

In [190] it is shown that by transforming the original distribution D to a training set under $\hat{D}$, one can effectively train a **cost-insensitive** classifier on a dataset $\hat{D}$ such that it will minimize the expected risk under the original distribution D.

Theorem 3.5.1 (Translation Theorem; [190]) *Learning a classifier h to minimize the expected cost-sensitive risk under the original distribution D is equivalent to learning a decision function to minimize the expected **cost-insensitive** risk under the distribution* $\hat{D}(x, y, z) \equiv \frac{z}{E_{(x,y,z)\sim D}[z]} D(x, y, z)$.

The proof is a straight-forward application of the definitions and simply follows by establishing an equivalence relationship between the expected cost-sensitive risk $E_{(x,y,z)\sim D}[z\Delta(y, h(x))]$ under the original distribution D and the expected cost-insensitive risk $E_{(x,y,z)\sim \hat{D}}[\Delta(y, h(x))]$ under the transformed distribution $\hat{D}$. The theorem produces an important implication that by transforming the original distribution D to $\hat{D}$ according to example-dependent importance weights, a classifier for the cost-sensitive problem over D can be obtained with a cost-insensitive learning algorithm over D. We can use this finding to show that the convergence proof for the original CPA with uniform sampling naturally applies to the proposed version of the algorithm that uses cost-proportionate rejection sampling:

Theorem 3.5.2 (Convergence) *Assume $R = max_{1\leq i\leq n}\|\phi(\mathbf{x}_i)\|$, i.e. R is an upper bound on the norm of any $\phi(\mathbf{x}_i)$, and $\Delta = max_{1\leq i\leq n}\| \Delta(y, y_i)\|$, the number of steps required by Alg. 2 using the sampling strategy of Alg. 9 is upper bounded by $8C\Delta R^2/\epsilon^2$.*

Proof. We first note that the cost-proportionate rejection sampling (Alg. 9), used to build the approximate cutting plane model, at each step re-weights the original distribution D according to the importance weights of the examples. This means that we are effectively training a cost insensitive classifier that draws examples to build the cutting plane model from the transformed distribution $\hat{D}$. By invoking the Translation Theorem (3.5.1), we establish that, to obtain a cost-sensitive classifier that minimizes the expected risk under the original distribution D, it is sufficient to learn a cost-insensitive classifier under the transformed distribution $\hat{D}$. The CPA that draws examples from D using rejection sampling is equivalent to the original CPA applying uniform sampling to the transformed distribution $\hat{D}$. Thus, we can reutilize the proof in [188] of the convergence bounds for the original CPA with uniform sampling over $\hat{D}$. This states that CPA with uniform sampling terminates after at most $8C\Delta R^2/\epsilon^2$ iterations. By applying such bound, we have proved the thesis of the theorem.

Remarks. The main idea to obtain convergence bounds in [188] is to set an upper bound on the value of the dual objective and if there exists a lower bound on the minimal improvement of the dual objective at each iteration, then the algorithm will terminate in a finite number of steps.

Indeed, using the relationship between primal and dual problems, we have that a feasible solution of the primal OP1, such as, for example: $\mathbf{w} = \mathbf{0}$, $\xi = \Delta$, forms an upper bound $C\Delta$ on the dual objective of 3.1. Next, in [166] it is shown that the inclusion of ϵ-violated constraint at each iteration improves the dual objective by at least $\epsilon/8R^2$. Since the dual objective is upper bounded by $C\Delta$, the algorithm terminates after at most $8C\Delta R^2/\epsilon^2$ iterations.

The derivation of the bound on the minimal improvement of the dual objective obtained at each step only depends on the values of ϵ and R and does not rely on the assumption about distribution of the examples. Also note that each cutting plane model built via rejection sampling is a valid constraint for the OP1.

3.6 Linearization of DAG models

In this section we first briefly describe a feature hashing approach that could be applied to learning a TK model by explicitly enumerating features implicitly generated by tree kernel functions. Next we propose our linearization approach based on the idea of Reverse Kernel Engineering and define the feature mining process directly on the DAG model.

3.6.1 Feature enumeration aka feature hashing

Recently, much attention has been drawn to feature hashing approaches, e.g. [49, 146], that aim at transforming high dimensional kernel spaces into the linear space, where fast learning methods can be applied. The approach forces multiple features to collide, thus

achieving drastic reduction in the effective dimension of the feature space. Still, such approaches may introduce several problems: (i) enumerating a huge number (exponential in the input size) of substructures encoded by the structural kernels may become intractable; and (ii) the noise introduced by feature collisions[4] may have a negative effect on problems where few complex structural features are essential for the learning system.

3.6.2 Reverse Kernel Engineering

A more principled feature extraction algorithm for Tree Kernel (TK) spaces has been proposed in [113]. Its greedy mining algorithm relies on the model learned by an SVM to extract the most relevant features (tree fragments).

Different from the naïve feature enumeration method, the support vectors of the model along with their weights are used to guide the greedy mining to extract the most relevant tree fragments. In particular, the greedy mining approach extracts the fragments f_j from support vectors (trees) t_i based on the relevance score defined as follows:

$$\sum_{i=1}^{n} \alpha_i y_i t_{i,j} \lambda^{l(f_j)} / ||t_i|| \tag{3.8}$$

where α_i are the Lagrange multipliers of the learned model, λ - kernel decay factor, $l(f_j)$ - depth of the fragment f_j, y_i - example label and $t_{i,j}$ is the frequency of the fragment j in the t_i.

While originally proposed to extract features from SVs of the SVM model, our Alg. 10 presents a simplified version of the greedy mining algorithm in [113] that works directly on a DAG model. Each node in the DAG is associated with a weight defined as follows: $\nu_i = \alpha_i y_i f_i / ||t_i||$. This definition of the node weights is convenient to simplify the mining procedure, since all the components to compute the relevance weights from Eq. 3.8 are already provided by the DAG. We iterate over all nodes in the DAG calling the function $frag(n)$, which generates the smallest fragment rooted at node n. These base fragments are further expanded by calling the function $expand(f)$ which enlarges the current fragment by adding direct children to some of its nodes. This process is repeated until the index is unchanged and the top K features are returned.

The information about the most relevant fragments stored in the index is then used to linearize both training and test datasets. Each tree in the input data can the be explicitly represented by a feature vector, where each attribute corresponds to one of the tree fragments as generated by TK.

[4] This poses little to no penalty for some tasks where less complex features suffice to build accurate models.

Algorithm 10 mineDAG(**dag**, K)

for each $\langle \nu, t \rangle \in$ **dag do**
  for each $n \in \mathcal{N}_t$ **do**
    $f \leftarrow frag(n); w = \lambda * \nu$
    $updateIndex(\langle f, w \rangle)$
while $Changed(Index)$ **do**
  for each $\langle f, w \rangle \in Index$ **do**
    for each $\chi \in expand(f)$ **do**
      $updateIndex(\langle \chi, \lambda * w \rangle)$
  $\mathcal{F}_K \leftarrow top(K)$
return $\mathcal{F}_K$

3.7 Experiments

In our experiments we pursue the following goals:

- study the effects of compacting the cutting plane model by using DAGs on both training and classification runtime
- demonstrate the speedup factors one can obtain after straight-forward parallelization offered by the CPA
- demonstrate the ability of the cost-proportionate sampling scheme to tune up Precision and Recall
- evaluate the speedup when applying our linearization process to the DAG models

3.7.1 Experimental setup

We integrated CPA with uniform sampling as described in [188] within the framework of SVM-Light-TK [65,94] to enable the use of structural kernels, e.g. we used STK and PTK (see Sec. 2.4.2). PTK has been indicated as the most accurate in similar tasks, e.g. [94], while PTK is a more general yet much more computationally demanding. To measure the classification performance, we use Precision, Recall and F_1-score, i.e. a harmonic mean between Precision and Recall.

For the DAG implementation, we employ highly efficient Judy arrays[5]. For brevity, we refer to the CPA with uniform sampling as uSVM; uSVM where each cutting plane $\vec{g}^{(j)}$ is compacted into a $\vec{D}^{(j)}$ as SDAG; uSVM with a single DAG that fits all active constraints in the set S as SDAG+; uSVM with cost-proportionate sampling as uSVM+j (Alg. 9), and SVM-light-TK as SVM. The margin trade off parameter is fixed at 1.0.

We ran all the experiments on machines equipped with Intel® Xeon® 2.33GHz CPUs carrying 6Gb of RAM under Linux. Parallel implementation relies on the OpenMP library.

[5] http://judy.sourceforge.net

3.7.2 Data

Semantic Role Labeling. To evaluate the efficiency of the compact model representation offered by SDAG and SDAG+ algorithms with respect to uSVM, we use Semantic Role Labeling (SRL) benchmark. The dataset consists of the Penn Treebank texts [85], PropBank annotation [106] and Charniak parse trees [29] as provided by the CoNLL 2005 shared task on Semantic Role Labeling [25]. The goal is to recognize semantic roles of the target verbs in a given sentence.

SRL is a complex tasks where the state-of-the-art systems achieve F_1 at about 80%, which indicates the importance of extracting the best features. A common approach to tackle SRL problem involves two steps: (i) detection of the verb arguments and (ii) classification of identified arguments into their respective semantic categories and final annotation of the original parse tree. In our experiments, we focus on the first task of argument identification (i.e. the exact sequence of words spanning an argument). This corresponds to the classification of parse tree nodes into correct or not correct boundaries. For this purpose, we train a binary *Boundary Classifier* using the AST subtree defined in [97], i.e. the minimal subtree, extracted from the sentence parse tree, including the predicate and the target argument nodes. To evaluate the learned models we report the F_1[6] on two sections: 23 and 24, that contain 230k and 150k examples respectively. SRL dataset has already been used to extensively test uSVM for structural kernels and we follow the same setting as described in [132] unless mentioned otherwise.

Unlexicalized trees. We also conduct an important study on the sparsity effect inherent to the syntactic parse trees. While the trees from SRL dataset have a very small number of unique non-terminal nodes (less than 100), the number of unique subtrees is huge. This is largely attributed to a great variety of the leaf nodes (lexicals) which for syntactic parse trees are simply words. Consequently, many subtrees that have identical structure up to the leaf nodes may differ because only one word is different. Hence, such nodes when inserted into a DAG will be stored separately, which prevents greater levels of compression. In NLP, the number of unique leaf nodes (words) can be hundreds of thousands and more, therefore, to better understand the sparsity effect caused by the leaf nodes, we carry out another set of experiments on unlexicalized trees from SRL. We simply remove the leaf nodes and re-train our models on this modified data. This allows for better assessment of how our approach may perform in other settings where trees are much less diverse and DAGs can yield better compression.

XML tree classification. Additionally, to better understand how the DAG idea translates to other domains different from NLP, we also present the results for XML tree classification from the INEX 2005 challenge [40]. The dataset is formed by XML docu-

[6] the reported scores corresponds to the accuracy of the binary classifier, which is slightly higher than the accuracy of the overall boundary detection due to errors in parsing.

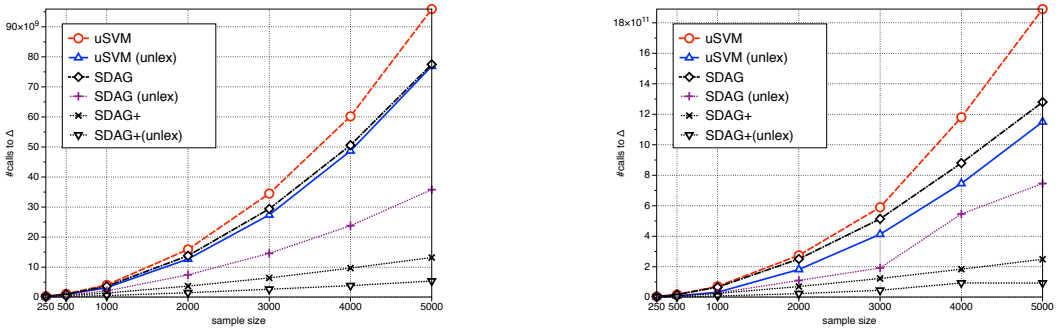


Figure 3.2: Comparison of the number of calls to Δ for uSVM, SDAG, and SDAG+ using STK (left) and PTK (right) kernels on the original SRL and unlexicalized version.

ments describing movies from the IMDB site[7]. The particular steps taken to pre-process the trees are described in [165]. The size of the training and test set is 4820 and 4810. The distinctive property of this data is that the total number of unique node labels (XML tags) is 197. This is much smaller when compared to very sparse syntactic structures in NLP. This fact has a direct effect on the number of common substructures shared among the training examples. As we will see in the next section this plays a significant role for the SDAG and SDAG+ algorithms, which can exploit more compact models.

Question Answering. In the next set of experiments to study the ability of uSVM+j to tune up Precision and Recall we used two different natural language datasets: TREC 10 QA[8] (training: 5,483, test: 500) and Yahoo! Answers (YA)[9](train: up to 300k, test: 10k) to perform two similar tasks of QA classification. The task for the first dataset is to select the most appropriate type of the answer from a set of given possibilities. The goal of the experiments on these relatively small datasets is to demonstrate that rejection sampling is able to effectively handle class imbalance similar to SVM. For Yahoo! Answers dataset the classification task was set up as follows. Given pairs of questions and corresponding answers learn if in a given pair the answer is the "best" answer for a question. The goal of this experiment is to have a large classification task (300k examples in our experiments) to demonstrate that class-imbalance problem can be handled effectively at a scale where SVM becomes too slow.

3.7.3 Comparative speedup analysis of DAG-based models

The goal of this set of experiments is to study computational savings that come from using a compact representation of individual (SDAG) or the full set (SDAG+) of cutting plane models in S. As the baseline for the learning and classification runtime comparison, we use plain uSVM algorithm. Note that the classification accuracy is not of concern here

[7] http://www.imdb.com
[8] http://l2r.cs.uiuc.edu/cogcomp/Data/QA/QC/
[9] retrieved through the Yahoo! Webscope program.

(hence, not reported) since SDAG and SDAG+ produce exact kernel evaluations, thus they train the same model as uSVM.

Learning speedups

To carry out training, we use 100k examples from the second section of the SRL dataset. Figure 3.2 provides the first intuition on the runtime savings provided by SDAG and, especially, SDAG+ that is able to provide the most compact model representation. The graph plots the total number of calls to Δ function made by the learning algorithm during the training phase for uSVM, SDAG and SDAG+ for both the original SRL and SRL with lexicals removed. While the number of kernel evaluations technically remains the same for all the algorithms (using the same tree kernel), it is the number of Δ calls (see Eq. (2.21) and Eq. (2.22)) that greatly differ between uSVM, SDAG and SDAG+. As we can see, SDAG+ provides much better computational savings in terms of the Δ-calls than uSVM and SDAG for both STK and PTK. This benefit becomes especially strong for unlexicalized dataset where DAGs are able to provide even more compact representation.

Tables 3.1 and 3.2 present a more detailed performance comparison of SDAG and SDAG+ with respect to uSVM on 100k subset of SRL and its unlexicalized version where leaf nodes representing words (lexicals) were removed. The bottom part shows the results of training on INEX dataset. We carried out comparative experiments on both STK and PTK kernels and used uSVM outcome as a yardstick. For each algorithm and each kernel, we report two relative metrics: ratios between the number of calls to Δ function as defined in equations (2.21) and (2.22) for STK and PTK correspondingly.

As one can see both SDAG and SDAG+ deliver significant speedups during the learning with respect to uSVMs. The main quantity to observe here is the savings in Δ-computations as they largely define the runtime of the algorithms. SDAG+ performs much better than SDAG being able to provide the most compact representation of tree forests. The results become much stronger when considering unlexicalized SRL and especially INEX datasets where subtrees tend to be less sparse. SDAG+ is a clear winner here for both STK and PTK kernels. Another metric reported in Tables 3.1 and 3.2 is the actual training time. Surprisingly, SDAG+ is able to deliver speedups in training time up to 20 for STK kernel when a large sample size is used. We also observe that training time speedups are much higher than the savings in Δ-computations. This can be explained by the fact that the compact DAG model require less memory and can remain in the CPU cache thus delivering time savings better than expected[10]. Another interesting finding is that as subtrees become less sparse (unlexicalized SRL and INEX) we obtain much better compression. Especially, the results on INEX data where the number of unique node labels (XML tags) is much smaller than for natural language trees, show the true

[10]The exact quantification of the role of the CPU cache along with the design of more efficient algorithms based on it is beyond the scope of this work.

Table 3.1: Evaluation using STK: speedups w.r.t. training time and number of calls to Δ function for SDAG and SDAG+ over uSVM on three datasets: SRL (top), unlexicalized SRL (middle), and INEX 2005 dataset (bottom). The size of both SRL datasets is 100k whereas the size of INEX 2005 is 5k. For uSVM we report the absolute values of Δ and training time (minutes), t, while for SDAG and SDAG+ we illustrate the relative speedups, i.e., the rate between Δ and t and Δ_S and t_S of SDAG or Δ_{S+} and t_{S+} of SDAG+.

SAMPLE	**uSVM**		**SDAG**		**SDAG+**	
	Δ	t	Δ/Δ_S	t/t_S	Δ/Δ_{S+}	t/t_{S+}
SRL						
250	3.3E+08	3	1.1	5.1	1.8	4.0
500	1.1E+09	10	1.1	6.3	2.2	5.4
1000	4.1E+09	37	1.1	7.0	3.0	7.8
2000	1.6E+10	138	1.1	7.3	4.3	11.0
3000	3.5E+10	303	1.2	7.2	5.4	14.3
4000	6.0E+10	517	1.2	7.2	6.2	17.2
5000	9.6E+10	834	1.2	7.6	7.3	20.0
SRL (unlexicalized)						
250	2.3E+08	2	1.2	8.1	3.4	8.8
500	8.6E+08	6	1.4	8.3	4.4	11.5
1000	3.2E+09	21	1.5	8.6	5.9	16.0
2000	1.3E+10	85	1.7	8.8	8.6	19.8
3000	2.7E+10	183	1.9	8.8	10.3	22.3
4000	4.9E+10	325	2.0	9.1	12.7	26.7
5000	7.7E+10	513	2.1	9.2	14.3	28.9
INEX 2005						
100	1.7E+08	1	13.1	7.5	21.7	6.9
250	8.6E+08	3	25.3	14.6	55.2	24.9
500	3.0E+09	8	43.1	23.1	90.4	43.3
1000	1.2E+10	31	78.8	46.0	156.4	77.5

potential of compressing learning models into equivalent DAGs.

As confirmed by the empirical evaluations SDAG and SDAG+ deliver considerable speedups relative to uSVM and as shown in Tables 3.1 and 3.2 the speedups increase with larger values of the sample size r. As previously discussed in section 3.2.2 using smaller sample sizes r in the approximate CPA introduces an additional parameter to the algorithm. Choosing the value for r to approximate CPA surely depends on the amount of training data but also on the type of the used structures and the general nature of the task. These dependencies are difficult to encode in a theory. Hence, the remaining question is how one should pick the optimal sample size for the learning task at hand. Like with many learning algorithms in machine learning, parameter tuning requires experiments on the validation set. While there is no general strategy for the optimal choice of the sampling size, as shown both theoretically and empirically in [188] the resulting training

Table 3.2: Evaluation using PTK kernel (see the caption of in Table 3.1).

	uSVM		SDAG		SDAG+	
SAMPLE	Δ	t	Δ/Δ_S	t/t_S	Δ/Δ_{S+}	t/t_{S+}
SRL						
250	4.8E+09	23	1.0	1.5	1.7	1.2
500	1.8E+10	90	1.1	1.5	2.1	1.4
1000	7.1E+10	340	1.1	1.5	2.9	1.8
2000	2.7E+11	1781	1.1	2.1	3.9	3.3
3000	5.9E+11	3696	1.1	2.1	4.8	5.1
4000	1.2E+12	5039	1.3	2.1	6.4	5.7
5000	1.9E+12	6687	1.5	2.1	7.6	6.5
SRL (unlexicalized)						
250	4.5E+09	20	1.8	1.7	2.9	2.3
500	8.8E+09	38	1.4	1.8	3.5	2.3
1000	3.3E+10	144	1.2	1.7	3.9	3.1
2000	1.8E+11	804	1.7	2.3	7.7	4.1
3000	4.1E+11	1405	2.1	2.2	9.1	5.4
4000	7.5E+11	2501	1.4	2.3	8.0	6.3
5000	1.0E+12	2822	1.5	2.4	10.8	7.2
INEX 2005						
100	2.3E+09	1	90.7	12.8	144.8	13.4
250	1.3E+10	8	171.6	26.5	283.4	27.4
500	3.3E+10	21	145.3	21.9	243.9	22.3
1000	1.4E+11	82	152.4	21.4	260.4	21.4

and test set errors are stable with respect to changes in the sample size r. Relying on the studies of [132], which carried out extensive studies on the impact of the sample size r on the resulting accuracy and training speed, we found a simple recipe for the choice of the sampling size that works well in practice: on datasets where the number of instances exceeds one million the sample size in the range of 5k to 10k examples provides the optimal accuracy/runtime tradeoff; for the datasets with 100k to 1 million examples, we choose r between 1k and 5k; while for datasets below 100k smaller values for r suffice. The above recommendations reflect our experience with SRL and QC tasks and are likely to provide good performance for other tasks, yet the validation sets should be used if one cares to obtain the optimal performance.

Classification experiments

Regarding classification, we compare SDAG+ with uSVM (see Table 3.3). We carry out learning for various sizes of the training set and perform testing on 10k of data. The values of interest here are the number of nodes in the final model and the testing time.

Table 3.3: Classification speedups for SDAG+ over uSVM using STK kernel. Testing on 10k subset when learning on SRL subsets of varying size (1st column). Time indicated in seconds; comp denotes ratio between the number of nodes in SDAG+ and uSVM models; #SVs- number of support vectors.

DATA	uSVM			SDAG+			
	#SVS	NODES	t	NODES	t_{S+}	COMP	t/t_{S+}
10K	1686	33516	10.8	6350	0.5	5.3	24.0
25K	3392	67840	40.8	14212	1.2	4.8	33.2
50K	5876	117520	82.1	25506	2.9	4.6	28.3
75K	7489	149780	111.7	33552	5.5	4.5	20.5
100K	8674	215764	130.6	39787	6.7	5.4	19.5
250K	11234	224680	172.5	62094	16.8	3.6	10.2
500K	13037	260740	199.0	79978	26.6	3.3	7.5
750K	13270	265400	205.9	91048	33.9	2.9	6.1
1MIL	13912	278240	216.3	97447	39.8	2.9	5.4

As we can see, as the size of the training set increases not only the model becomes larger but also the nodes that end up in the model become sparser. This affects the compression rate of SDAG+ w.r.t. uSVM which results in smaller speedups for larger data.

Finally, in the Table 3.4, we report the results of comparison on 100k of SRL data between SVM, uSVM, SDAG, and SDAG+. This replicates the same setting as in [132]): the sample size is relatively small, i.e., 1k and 5k for 100k and 1 million datasets respectively. Therefore, the test conditions do not emphasize the benefits if the DAG models. Nevertheless, SDAG and SDAG+ algorithms deliver high speedups w.r.t. to uSVM and it becomes much larger when compared to SVM. Thus, DAG compression even on relatively sparse trees from SRL (compared to INEX dataset) as carried out in this experiment delivers very significant computational savings over conventional SVMs. We believe that applying it to very large XML document classification datasets would deliver even higher speedups against widely used in this setting SVM-light algorithm.

Table 3.4: Comparison of SVM, uSVM, SDAG and SDAG+ on 100k and 1mil SRL using STK kernel. For 100k the sample size was fixed at 1000 and for 1mil is 5000 (to replicate the experiment in [132]). The number of iterations is 300. The reported values are training time in minutes; values in parenthesis are relative speedups w.r.t. SVM.

size	SVM	uSVM	SDAG	SDAG+
100k	214	37 (6)	5 (41)	5 (45)
1mil	10705	814 (13)	349 (31)	264 (41)

Table 3.5: Handling class-imbalance problem on TREC 10 (top) YA (bottom). Ratio - proportion of negative examples w.r.t. positive; P/R - precision (P) and recall (R). The bottom row in YA is the performance using bag-of-words features on 75k subset. The reported F_1 scores that do not pass the statistical significance test (with $p \leq 0.05$) are marked by [1] for uSVM+j/uSVM and by [2] for uSVM+j/SVM.

Data	Ratio	uSVM		uSVM+j		SVM	
		F-1	P/R	F-1	P/R	F-1	P/R
Trec 10							
ABBR	1:60	87.5	100.0/77.8	84.2[2]	80.0/88.9	84.2	80.0/88.9
DESC	1:4	96.1	95.0/97.1	96.1[12]	95.0/97.1	94.8	97.7/92.0
ENTY	1:3	72.3	91.8/59.6	79.1	79.6/78.7	80.4	82.2/78.7
HUM	1:3	88.1	98.1/80.0	90.3	94.9/86.2	87.5	88.9/86.2
LOC	1:3	81.4	96.6/70.4	87.0	87.5/86.4	82.6	86.5/79.0
NUM	1:5	86.0	98.9/76.1	91.2	96.1/86.7	89.9	98.9/82.3
Yahoo Answers							
10k	1:1.5	37.4	33.5/42.2	39.1	29.6/57.7	37.9	24.2/87.7
50k	1:2.0	36.5	36.0/36.9	40.6	30.0/62.5	39.6	25.7/86.9
100k	1:2.4	33.4	36.2/31.1	40.2	30.2/59.9	40.3	26.6/83.5
150k	1:2.8	33.5	36.9/30.7	41.0	30.2/64.0	-	-
300k	1:3.4	23.8	40.1/16.9	41.4	30.7/63.8	-	-
BOW	1:2.0	34.2	33.2/35.3	38.1	27.5/61.7	36.3	22.5/93.5

3.7.4 Tuning up Precision and Recall

To measure if the difference in the observed values of F_1 scores of the compared models is statistically significant we employed the implementation [105] of the assumption-free randomization framework [102]. The conclusion about the statistical significance of the difference in F_1 scores of considered models is made by assessing how likely the difference in the randomly shuffled predictions of two models is due to chance. We used the default number of 10,000 shuffles for each measurement.

We first report experimental results on question classification corpus on six different categories in Table 3.5 (since the dataset is small, we only report the accuracy). For both uSVM and uSVM+j, we fixed the sample size to 100. For uSVM+j, we picked the value of j from {1, 2, 3, 4, 5, 10} and use the best results obtained on the validation set. For SVM, we carried out tuning of j parameter on a validation set. It is important to note that such parameter has slightly different meaning for uSVM+j and SVM. For the former, it controls the bias to reject negative examples during sampling (Alg. 9) to compute CPM, while for the latter it defines the factor by which training errors on positive examples outweigh errors on negative examples.

Analyzing the results from Table 3.5 (top), we can see that uSVM algorithm that uses uniform sampling obtains high Precision, as it minimizes the training error dominated by

examples from negative class. This results in lower values of the Recall. Its rather high F_1 for ABBR dataset shows that the model simply misclassifies the examples from the minority class saturating the Precision. On the other hand, uSVM+j is able to establish a much better balance between Precision and Recall resulting in high F_1 scores across the majority of categories. Also the performance of SVM with the optimal set of parameters suggests that our method has a better capacity to control the imbalance problem than SVM. This can be explained by the fact, as suggested in [179], that $z_i C$ imposes only an upper bound on dual variables α_i, which results in poorer flexibility to control the class-imbalance with the j parameter of SVM.

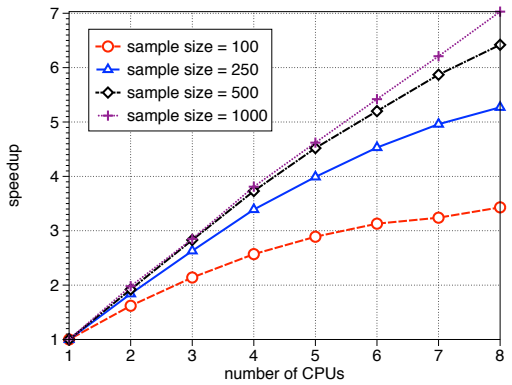
The results on Yahoo! Answers are displayed in Table 3.5 (bottom). For uSVM and uSVM+j, we fix the sample size at 500. Due to the constant time scaling behavior of uSVM [188], the training time for both uSVM and uSVM+j was slightly less than 10 hours across all subsets reported here. While being faster on small subsets of 5k, 10k and 25k, SVM begins to scale poorly on the subsets larger than 50k. Indeed, as studied in [132, 188], CPA with sampling begins to outperform SVM starting from datasets of moderate size (around 50k in our experiments). SVM did not finish the training within 5 days for 150k and 300k subsets, hence there are missing values. We set the value of j parameter for uSVM+j equal to the ratio of negative to positive examples. This natural setting of j parameter for uSVM+j is driven by the intuition to make the distribution of examples from different classes approximately balanced inside each sample, such that the classifier learns on a balanced data. As one can see, this gives much better trade-off between Precision and Recall compared to uSVM. Looking at the results of SVM, we conjecture that here j parameter, similar to the results in previous experiments, is not flexible enough to deliver the optimal P/R trade-off. Also note that training SVM on 100k subset requires almost 4 days, which makes uSVM+j a viable tool for advanced text classification on large datasets, where obtaining optimal balance between Precision and Recall is hindered by the class imbalance problem.

The bottom row of Table 3.5 reports the results using bag-of-words (BOW) feature representation on 75k subset. We note that STK delivers an interesting 12% of relative improvement over BOW model on SVM. However, the main goal of this experiment was not to obtain the top classification performance on such noisy web data but rather to demonstrate that uSVM+j can efficiently deal with large imbalanced data.

3.7.5 Parallelization

To assess the effects of parallelization, we tested parallel versions of SDAG and SDAG+ on 50k subset of Yahoo! Answers dataset using up to 8 CPUs. The achieved speedups over the sequential algorithm are reported in Figure 3.3, where each curve corresponds to runtimes using different sample sizes: {100, 250, 500, 1000}. Increasing the sample size leads to the increase of the time spent to compute CPM, which makes the speedup achieved by

Figure 3.3: Speedups due to parallelizing SDAG/SDAG+ on 50k Yahoo! Answers dataset.



parallelization for large sample sizes even more significant. Using the maximum number of 8 CPUs, we are able to achieve the speedup factor of about 7.0 (using sample size equal to 1000). Since classification can also be easily parallelized, we could experiment with larger sample sizes to obtain a more accurate model.

3.7.6 Linearization experiments

Data and task

We use the Semantic Role Labeling dataset as described in Sec. 3.7.2 Again, we focus on the task of argument identification (i.e., identification of the exact sequence of words spanning an argument). This corresponds to the binary classification of parse tree nodes into correct or not correct boundaries. The models are trained on the subsets {250k, 500k, 1mil, and 4mil}. To evaluate the learned models we report the F1[11] on two sections: 23 and 24 (used as development and test sets in CoNLL-2005 task, respectively), that contain 230k and 150k examples respectively.

Setup

For all our experiments we used Syntactic Tree Kernel (STK) [32]. Learning in the linear input space is carried out with LibLinear solver [43]. Models that use tree kernels and their combination with feature vectors are trained by SDAG software[12] [131], which couples approximate cutting plane algorithm and the DAG approach. It has an additional parameter q that controls the number of examples used to approximate cuts at each

[11] The reported scores correspond to the accuracy of the binary classifier, which upper bounds than the accuracy of the overall boundary detection due to errors in parsing.

[12] We modified the software available at http://disi.unitn.it/~severyn/code.html

iteration. We fix q at {1k, 5k, and 10k} for training on {500k, 1mil, and 4mil} subsets of data, respectively.

To contrast the training time and accuracy obtained by SDAG, we use the SVM-Light-TK[13] [94] (henceforth, referred to as SVM), which encodes tree kernels into the SVM-Light solver [65]. To linearize the kernel space using the learned SVM model we use reverse-kernel engineering framework FLINK[14] [113]. For the kernel space mining procedure we set the minimum fragment frequency to 3 and the threshold factor to 1000.

Feature-based learning

Manually designing useful feature sets for complex tasks such as SRL is a highly nontrivial task. Most of the feature-based learning systems for SRL rely on the linear features extracted from the syntactic parse trees as described in [50], which stresses the necessity and importance of syntactic parsing to derive the best features. We use these manually derived features (MF) as a preliminary baseline to compare and augment more feature-rich TK models.

As an alternative to manually designed features, we first study a straightforward approach to linearize the input space by simply enumerating all the STK features (henceforth, referred to as ETK). It is important to note that we work on already preselected parts of the parse trees, i.e., AST subtrees that by design contain predicate and target argument nodes. This is an essential pre-filtering step for only selecting the most relevant parts of the full parse trees. Features are generated by considering all STK fragments rooted at each node of a tree up to a given depth d, s.t. each fragment contains a given node as a root and all its descendants within the depth d. The number of generated fragments increases exponentially with d, hence to avoid long pre-processing times required to enumerate all possible STK features (which can be billions) we consider fragments up to $d = 5$. For example, for a tree with just 80 nodes for $d \in \{2, 3, 4, 5\}$ the number of generated STK features is respectively {340, 20k, 350k, 16mil}. Hence, to further speed up the feature generation for values of $d > 3$ we limit the maximum number of features each node in a tree can generate to 10k. To get a numeric representation of each tree fragment we hash its string representation and project the obtained numeric value into the linear feature space of the dimensionality 2^k. The obtained linear feature vectors are further normalized. We performed 5-fold cross-validation with different values of $k \in \{16, 18, 20, 22\}$ and found that $k = 20$ (which encodes up to 1 million features) to yield the most accurate results. Note that feature hashing approaches, e.g. [146], provide an efficient way to reduce the dimensionality of the resulting feature vector, while enumerating an exponential number of all tree fragments is infeasible. In the next section we discuss a principled approach to greatly reduce the space of considered tree fragments by

[13] http://disi.unitn.it/moschitti/Tree-Kernel.htm
[14] http://danielepighin.net/cms/software/flink

d	250k		500k		1mil	
	P/R	F1	P/R	F1	P/R	F1
2	82.7/70.9	76.4	83.2/70.7	76.5	83.0/77.1	79.9
3	83.5/67.4	74.6	83.6/72.6	77.7	83.5/75.0	79.0
4	84.0/57.5	68.2	83.9/63.9	72.5	83.8/68.2	75.2
5	84.9/53.5	65.6	84.1/59.6	69.8	83.7/63.6	72.3

Table 3.6: Accuracy of LibLinear models for ETK model on Section 24 of the SRL dataset.

relying on the relevance weights derived from a pre-trained SVM model.

Table 3.6 presents the results of training using ETK models on datasets containing {250k, 500k, and 1mil} examples w.r.t. the maximum depth of the generated fragments. As we can see, the recall of the classifiers goes down with larger values of d, hence negatively affecting the F1 score. This demonstrates the drawback of the feature hashing approach failing to deal with exponential feature spaces where few relevant features may get lost due to the hash collisions. Similar findings for SRL were reported in [80].

Interestingly, as demonstrated by the learning curves in Fig. 3.4, the features extracted by enumerating TK fragments ($\text{LibLinear}_{\text{ETK}}$)[15] still greatly outperforms carefully designed manual features ($\text{LibLinear}_{\text{MF}}$) used in many state-of-the-art SRL systems. Nevertheless, much better accuracy obtained by TK learning with SVMs (SVM_{TK}) is a strong indication of the high discriminative power of structural features. Unfortunately, quadratic scaling behavior of conventional algorithms to train kernelized SVMs prevented us to carry out experiments on data larger than 1 million.

In the next experiment we use SDAG to make the training with tree kernels tractable on large data. SDAG achieves its speedup due to its faster approximate cutting plane algorithm and model compression where trees are kept in an equivalent DAG.

Learning with kernels

While widely applied across many areas of NLP, purely feature-based methods are less expressive in modeling structured features, which have been shown important in complex NLP tasks such as SRL. Although learning with kernels allows for training models with higher discriminative power, it requires much larger training times. First, we contrast the accuracy obtained by learning with kernels w.r.t. to feature-based models presented previously.

Fig. 3.4 shows a learning curve for various models when tested on Section 23 (Section 24 demonstrated similar behavior). We first observe that the baseline MF model, indeed, is the least accurate. A better accuracy can be achieved by applying polynomial kernel (of degree 3), but this incurs considerably larger running times due to learning in kernel

[15] limited to the maximum depth 2

spaces. As confirmed before, the ETK approach works much better than MF, but still is unable to match the accuracy of learning with tree kernels. Importantly, TK models trained by SDAG match the accuracy of SVM. Furthermore, using a combination of TKs with linear or polynomial kernels applied to MF gives the highest F1 of 85.9% (when trained on 4 mil).

Another interesting dimension to compare TK models is to look at the overlap of the most relevant tree fragments extracted by each model. We sort the fragments extracted by each model according to the relevance score (Eq. (3.8)). Table 3.7 reports the overlap percentage of the top $k \in \{1k, 5k, 10k\}$ features. Features extracted by SVM serve as the baseline to compare other models. We also include an experiment with perceptron model, which shows that less accurate training algorithm extracts different features. Interestingly, SDAG, a completely different learning algorithm, has 90% feature overlap with SVM for the top 1k features. This is especially surprising considering that the size of the feature space generated with feature enumeration (depth=3) is about 5.5 mil.

k	SDAG	ETK	Perceptron
1,000	90%	17%	69%
5,000	80%	8%	50%
10,000	77%	5%	43%

Table 3.7: Feature overlap for the top k features w.r.t. SVM when trained on 250k instances of SRL.

Next, we consider the training times for learning with kernels. Linear models MF and ETK can be trained in linear time and took less than 5 minutes (for the latter we need to account for the time to extract TK features, which is about 1 hour with $d = 2$). The expressive power of kernelized models comes at a cost of much larger training times. However, as Table 3.8 shows training with SDAG algorithm (using 4 CPUs) makes learning with TK and combinations with feature vectors tractable even on 4 million, which is prohibitively expensive for SVM.

Model	**Time (hours)**	
	1mil	**4mil**
$\text{SDAG}_{\text{MF}^{poly}}$	7.1	13.1
SVM_{TK}	84.1	-
SDAG_{TK}	4.4	7.2
$\text{SDAG}_{\text{TK+MF}}$	4.9	8.1
$\text{SDAG}_{\text{TK+MF}^{poly}}$	9.4	15.3

Table 3.8: Runtimes for kernelized models.

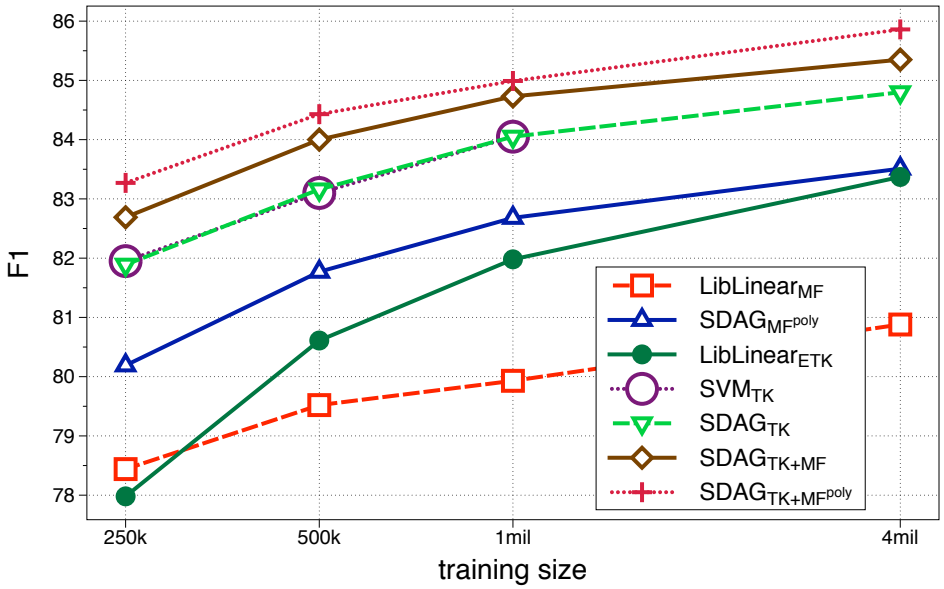


Figure 3.4: Learning curves for models tested on sec23. MF - manual feature vectors, poly - polynomial kernel of degree 3; ETK - enumeration of TKs (depth 2); TK+MF/MF^{poly} combination of TK with linear or polynomial kernel applied to MF. Linear feature models are learned with LibLinear, while TK models are trained with SDAG.

Tree Kernel Linearization

As shown above, fast SDAG algorithm allows for efficient learning of TK models on the dataset of 4 million examples, which is impossible for conventional SVM training algorithms. Nevertheless, the training time for SDAG is still in the range of hours even when run on multiple CPUs. Hence, even using very fast approaches to learning in kernel spaces, the obtained runtimes are far from matching the speed of learning in the linear spaces with linear-time solvers such as LibLinear.

The first step towards reaching the speed of linear-time SVM solvers is to linearize the input structures. Once trees are converted into explicit feature vectors training and testing becomes linear-time. Hence, we require an efficient yet accurate method to perform linearization of TK spaces. Previous experiments with ETK have shown that applying a rather naïve technique to enumerate an exponential number of tree fragments generated by STK suffers from a large drop of accuracy, thus unable to encode complex structural features essential for the task. Moreover the pre-processing time to transform trees to explicit feature vectors exhibits exponential growth with the number of nodes in a tree and the depth of the generated tree fragments.

A more principled approach to linearization of tree kernel spaces, as described in Sec. **??**, is to apply reverse kernel (RKE) engineering where an SVM model is used as a source of fragment weights to guide the greedy feature extraction. Hence, in this exper-

iment our goal is to assess the accuracy of this method when linearizing TK spaces. In the following experiment we first verify that indeed, linearizing the input feature spaces generated by tree kernels does not incur substantial loss of accuracy and is in the line with the results demonstrated in [113]. Table 3.9 reports the accuracy of linearized models using RKE when fed with TK models trained by SVM and SDAG. As we can see, RKE does a good job on extracting most relevant features generated by TKs when converting to the explicit feature vector representation.

While shown to be accurate, the greedy feature mining requires to train a TK model. Hence, the total runtime of the entire linearization procedure is lower-bound by the time to learn a model, which is rather large even when using SDAG. Conversely, the complexity of the successive steps in the linearization pipeline are negligibly small: mining most relevant tree fragments from the model learned on 4 million (about 250k support vectors) takes about 2 minutes. Linearizing 4 million examples took only 3 minutes. Finally, training and classification with the linearized datasets using LibLinear takes 2 minutes. Nevertheless, the bottleneck runtime imposed by learning a TK model can be overcome, which we explore in the next set of experiments.

Model	**1mil**		**4mil**	
	sec23	**sec24**	**sec23**	**sec24**
SVM_{TK}	84.1	81.8	-	-
RKE-SVM_{TK}	84.0	81.3	-	-
$SDAG_{TK}$	84.0	81.5	84.8	82.6
RKE-$SDAG_{TK}$	83.9	81.3	84.5	82.5

Table 3.9: RKE applied to TK models trained by SVM and SDAG. The reported scores are F1-metric. Linearized models are trained with LibLinear.

Distributed linearization

In the following, we take advantage of the observation (also explored in [113]), where hefty training of the SVM model is sped up by splitting the training set into smaller subsets. Previous work in [54] suggests that support vectors collected from locally learned models can encode many of the relevant features retained by models learned globally. Thus, the quadratic scaling behavior of SVMs with kernels can be conquered by carrying out learning on much smaller subsets of the data, which also allows for learning the individual models in parallel. This approach is justified by the fact that the main purpose of the SVM learning in the original kernel space is not to provide the final model that will be used for classification on the test data but is rather used to drive the feature mining process to extract the most relevant features. Hence, the sub-optimal fragment weights derived from the local models learned on the subsets of the original training set still carry

Model	Time (min)	sec23	sec24
SVM_{TK}	57 / 233	84.2	82.2
$SDAG_{TK}$ (q=250)	3 / 17	84.4	82.4
$SDAG_{TK}$ (q=500)	7 / 33	84.4	82.4
$SDAG_{TK}$ (q=1000)	11 / 49	84.5	82.5

Table 3.10: RKE with local models on 4 mil of SRL ($n = 40$ subsets). First value in the 2nd column indicates the time to learn a single model of 100k, while the second value is the overall time for 4 mil (from learning a TK model to linearization and training in the linear space).

enough information to extract highly discriminative features.

Furthermore, splitting the training data into smaller subsets allows for setting up a fully distributed system, e.g., MapReduce. Each subset of the input data is mapped to individual workers. Locally learned models are recombined and used in the feature extraction and further linearization of the training and test data. Finally, fast SVM solvers are used to produce the final linear model for testing.

To test the efficacy of the distributed linearization approach we split the entire dataset of 4mil into $n \in \{10, 20, 40, \text{and } 80\}$ subsets, which corresponds to learning on the subsets of size $s \in \{400\text{k}, 200\text{k}, 100\text{k} \text{ and } 50\text{k}\}$ respectively. We have found that learning local models on subsets smaller than 100k lead to a small drop of the final accuracy of the linearized model, while using larger subsets did not provide any further improvement. Hence, we fix $n = 40$. Table 3.10 reports the running times to train a local model on a single subset of 100k and the overall time of the linearization approach (from learning a TK model to linearization and training in the linear space) on the entire dataset of 4 million. Since learning of local models is fully independent: all 40 jobs were distributed among 10 CPUs, which cut down the total time by 4x. We see that SDAG delivers much faster training times. Interestingly, by using smaller sample sizes to approximate the cutting planes inside the SDAG algorithm, we obtain the same final accuracy while reducing the training time to only 3 minutes for learning one local model.

Hence, the proposed approach achieves remarkably low runtime of only 17 minutes for the full set of activities required by linearization process on the entire 4 million dataset.

3.8 Related work

To improve the scaling properties of SVM-light, a number of efficient algorithms using CPA-based algorithms have been proposed. For example, SVM^{perf} [69] exhibits linear computational complexity in the number of examples when linear kernels are used. While CPA-based approaches deliver state of the art performance w.r.t. accuracy and training time, they scale well only when linear kernels are used. The problem of efficient kernel learning for CPA has been studied in [67], where cutting plane models are compacted by

extracting basis vectors. This, however, leads to a non-trivial optimization problem when arbitrary kernel functions are applied. Another approximate technique to speed up the computation time of tree kernels by comparing only a sparse subset of relevant subtrees is discussed in [124].

The recent advances in the optimization theory suggest that the exponential number of constraints in the optimization problem 3.1 can be tackled by the symmetry breaking (integer) linear programming solvers. It would be interesting to see the connection with symmetry collapsing approaches, e.g. [20], as an alternative way to handle redundancies (symmetries) in the data to efficiently solve OP3.1.

Regarding learning with structural kernels, compact representation of tree forests offered by DAGs was applied for speeding up training of the voted perceptron algorithm in [5]. Another interesting idea of hash kernels for structured data is proposed in [146], where hashing can generate explicit vector representation such that linear learning methods can be applied. However, it is likely that hashing all possible substructures generated by STK, which is exponential in the tree length, will make the preprocessing step too expensive. Also, due to hash collisions, this method computes approximate kernel values and its implications on the accuracy need to be studied more extensively.

A highly efficient subtree kernel on graphs that exploits the idea from Weisfeiler-Lehman test of graph isomorphism is proposed in [145]. While, it has been shown to work well on various graph datasets from bioinformatics, the subtree feature space generated by this kernel is inferior to more general STK and PTK, as its feature generation mechanism includes uniformly all the nodes in the neighborhood of a currently considered node within a given radius, i.e. it does not allow for incomplete tree fragments.

In [113], a more principled feature extraction algorithm to linearize TK spaces has been proposed. Its soundness is justified by the norm-preservation of the model learned by an SVM to extract the most relevant features. The post-analysis of the most relevant tree fragments extracted by the algorithm on the SRL task reveals that fairly complex structures with long term dependencies between the sentence constituents are pertinent for the SVM learner. This suggests that naïve feature enumeration coupled with feature hashing to reduce the effective dimension of the resulting feature vectors will result in lower performance on complex tasks such as SRL, were a few complex features provide essential discriminative power to the classifier.

Concerning class-imbalance problem for SVM learning, the most widely adopted method is to introduce different cost factors in the objective function s.t. the training errors for positive and negative examples receive different penalties [168]. This approach is implemented as the j option in SVM-light [65] that has a super-linear scaling behavior, which prohibits its use on large datasets. Our approach to accomplish cost-sensitive classification shares the idea of reductions put forward in [190] together with the benefit of the conventional approach in SVMs [168] to incorporate importance weights directly into the

optimization process.

Linearization methods aim to convert exponentially large feature spaces implicitly generated by kernels into explicit vector spaces of a reasonable size (e.g., ranging from a few thousands to a tens of millions of features). For this purpose two important assumptions should hold: (i) it is possible to define greedy algorithms that constructively extract features working in the top-down fashion starting from the most relevant fragments and progressing to the least relevant ones; and (ii) only a small percentage of the features suffices to train accurate models.

The approach in [79] is based on the first hypothesis. The authors suggest that a similar method to the PrefixSpan algorithm [111] can be applied to tree kernels. However, it is efficient only on a limited size of support examples. When the number of support vectors becomes very large, e.g., millions of instances, mining the most frequent substructures [192] becomes slow. Additionally, the assumption of point (i), the most frequent tree is the most relevant, is problematic. Suzuki and Isozaki [158] present a feature selection method for convolution kernels based on a distribution-driven relevance assessment. The kernel function is extended to embed substructure mining and techniques for the evaluation of a fragment's χ^2.

Another recent set of approaches is based on feature hashing, e.g. [146]. It enables a large number of features to be generated and efficiently stored in feature vectors of limited size (i.e., millions of dimensions). The main idea is that features hashing to the same value will contribute to the same component of a feature vector. The resulting information loss due to the collisions is supposed to be backed up by the assumption in point (ii).

A rather comprehensive overview of feature selection techniques is carried out in [56]. However, most of them cannot be applied to the large-scale learning in implicit kernel spaces.

3.9 Summary

In this chapter we have presented several techniques to make learning with SVMs and convolution tree kernels applicable to a larger set of real-world applications. Firstly, we have defined a generalized theory and methods for using DAG kernels in the CPA algorithm with sampling. We have proved that our approach can be applied to any tree kernel computable by summing over $\Delta(n_1, n_2)$, where n_1 and n_2 are pairs of nodes from two trees (Th.1).

Secondly, we verified the theory above by modeling and implementing two algorithms: SDAG and SDAG+. The former compresses only the current CPM whereas the latter compacts the entire set of CPMs built so far during the learning of CPA. Both algorithms were used with STK and PTK that clearly satisfy the hypothesis of Th.1.

Thirdly, as PTK considers any node-child subsets to represent trees, we modified the

organization of the DAG structure which results in different levels of compression. Consequently, we extensively studied the effects of using PTK when compacting tree forests into DAGs. In particular, we analyzed the efficiency of our algorithm based on the number of calls to Δ function to exactly verify the speedup independently of the hardware used in the experiments.

Additionally, we also included a parallelization approach and a method for handling imbalanced datasets in SDAG and SDAG+.

Finally, we have experimented with the models above on four datasets: (i) two versions of SRL data, lexicalized and unlexicalized trees; (ii) a question classification dataset; (iii) question and answer pairs from Yahoo! answers; and (vi) a new dataset from INEX [165].

We evaluated the speedup in terms of the training time and the number of Δ-iterations for both STK and the newly proposed PTK for SDAG and SDAG+ on the above datasets. The results have shown that: (1) our approach generalizes to most of tree-based kernels as we obtain significant speedup of PTK-based learning; and (2) when the training data is relatively small (up[16] to 100k) the compactness of the SDAG+ models allows for better usage of the CPU cache, amplifying the benefit of our approach; (3) the results on the NLP tasks underrepresent the potential of our approach as the subtrees are based on words, which make subtrees sparser. Indeed, the results on INEX show speedup up to 283 in terms of Δ computations and up to 77 in runtime.

We have also experimented with learning in linear spaces using manual features, linearized kernel spaces through hashing methods, reverse kernel engineering and approximate cutting plane training with DAG model compression.

Our findings reveal that on a high-level semantic task such as SRL: (i) the naïve approach of enumerating all possible sub-structures becomes intractable and hashed feature vectors fail to achieve both the same accuracy of tree kernels and high efficiency. (ii) In contrast, SDAG allows for achieving the same accuracy as SVMs and makes learning practical. However, the classification and learning time may still not be appealing for large-scale experiments. (iii) Linearization with RKE is rather effective as again there is almost no loss in accuracy and it benefits from extracting complex and highly discriminative features derived from learning in kernel spaces.

As a result, we derive an efficient approach to kernel learning: applying reverse-kernel engineering directly on the SDAG model. This alleviates the major computational bottleneck of the original approach, where traditional SVM training was used. Interestingly, the extracted features have high overlap with the baseline SVM. Additionally, we achieve a significant speedup with almost no loss in accuracy by splitting the data into smaller subsets. This allows for more efficient kernel space learning in a fully distributed manner.

Summing up, we can train an accurate tree kernel model on 4 million instances from SRL, in less than 20 minutes using 10 CPUs. We achieved F1 of 84.5% on Section 23,

[16] Of course, this mainly depends on the hardware characteristics.

which is the state-of-the-art performance of the binary classifier for boundary detection without using ensembles of learners and relying only one a single source of the syntactic information from the parse trees.

Our study opens several future research directions: application of tree kernels to many tasks, where large data size has prevented their use. This surely regards SRL in many languages but also parse tree re-ranking [32] and question answering applications. Also applications to other data mining tasks would be interesting, e.g., XML tree classification.

From the algorithmic perspective, it would be promising to explore approaches to prune the DAGs for achieving higher compression rates without any loss in accuracy. Finally, the ultimate goal would be to use tree kernels for structured output prediction.

Chapter 4

Modelling YouTube comments with Shallow Syntactic Tree Structures

In this chapter we demonstrate how tree kernel learning technology, and, in particular, carefully designed syntactic trees enriched with additional semantic information, can be effectively used to tackle the task of Opinion Mining on Youtube comments. In particular, we define a systematic approach to Opinion Mining (OM) on YouTube comments by (i) modeling classifiers for predicting the opinion polarity and the type of comment and (ii) proposing robust shallow syntactic structures for improving model adaptability. We rely on the tree kernel technology to automatically extract and learn features with better generalization power than conventional feature-based models. An extensive empirical evaluation on our manually annotated YouTube comments corpus shows a high classification accuracy and highlights the benefits of structural models in a cross-domain setting.

4.1 Overview

Social media such as Twitter, Facebook or YouTube contain rapidly changing information generated by millions of users that can dramatically affect the reputation of a person or an organization. This raises the importance of automatic extraction of sentiments and opinions expressed in social media.

YouTube is a unique environment, just like Twitter, but probably even richer: multimodal, with a social graph, and discussions between people sharing an interest. Hence, doing sentiment research in such an environment is highly relevant for the community. While the linguistic conventions used on Twitter and YouTube indeed show similarities [13], focusing on YouTube allows to exploit context information, possibly also multimodal information, not available in isolated tweets, thus rendering it a valuable resource for the future research.

Nevertheless, there is almost no work showing effective OM on YouTube comments. To the best of our knowledge, the only exception is given by the classification system of YouTube comments proposed by [147].

While previous state-of-the-art models for opinion classification have been successfully applied to traditional corpora [108], YouTube comments pose additional challenges: (i) polarity words can refer to either video or product while expressing contrasting sentiments; (ii) many comments are unrelated or contain spam; and (iii) learning supervised models requires training data for each different YouTube domain, e.g., *tablets*, *automobiles*, etc. For example, consider a typical comment on a YouTube review video about a *Motorola Xoom* tablet:

> *this guy really puts a negative spin on this , and I 'm not sure why , this seems crazy fast , and I 'm not entirely sure why his pinch to zoom his laggy all the other* ***xoom*** *reviews*

The comment contains a product name *xoom* and some negative expressions, thus, a bag-of-words model would derive a negative polarity for this product. In contrast, the opinion towards the product is neutral as the negative sentiment is expressed towards the video. Similarly, the following comment:

> *iPad 2 is better. the superior apps just destroy the* ***xoom****.*

contains two positive and one negative word, yet the sentiment towards the product is negative (the negative word *destroy* refers to *Xoom*). Clearly, the bag-of-words lacks the structural information linking the sentiment with the target product.

4.1.1 Our contributions

In this chapter, we carry out a systematic study on OM targeting YouTube comments; its contribution is three-fold:

- We design novel structural representation, based on shallow syntactic trees enriched with conceptual information (Sec. 4.2.2), i.e., tags generalizing the specific topic of the video, e.g., *iPad*, *Kindle*, *Toyota Camry*. Given the complexity and the novelty of the task, we exploit structural kernels to automatically engineer novel features. In particular, we define an efficient tree kernel derived from the Partial Tree Kernel, [93], suitable for encoding structural representation of comments into Support Vector Machines (SVMs).

- To effectively tackle the problem of Opinion Mining on Youtube comments, we define a classification schema (using tree kernel learning technology), which separates spam and not related comments from the informative ones, which are, in turn, further categorized into video- or product-related comments (type classification). At the final stage, different classifiers assign polarity (positive, negative or neutral) to each

type of a meaningful comment. This allows us to filter out irrelevant comments, providing accurate OM distinguishing comments about the video and the target product.

- We create and annotate (by an expert coder) of a comment corpus containing 35k manually labeled comments for two product YouTube domains: *tablets* and *automobiles*.[1] It is the first manually annotated corpus that enables researchers to use supervised methods on YouTube for comment classification and opinion analysis. The comments from different product domains exhibit different properties (cf. Sec. 4.4.2), which give the possibility to study the domain adaptability of the supervised models by training on one category and testing on the other (and vice versa).

Finally, our results show that our models are adaptable, especially when the structural information is used. Structural models generally improve on both tasks – polarity and type classification – yielding up to 30% of relative improvement, when little data is available. Hence, the impractical task of annotating data for each YouTube category can be mitigated by the use of models that adapt better across domains.

The remainder of this chapter is structured as follows. Section 4.2 introduces our baseline models and structured representation, Section 4.3 introduces our corpus, Section 4.4 describes our experiments. Section 4.5 discusses the related work, and Section 4.6 provides a summary and directions for future research.

4.2 Representations and models

Our approach to OM on YouTube relies on the design of classifiers to predict comment type and opinion polarity. Such classifiers are traditionally based on bag-of-words and more advanced features. In the next sections, we define a baseline feature vector model and a novel structural model based on tree kernel methods.

4.2.1 Feature Set

We enrich the traditional bag-of-word representation with features from a sentiment lexicon and features quantifying the negation present in the comment. Our model (`FVEC`) encodes each document using the following feature groups:
- **word n-grams**: we compute unigrams and bigrams over lower-cased word lemmas where binary values are used to indicate the presence/absence of a given item.
- **lexicon**: a sentiment lexicon is a collection of words associated with a positive or negative sentiment. We use two manually constructed sentiment lexicons that are freely available: the MPQA Lexicon [177] and the lexicon of [61]. For each of the lexicons, we

[1]The corpus and the annotation guidelines are publicly available at: http://projects.disi.unitn.it/iKernels/projects/sentube/

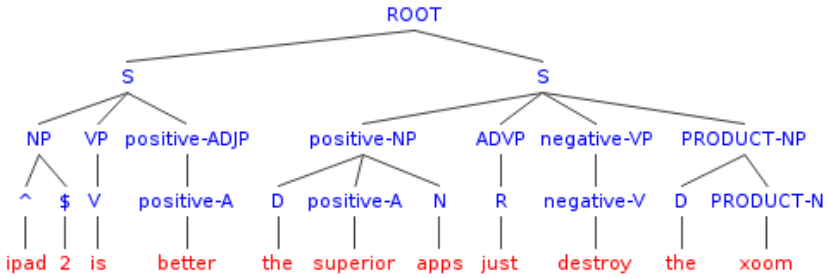


Figure 4.1: Shallow tree representation of the example comment (labeled with `product` type and `negative` sentiment): *"iPad 2 is better. the superior apps just destroy the xoom."* (lemmas are replaced with words for readability) taken from the video "Motorola Xoom Review". We introduce additional tags in the tree nodes to encode the central concept of the video (motorola *xoom*) and sentiment-bearing words *(better, superior, destroy)* directly in the tree nodes. For the former we add a `PRODUCT` tag on the chunk and part-of-speech nodes of the word *xoom*) and polarity tags (*positive* and *negative*) for the latter. Two sentences are split into separate root nodes `S`.

use the number of words found in the comment that have *positive* and *negative* sentiment as a feature.

- **negation**: the count of negation words, e.g., {*don't, never, not, etc.*}, found in a comment.[2] Our structural representation (defined next) enables a more involved treatment of negation.

- **video concept**: cosine similarity between a comment and the title/description of the video. Most of the videos come with a title and a short description, which can be used to encode the topicality of each comment by looking at their overlap.

4.2.2 Structural model

We go beyond traditional feature vectors by employing structural models (`STRUCT`), which encode each comment into a shallow syntactic tree. These trees are input to tree kernel functions for generating structural features. Our structures are specifically adapted to the noisy user-generated texts and encode important aspects of the comments, e.g., words from the sentiment lexicons, product concepts and negation words, which specifically targets the sentiment and comment type classification tasks.

In particular, our shallow tree structure is a two-level syntactic hierarchy built from word lemmas (leaves) and part-of-speech tags that are further grouped into chunks (Fig. 5.2). As full syntactic parsers such as constituency or dependency tree parsers would significantly degrade in performance on noisy texts, e.g., Twitter or YouTube comments, we opted for shallow structures, which rely on simpler and more robust components: a part-of-speech tagger and a chunker. Moreover, such taggers have been recently updated with models [51, 127] trained specifically to process noisy texts showing significant reductions in the error rate on user-generated texts, e.g., Twitter. Hence, we use the CMU Twitter

[2]The list of negation words is adopted from http://sentiment.christopherpotts.net/lingstruc.html

pos-tagger [51,104] to obtain the part-of-speech tags. Our second component – chunker – is taken from [127], which also comes with a model trained on Twitter data[3] and shown to perform better on noisy data such as user comments.

To address the specifics of OM tasks on YouTube comments, we enrich syntactic trees with semantic tags to encode: (i) central concepts of the video, (ii) sentiment-bearing words expressing *positive* or *negative* sentiment and (iii) negation words. To automatically identify concept words of the video we use context words (tokens detected as nouns by the part-of-speech tagger) from the video title and video description and match them in the tree. For the matched words, we enrich labels of their parent nodes (part-of-speech and chunk) with the `PRODUCT` tag. Similarly, the nodes associated with words found in the sentiment lexicon are enriched with a polarity tag (either *positive* or *negative*), while negation words are labeled with the `NEG` tag. It should be noted that vector-based (`FVEC`) model relies only on feature counts whereas the proposed tree encodes powerful contextual syntactic features in terms of tree fragments. The latter are automatically generated and learned by SVMs with expressive tree kernels.

For example, the comment in Figure 5.2 shows two *positive* and one *negative* word from the sentiment lexicon. This would strongly bias the `FVEC` sentiment classifier to assign a `positive` label to the comment. In contrast, the `STRUCT` model relies on the fact that the negative word, *destroy*, refers to the PRODUCT (*xoom*) since they form a verbal phase (VP). In other words, the tree fragment: `[S [negative-VP [negative-V [destroy]] [PRODUCT-NP [PRODUCT-N [xoom]]]]` is a strong feature (induced by tree kernels) to help the classifier to discriminate such hard cases. Moreover, tree kernels generate all possible subtrees, thus producing generalized (back-off) features, e.g., `[S [negative-VP [negative-V [destroy]] [PRODUCT-NP]]]` or `[S [negative-VP [PRODUCT-NP]]]`.

4.2.3 Learning

We perform OM on YouTube using supervised methods, e.g., SVM. Our goal is to learn a model to automatically detect the sentiment and type of each comment. For this purpose, we build a multi-class classifier using the one-vs-all scheme. A binary classifier is trained for each of the classes and the predicted class is obtained by taking a class from the classifier with a maximum prediction score. Our back-end binary classifier is SVM-light-TK[4], which encodes structural kernels in the SVM-light [68] solver. We define a novel and efficient tree kernel function, namely, **Sh**allow syntactic **T**ree **K**ernel (SHTK), which is as expressive as the Partial Tree Kernel (PTK) [93] to handle feature engineering over the structural representations of the `STRUCT` model. A polynomial kernel of degree 3 is applied to feature vectors (`FVEC`).

[3]The chunker from [127] relies on its own POS tagger, however, in our structural representations we favor the POS tags from the CMU Twitter tagger and take only the chunk tags from the chunker.

[4]http://disi.unitn.it/moschitti/Tree-Kernel.htm

Combining structural and vector models. A typical kernel machine, e.g., SVM, classifies a test input $\boldsymbol{x}$ using the following prediction function: $h(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i)$, where α_i are the model parameters estimated from the training data, y_i are target variables, $\boldsymbol{x}_i$ are support vectors, and $K(\cdot, \cdot)$ is a kernel function. The latter computes the *similarity* between two comments. The `STRUCT` model treats each comment as a tuple $\boldsymbol{x} = \langle \boldsymbol{T}, \boldsymbol{v} \rangle$ composed of a shallow syntactic tree $\boldsymbol{T}$ and a feature vector $\boldsymbol{v}$. Hence, for each pair of comments $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we define the following comment similarity kernel:

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = K_{\text{TK}}(\boldsymbol{T}_1, \boldsymbol{T}_2) + K_{\text{v}}(\boldsymbol{v}_1, \boldsymbol{v}_2), \tag{4.1}$$

where K_{TK} computes SHTK (defined next), and K_{v} is a kernel over feature vectors, e.g., linear, polynomial, Gaussian, etc.

Shallow syntactic tree kernel. Following the convolution kernel framework, we define the new SHTK function from Eq. 6.3 to compute the similarity between tree structures. It counts the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. The general equations for Convolution Tree Kernels is:

$$TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2), \tag{4.2}$$

where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2)$ is equal to the number of common fragments rooted in the n_1 and n_2 nodes, according to several possible definition of the atomic fragments.

To improve the speed computation of TK, we consider pairs of nodes (n_1, n_2) belonging to the same tree level. Thus, given H, the height of the `STRUCT` trees, where each level h contains nodes of the same type, i.e., chunk, POS, and lexical nodes, we define SHTK as the following[5]:

$$SHTK(T_1, T_2) = \sum_{h=1}^{H} \sum_{n_1 \in N_{T_1}^h} \sum_{n_2 \in N_{T_2}^h} \Delta(n_1, n_2), \tag{4.3}$$

where $N_{T_1}^h$ and $N_{T_2}^h$ are sets of nodes at height h.

The above equation can be applied with any Δ function. To have a more general and expressive kernel, we use Δ previously defined for PTK. More formally: if n_1 and n_2 are leaves then $\Delta(n_1, n_2) = \mu\lambda(n_1, n_2)$; else $\Delta(n_1, n_2) =$

$$\mu\Big(\lambda^2 + \sum_{\vec{I}_1, \vec{I}_2, |\vec{I}_1| = |\vec{I}_2|} \lambda^{d(\vec{I}_1) + d(\vec{I}_2)} \prod_{j=1}^{|\vec{I}_1|} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big),$$

where $\lambda, \mu \in [0, 1]$ are decay factors; the large sum is adopted from a definition of the subsequence kernel [140] to generate children subsets with gaps, which are then used in a recursive call to Δ. Here, $c_{n_1}(i)$ is the i^{th} child of the node n_1; $\vec{I}_1$ and $\vec{I}_2$ are two

[5] To have a similarity score between 0 and 1, a normalization in the kernel space, i.e. $\frac{SHTK(T_1,T_2)}{\sqrt{SHTK(T_1,T_1) \times SHTK(T_2,T_2)}}$ is applied.

sequences of indexes that enumerate subsets of children with gaps, i.e., $\vec{I} = (i_1, i_2, .., |I|)$, with $1 \leq i_1 < i_2 < .. < i_{|I|}$; and $d(\vec{I}_1) = \vec{I}_{1l(\vec{I}_1)} - \vec{I}_{11} + 1$ and $d(\vec{I}_2) = \vec{I}_{2l(\vec{I}_2)} - \vec{I}_{21} + 1$, which penalizes subsequences with larger gaps.

It should be noted that: firstly, the use of a subsequence kernel makes it possible to generate child subsets of the two nodes, i.e., it allows for gaps, which makes matching of syntactic patterns less rigid. Secondly, the resulting SHTK is essentially a special case of PTK [93], adapted to the shallow structural representation `STRUCT` (see Sec. 4.2.2). When applied to `STRUCT` trees, SHTK exactly computes the same feature space as PTK, but in faster time (on average). Indeed, SHTK required to be only applied to node pairs from the same level (see Eq. 4.3), where the node labels can match – chunk, POS or lexicals. This reduces the time for selecting the matching-node pairs carried out in PTK [93, 94]. The fragment space is obviously the same, as the node labels of different levels in `STRUCT` are different and will not be matched by PTK either.

Finally, given its recursive definition in Eq. 4.3 and the use of subsequence (with gaps), SHTK can derive useful dependencies between its elements. For example, it will generate the following subtree fragments: `[positive-NP [positive-A N]]`, `[S [negative-VP [negative-V [destroy]] [PRODUCT-NP]]]` and so on.

4.3 YouTube comments corpus

To build a corpus of YouTube comments, we focus on a particular set of videos (technical reviews and advertisings) featuring commercial products. In particular, we chose two product categories: automobiles (`AUTO`) and tablets (`TABLETS`). To collect the videos, we compiled a list of products and queried the YouTube gData API[6] to retrieve the videos. We then manually excluded irrelevant videos. For each video, we extracted all available comments (limited to maximum 1k comments per video) and manually annotated each comment with its type and polarity. We distinguish between the following types:

product: discuss the topic product in general or some features of the product;

video: discuss the video or some of its details;

spam: provide advertising and malicious links; and

off-topic: comments that have almost no content ("lmao") or content that is not related to the video ("Thank you!").

Regarding the polarity, we distinguish between {*positive, negative, neutral*} sentiments with respect to the product and the video. If the comment contains several statements of different polarities, it is annotated as both *positive* and *negative*: "Love the video but waiting for iPad 4". In total we have annotated 208 videos with around 35k comments (128 videos `TABLETS` and 80 for `AUTO`).

[6] https://developers.google.com/youtube/v3/

To evaluate the quality of the produced labels, we asked 5 annotators to label a sample set of one hundred comments and measured the agreement. The resulting annotator agreement α value [9, 77] scores are 60.6 (`AUTO`), 72.1 (`TABLETS`) for the sentiment task and 64.1 (`AUTO`), 79.3 (`TABLETS`) for the **type** classification task. For the rest of the comments, we assigned the entire annotation task to a single coder. Further details on the corpus can be found in [167].

4.4 Experiments

This section reports: (i) experiments on individual subtasks of opinion and type classification; (ii) the full task of predicting type and sentiment; (iii) study on the adaptability of our system by learning on one domain and testing on the other; (iv) learning curves that provide an indication on the required amount and type of data and the scalability to other domains.

4.4.1 Task description

Sentiment classification. We treat each comment as expressing `positive`, `negative` or `neutral` sentiment. Hence, the task is a three-way classification.

Type classification. One of the challenging aspects of sentiment analysis of YouTube data is that the comments may express the sentiment not only towards the `product` shown in the video, but also the `video` itself, i.e., users may post positive comments to the video while being generally negative about the product and vice versa. Hence, it is of crucial importance to distinguish between these two types of comments. Additionally, many comments are irrelevant for both the product and the video (`off-topic`) or may even contain `spam`. Given that the main goal of sentiment analysis is to select sentiment-bearing comments and identify their polarity, distinguishing between `off-topic` and `spam` categories is not critical. Thus, we merge the `spam` and `off-topic` into a single `uninformative` category. Similar to the opinion classification task, comment type classification is a multi-class classification with three classes: `video`, `product` and `uninform`.

Full task. While the previously discussed sentiment and type identification tasks are useful to model and study in their own right, our end goal is: given a stream of comments, to jointly predict both the type and the sentiment of each comment. We cast this problem as a single multi-class classification task with seven classes: the Cartesian product between {`product, video`} type labels and {`positive, neutral, negative`} sentiment labels plus the `uninformative` category (*spam* and *off-topic*). Considering a real-life application, it is important not only to detect the polarity of the comment, but to also identify if it is expressed towards the product or the video.[7]

[7] We exclude comments annotated as both `video` and `product`. This enables the use of a simple flat multi-classifiers with seven categories for the full task, instead of a hierarchical multi-label classifiers (i.e., type classification first and then

Task	class	AUTO		TABLETS	
		TRAIN	TEST	TRAIN	TEST
Sentiment	positive	2005 (36%)	807 (27%)	2393 (27%)	1872 (27%)
	neutral	2649 (48%)	1413 (47%)	4683 (53%)	3617 (52%)
	negative	878 (16%)	760 (26%)	1698 (19%)	1471 (21%)
	total	5532	2980	8774	6960
Type	product	2733 (33%)	1761 (34%)	7180 (59%)	5731 (61%)
	video	3008 (36%)	1369 (26%)	2088 (17%)	1674 (18%)
	off-topic	2638 (31%)	2045 (39%)	2334 (19%)	1606 (17%)
	spam	26 (>1%)	17 (>1%)	658 (5%)	361 (4%)
	total	8405	5192	12260	9372
Full	product-pos.	1096 (13%)	517 (10%)	1648 (14%)	1278 (14%)
	product-neu.	908 (11%)	729 (14%)	3681 (31%)	2844 (32%)
	product-neg.	554 (7%)	370 (7%)	1404 (12%)	1209 (14%)
	video-pos.	909 (11%)	290 (6%)	745 (6%)	594 (7%)
	video-neu.	1741 (21%)	683 (14%)	1002 (9%)	773 (9%)
	video-neg.	324 (4%)	390 (8%)	294 (2%)	262 (3%)
	off-topic	2638 (32%)	2045 (41%)	2334 (20%)	1606 (18%)
	spam	26 (>1%)	17 (>1%)	658 (6%)	361 (4%)
	total	8196	5041	11766	8927

Table 4.1: Summary of YouTube comments data used in the sentiment, type and full classification tasks. The comments come from two product categories: AUTO and TABLETS. Numbers in parenthesis show proportion w.r.t. to the total number of comments used in a task.

4.4.2 Data

We split all the videos 50% between training set (TRAIN) and test set (TEST), where each video contains all its comments. This ensures that all comments from the same video appear either in TRAIN or in TEST. Since the number of comments per video varies, the resulting sizes of each set are different (we use the larger split for TRAIN). Table 4.1 shows the data distribution across the task-specific classes – **sentiment** and **type** classification. For the **sentiment** task we exclude `off-topic` and `spam` comments as well as comments with ambiguous sentiment, i.e., annotated as both `positive` and `negative`.

For the **sentiment** task about 50% of the comments have `neutral` polarity, while the `negative` class is much less frequent. Interestingly, the ratios between polarities expressed in comments from `AUTO` and `TABLETS` are very similar across both TRAIN and TEST. Conversely, for the **type** task, we observe that comments from `AUTO` are uniformly distributed among the three classes, while for the `TABLETS` the majority of comments are `product` related. It is likely due to the nature of the `TABLETS` videos, that are more geek-oriented, where users are more prone to share their opinions and enter involved discussions

opinion polarity). The number of comments assigned to both `product` and `video` is relatively small (8% for `TABLETS` and 4% for `AUTO`).

Table 4.2: In-domain experiments on AUTO and TABLETS using two models: FVEC and STRUCT. The results are reported for sentiment, type and full classification tasks. The metrics used are precision (P), recall (R) and F1 for each individual class and the general accuracy of the multi-class classifier (Acc).

Task	Class	AUTO						TABLETS					
		FVEC			STRUCT			FVEC			STRUCT		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
Sent	positive	49.1	72.1	58.4	50.1	73.9	59.0	67.5	70.3	69.9	71.2	71.3	71.3
	neutral	68.2	55.0	61.4	70.1	57.6	63.1	81.3	71.4	76.9	81.1	73.1	77.8
	negative	42.0	36.9	39.6	41.3	35.8	38.8	48.3	60.0	54.8	50.2	62.6	56.5
	Acc			54.7			55.7			68.6			70.5
Type	product	66.8	73.3	69.4	68.8	75.5	71.7	78.2	95.3	86.4	80.1	95.5	87.6
	video	45.0	52.8	48.2	47.8	49.9	48.7	83.6	45.7	58.9	83.5	46.7	59.4
	uninform	59.3	48.2	53.1	60.6	53.0	56.4	70.2	52.5	60.7	72.9	58.6	65.0
	Acc			57.4			59.4			77.2			78.6
Full	product-pos	34.0	49.6	39.2	36.5	51.2	43.0	48.4	56.8	52.0	52.4	59.3	56.4
	product-neu	43.4	31.1	36.1	41.4	36.1	38.4	68.0	67.5	68.1	59.7	83.4	70.0
	product-neg	26.3	29.5	28.8	26.3	25.3	25.6	43.0	49.9	45.4	44.7	53.7	48.4
	video-pos	23.2	47.1	31.9	26.1	54.5	35.5	69.1	60.0	64.7	64.9	68.8	66.4
	video-neu	26.1	30.0	29.0	26.5	31.6	28.8	56.4	32.1	40.0	55.1	35.7	43.3
	video-neg	21.9	3.7	6.0	17.7	2.3	4.8	39.0	17.5	23.9	39.5	6.1	11.5
	uninform	56.5	52.4	54.9	60.0	53.3	56.3	60.0	65.5	62.2	63.3	68.4	66.9
	Acc			40.0			41.5			57.6			60.3

about a product. Additionally, videos from the `AUTO` category (both commercials and user reviews) are more visually captivating and, being generally oriented towards a larger audience, generate more video-related comments. Regarding the **full** setting, where the goal is to have a joint prediction of the comment sentiment and type, we observe that `video-negative` and `video-positive` are the most scarce classes, which makes them the most difficult to predict.

4.4.3 Results

We start off by presenting the results for the traditional in-domain setting, where both TRAIN and TEST come from the same domain, e.g., `AUTO` or `TABLETS`. Next, we show the learning curves to analyze the behavior of `FVEC` and `STRUCT` models according to the training size. Finally, we perform a set of cross-domain experiments that describe the enhanced adaptability of the patterns generated by the `STRUCT` model.

In-domain experiments

We compare `FVEC` and `STRUCT` models on three tasks described in Sec. 4.4.1: sentiment, type and full. Table 4.2 reports the per-class performance and the overall accuracy of the

multi-class classifier. Firstly, we note that the performance on TABLETS is much higher than on AUTO across all tasks. This can be explained by the following: (i) TABLETS contains more training data and (ii) videos from AUTO and TABLETS categories draw different types of audiences – well-informed users and geeks expressing better-motivated opinions about a product for the former vs. more general audience for the latter. This results in the different quality of comments with the AUTO being more challenging to analyze. Secondly, we observe that the STRUCT model provides 1-3% of absolute improvement in accuracy over FVEC for every task. For individual categories the F1 scores are also improved by the STRUCT model (except for the *negative* classes for AUTO, where we see a small drop). We conjecture that sentiment prediction for AUTO category is largely driven by one-shot phrases and statements where it is hard to improve upon the bag-of-words and sentiment lexicon features. In contrast, comments from TABLETS category tend to be more elaborated and well-argumented, thus, benefiting from the expressiveness of the structural representations.

Considering per-class performance, correctly predicting negative sentiment is most difficult for both AUTO and TABLETS, which is probably caused by the smaller proportion of the negative comments in the training set. For the **type** task, video-related class is substantially more difficult than product-related for both categories. For the **full** task, the class video-negative accounts for the largest error. This is confirmed by the results from the previous sentiment and type tasks, where we saw that handling negative sentiment and detecting video-related comments are most difficult.

Learning curves

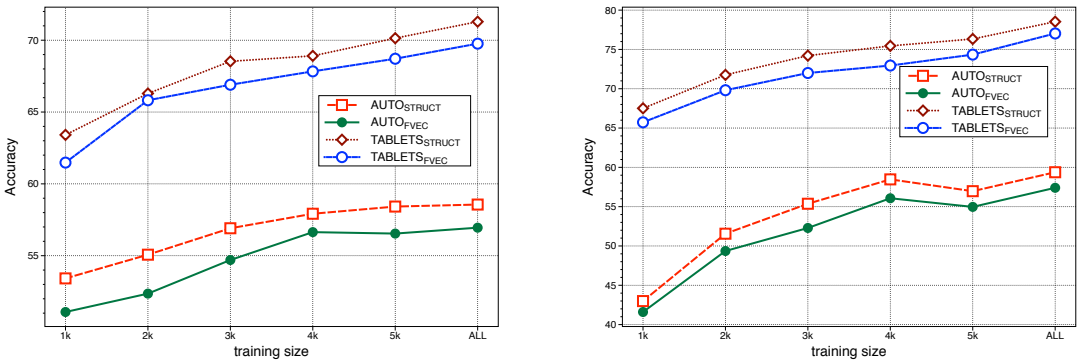
The learning curves depict the behavior of FVEC and STRUCT models as we increase the size of the training set. Intuitively, the STRUCT model relies on more general syntactic patterns and may overcome the sparseness problems incurred by the FVEC model when little training data is available.

Nevertheless, as we see in Figure 4.2, the learning curves for sentiment and type classification tasks across both product categories do not confirm this intuition. The STRUCT model consistently outperforms the FVEC across all training sizes, but the gap in the performance does not increase when we move to smaller training sets. As we will see next, this picture changes when we perform the cross-domain study.

Cross-domain experiments

To understand the performance of our classifiers on other YouTube domains, we perform a set of cross-domain experiments by training on the data from one product category and testing on the other.

Table 4.3 reports the accuracy for three tasks when we use all comments (TRAIN +

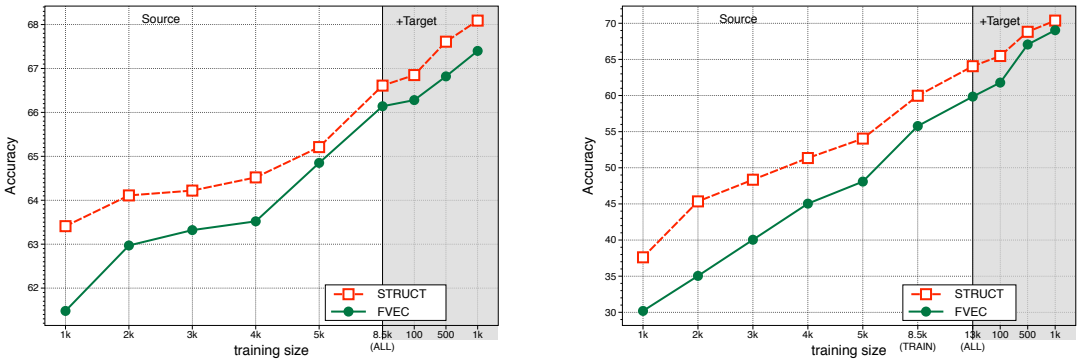


(a) Sentiment classification (b) Type classification

Figure 4.2: In-domain learning curves. ALL refers to the entire TRAIN set for a given product category, i.e., AUTO and TABLETS (see Table 4.1)

Source	Target	Task	FVEC	STRUCT
AUTO	TABLETS	Sent	66.1	66.6
		Type	59.9	64.1†
		Full	35.6	38.3†
TABLETS	AUTO	Sent	60.4	61.9†
		Type	54.2	55.6†
		Full	43.4	44.7†

Table 4.3: Cross-domain experiment. Accuracy using FVEC and STRUCT models when trained/tested in both directions, i.e. AUTO→TABLETS and TABLETS→AUTO. † denotes results statistically significant at 95% level (via pairwise t-test).



(a) Sentiment classification (b) Type classification

Figure 4.3: Learning curves for the cross-domain setting (AUTO→TABLETS). Shaded area refers to adding a small portion of comments from the same domain as the target test data to the training.

TEST) from AUTO to predict on the TEST from TABLETS and in the opposite direction (TABLETS→AUTO). When using AUTO as a source domain, STRUCT model provides additional 1-3% of absolute improvement, except for the sentiment task.

Similar to the in-domain experiments, we studied the effect of the source domain size on the target test performance. This is useful to assess the adaptability of features exploited by the FVEC and STRUCT models with the change in the number of labeled examples available for training. Additionally, we considered a setting including a small amount of training data from the target data (i.e., supervised domain adaptation).

For this purpose, we drew the learning curves of the FVEC and STRUCT models applied to the **sentiment** and **type** tasks (Figure 4.3): AUTO is used as the source domain to train models, which are tested on TABLETS.[8] The plot shows that when little training data is available, the features generated by the STRUCT model exhibit better adaptability (up to 10% of improvement over FVEC). The bag-of-words model seems to be affected by the data sparsity problem which becomes a crucial issue when only a small training set is available. This difference becomes smaller as we add data from the same domain. This is an important advantage of our structural approach, since we cannot realistically expect to obtain manual annotations for 10k+ comments for each (of many thousands) product domains present on YouTube.

4.4.4 Discussion

Our STRUCT model is more accurate since it is able to induce structural patterns of sentiment. Consider the following comment: *optimus pad is better. this xoom is just to bulky but optimus pad offers better functionality.* The FVEC bag-of-words model misclassifies it to be positive, since it contains two positive expressions (*better*, *better functionality*) that outweigh a single negative expression (*bulky*). The structural model, in contrast, is able to identify the product of interest (*xoom*) and associate it with the negative expression through a structural feature and thus correctly classify the comment as negative.

Some issues remain problematic even for the structural model. The largest group of errors are implicit sentiments. Thus, some comments do not contain any explicit positive or negative opinions, but provide detailed and well-argumented criticism, for example, *this phone is heavy.* Such comments might also include irony. To account for these cases, a deep understanding of the product domain is necessary.

4.5 Related work

In the past decade, automatic sentiment analysis of texts has attracted attention from both industry and academia. Such interest has produced a large body of research work, mainly focusing on the use of machine learning algorithms for opinion classification [90, 108].

Most prior work on opinion mining has been performed on more standardized forms of text, such as consumer reviews or newswire. The most commonly used datasets include:

[8] The results for the other direction (TABLETS→AUTO) show similar behavior.

the MPQA corpus of news documents [177], web customer review data [61], Amazon review data [19], the JDPA corpus of blogs [75], etc. However, these corpora are only partially suitable for developing models on social media, since the informal text poses additional challenges for Information Extraction and Natural Language Processing.

Opinion mining on Social Media has started to receive a lot of attention from the scientific community only very recently [22, 101]. Several annotation projects have been proposed to support the development of sentiment analysis models for social media, focusing mainly on Twitter—one of the biggest initiatives being the SemEval 2013 task on the sentiment analysis [101].

Similar to Twitter, most YouTube comments are very short, the language is informal with numerous accidental and deliberate errors and grammatical inconsistencies [13], which make previous corpora less suitable to train models for OM on YouTube. Nevertheless, YouTube is a much less explored social media, with almost no work on sentiment analysis published so far. Siersdorfer et al. [147] focus on exploiting user ratings (the counts of 'thumbs up/down' as flagged by other users) of YouTube video comments to train classifiers to predict the community acceptance of new comments. Their goal is thus different: predicting comment ratings, rather than predicting the sentiment expressed in a YouTube comment or its information content. Exploiting the information from user ratings is a feature that we have not exploited thus far, but we believe that it is a valuable feature to use in future work.

Early studies on opinion mining focused on the *document polarity* classification problem: for a given document, the algorithm assigns a label determining its general attitude (positive, negative or neutral). This formulation, however, is often too simplistic and thus the most recent studies address more fine-grained tasks, including identifying subjective vs. objective parts of a document [107, 187], opinion holders [70] or more complex sentiments and emotions [22], in particular, as irony or sarcasm [26, 123]. In our work, we refine the traditional polarity classification formulation, distinguishing between different sentiment targets (video vs. product). This allows us to provide a better understanding of user-generated comments that may address several topics, expressing different emotions.

Most of the previous work on supervised sentiment analysis use feature vectors to encode documents. Several studies provide in-depth analysis of lexical features for opinion mining [109,126]. Such features can be effectively combined with external information, for example, with personalized co-occurence statistics [110]. Our feature-based baseline model (cf. Section 4.2) is very similar to the best performing system from the SemEval 2013 shared task on Twitter [89].

While a few successful attempts have been made to use more involved linguistic analysis for opinion mining, such as dependency trees [116,159] and constituency trees with vectorized nodes [150], recently, a comprehensive study by Wang and Manning [176] showed that a simple model using bigrams and SVMs performs on par with more complex algorithms.

In contrast, we show that adding structural features (cf. Section 4.2) from syntactic trees is particularly useful for the cross-domain setting (cf. Section 4.4). The structural features help to build a system that is more robust across domains. Therefore, rather than trying to build a specialized system for every new target domain, as it has been done in most prior work on domain adaptation [19, 36], the domain adaptation problem boils down to finding a more robust system [115, 151]. This is in line with recent advances in parsing "The Web" [112], where participants were asked to build a single system able to cope with different yet related domains.

Our approach, which we will describe in detail in the next section, relies on robust syntactic structures to automatically generate patterns that, given our empirical findings, have shown to adapt better. These representations were inspired by the semantic models developed for Question Answering [95, 133, 134] and Semantic Textual Similarity [138]. Moreover, we introduce additional tags, e.g., video concepts, polarity and negation words, to achieve better generalization across different domains, where the word distribution and vocabulary changes.

Most studies on opinion mining, especially for Social Media data, focus on English. Thus, several algorithms have been proposed for detecting opinions in English tweets within the recent SemEval evaluation campaign [101]. We have created a manually annotated corpus (cf. Section 4.3) that can be used by the scientific community for experiments on supervised opinion mining. The previous work was based on automatically extracted data and a small manually tagged test collection. We propose structural models for our data representation and show that they yield superior performance. The previous work adopted a more baseline methodology, relying on term-based opinion scores.

4.6 Summary

We carried out a systematic study on OM from YouTube comments by training a set of supervised multi-class classifiers distinguishing between video and product related opinions. We use standard feature vectors augmented by shallow syntactic trees enriched with additional conceptual information.

In this chapter we make the following contributions: (i) we show that effective OM can be carried out with supervised models trained on high quality annotations; (ii) we introduce a novel annotated corpus of YouTube comments, which we make available for the research community; (iii) we define novel structural models and kernels, which can improve on feature vectors, e.g., up to 30% of relative improvement in type classification, when little data is available, and demonstrates that the structural model scales well to other domains.

In the future, we plan to work on a joint model to classify all the comments of a given video, s.t. it is possible to exploit latent dependencies between entities and the sentiments

of the comment thread. Additionally, we plan to experiment with hierarchical multi-label classifiers for the full task (in place of a flat multi-class learner).

Chapter 5

Modelling text pairs with syntactic trees

Learning to represent pairs of texts is of core importance to IR and NLP and is a key problem in such applications as Machine Translation, Question Answering, Paraphrasing, Textual Entailment, etc. Conventional approaches treat input text pairs as feature vectors where each feature represents a score corresponding to a certain type of similarity between elements of a pair. This approach is conceptually easy to implement although it requires to design a fairly large number of feature extractors that would encode various aspects of similarity. Nevertheless, by using only similarity features to represent a text pair we may loose a considerable amount of relational information encoding connections between texts in a pair that are not straight-forward to directly encode as features.

In this chapter we propose an alternative method to encode text pairs based on the tree kernel learning approach by carefully designing linguistic structures to represent input pairs. The novelty of our approach is that we treat the input text pairs as structural objects and rely on the power of kernel learning to extract relevant structural features. To link the documents in a pair we mark the nodes in the related structures with a special relational tag. This way effective structural relational patterns are implicitly encoded in the trees and can be automatically learned by the kernel-based machines.

We expreriment with our structural learning approach and propose novel syntactic tree representations for two tasks: Semantic Textual Similarity (STS) and Microblog Retrieval.

Different from the majority of approaches to tackle STS problem, where a large number of pairwise similarity features are used to represent a text pair, our structural tree kernel model features the following: (i) it directly encodes input texts into relational syntactic structures; (ii) tree kernels are used to handle feature engineering; (iii) it is easy to combine both structural and feature vector representations in a single scoring model, i.e., in Support Vector Regression (SVR); and (iv) our final model delivers significant improvement over the best STS systems.

Next, we tackle the problem of improving Microblog retrieval algorithms by proposing a robust structural representation of query-tweet pairs. We test the generalization power of our approach on the TREC Microblog 2011 and 2012 tasks. We find that relational syntactic features generated by structural kernels are effective for learning to rank (L2R) and can easily be combined with those of other existing systems to boost their accuracy. In particular, the results show that our L2R approach improves on almost all of the participating systems at TREC, only using their raw output scores as a single feature fed to our model. Our method yields an average increase of 5% in retrieval effectiveness and an average boost of 7 positions in system ranks.

5.1 Overview

In the following we briefly define the problems of STS and Microblog retrieval and describe our contributions.

5.1.1 Semantic Textual Similarity

In STS the goal is to learn a scoring model that given a pair of two short texts returns a similarity score that correlates with human judgement. Hence, the key aspect of having an accurate STS framework is the design of features that can adequately represent various aspects of the similarity between texts, e.g., using lexical, syntactic and semantic similarity metrics.

The majority of approaches treat input text pairs as feature vectors where each feature is a score corresponding to a certain type of similarity. This approach is conceptually easy to implement and the STS shared task at SemEval 2012 [2] (STS-2012) has shown that the best systems were built following this idea, i.e., a number of features encoding similarity of an input text pair were combined in a single scoring model, e.g., SVR. Nevertheless, one limitation of using only similarity features to represent a text pair is that of low representation power.

The novelty of our approach is that we treat the input text pairs as structural objects and rely on the power of kernel learning to extract relevant structures. To link the documents in a pair we mark the nodes in the related structures with a special relational tag. This way effective structural relational patterns are implicitly encoded in the trees and can be automatically learned by the kernel-based machines. We combine our relational structural model with the features from two best systems of STS-2012. Finally, we use the approach of classifier stacking to combine several structural models into the feature vector representation.

5.1.2 Microblog Retrieval

Social media has become part of our daily lives, and is increasingly growing into the main outlet for answering various information needs, e.g., the query, *Facebook privacy*, may be answered by the following tweet: *Facebook Must Explain Privacy Practices to Congress http://sns.ly/2Qbry7*. Such queries have proven difficult to answer with a single retrieval model, and lead to models that learn to combine a large number of rankers.

Learning to rank (L2R) methods have been shown to improve retrieval effectiveness and they have recently been used for ranking short documents from social media. However, L2R suffers from an important drawback: different training data is needed for different applications. The required amount of training data critically depends on the task being tackled and the *quality* of the used text representations, e.g., lexical features are less powerful than search engine scores or other meta-features. Optimal representations require considerable effort to be designed and implemented. Hence, flexible and adaptable features can be valuable for rapid and effective designs of L2R systems.

Previous work has shown that one source of more *adaptable* features comes from structural relations between object pairs [133], which in the case of text mainly refers to its syntactic structure. Unfortunately, the latter is subject to errors when it is automatically generated. This problem is exacerbated when we deal with informal and unedited text typically prominent in social media.

Most importantly, it is not clear which part of the structure should be considered to design effective features.

We tackle the problems noted above in the context of recent TREC Microblog retrieval tasks by proposing relational shallow syntactic structures to represent (query, tweet) pairs.

Instead of trying to explicitly encode salient features from syntactic structures, we opt for a structural kernel learning framework, where the learning algorithm operates in rich feature spaces of tree fragments automatically generated by expressive tree kernel functions.

The following characterizes our approach: (i) it uses shallow syntactic parsers developed for social media, which are robust and shown to be accurate in such domains; (ii) tree kernels implicitly generate all possible tree fragments, thus all of them are used as features by the learning algorithm, solving the problem of engineering task-specific features.

We design experiments using the 2011 and 2012 editions of the TREC Microblog track to verify the following: (i) relational syntactic features produced by a shallow syntactic parser are effective for L2R; (ii) our automatic feature engineering approach based on structural kernels is accurate and produces general features, which are complementary to those typically used in L2R models; and (iii) our structural representations can easily be combined with existing systems to boost their accuracy.

Our results show that employing relational syntactic structures improves on almost all

the participating systems by only using their raw scores along with our L2R model based on relational syntactic structures. Our method boosts retrieval effectiveness by more than 5% on average and improves the rankings of participating systems by at least 7 positions on average.

5.1.3 Our contributions

The contribution of this chapter are as follows:

1. Semantic Textual Similarity:

- We provide a convincing evidence that adding structural features automatically extracted by tree kernels yields significant improvements in accuracy.
- We define a combination kernel that integrates both structural and feature vector representations within a single scoring model, e.g., Support Vector Regression.
- We provide a simple way to construct relational structural models that can be built using off-the-shelf NLP tools.
- We experiment with four structural representations and show that constituency and dependency trees represent the best source for learning structural relationships for STS.
- Using a classifier stacking approach, structural models can be easily combined and integrated into existing feature-based STS models.

2. Microblog Retrieval:

- We show that relational syntactic structures are effective for L2R on noisy social media data.
- Our automatic feature engineering approach based on structural kernels is accurate and produces general features, which are complementary to those typically used in L2R models.
- Our structural representations can be easily combined with existing systems to boost their accuracy.

The remainder of this chapter is structured as follows. Section 5.2 describes our approach to tackle the STS problem, while Section 5.3 describes our syntax-aware reranker for Microblog retrieval. Each of these two sections includes a description of our syntactic representations to represent text pairs (or query-tweet pairs) and the learning process where we rely on tree kernels to automatically extract and learn relational features. Each of the sections reports on the the experimental evaluation and includes a discussion of the obtained results. Section 5.4 provides a summary and directions for future research.

5.2 Semantic Textual Similarity

The approach of relating pairs of input structures by learning predictable syntactic transformations has shown to deliver state-of-the-art results in question answering, recognizing textual entailment, and paraphrase detection, e.g. [58, 174, 175]. Previous work relied on fairly complex approaches, e.g. applying quasi-synchronous grammar formalism and variations of tree edit distance alignments, to extract syntactic patterns relating pairs of input structures. Our approach is conceptually simpler, as it regards the problem within the kernel learning framework, where we first encode salient syntactic/semantic properties of the input text pairs into tree structures and rely on tree kernels to automatically generate rich feature spaces. This work extends in several directions our earlier work in question answering, e.g., [98, 99], in textual entailment recognition, e.g., [92], and more in general in relational text categorization [95, 133].

In this section we describe: (i) our kernel framework to combine structural and vector models; (ii) structural kernels to handle feature engineering; and (iii) suitable structural representations for relational learning.

5.2.1 Learning

A conventional approach in supervised learning to model text pairs is to represent a pair of texts as a set of similarity features $\{f_i\}$, s.t. the predictions are computed as $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_i w_i f_i$, where $\mathbf{w}$ is the model weight vector. Hence, the learning problem boils down to estimating individual weights of each of the similarity features f_i. One downside of such approach is that a great deal of similarity information encoded in a given text pair is lost when modeled by single real-valued scores.

A more versatile approach in terms of the input representation relies on kernels. In a typical kernel learning approach, e.g., SVM, the prediction function for a test input $\mathbf{x}$ takes on the following form $h(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$, where α_i are the model parameters estimated from the training data, y_i are target variables, $\mathbf{x}_i$ are support vectors, and $K(\cdot, \cdot)$ is a kernel function.

To encode both structural representation and similarity feature vectors of a given text pair in a single model we define each document in a pair to be composed of a tree and a vector: $\langle \mathbf{t}, \mathbf{v} \rangle$. To compute a kernel between two text pairs $\mathbf{x}_i$ and $\mathbf{x}_j$ we define the following all-vs-all kernel, where all possible combinations of components, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, from each text pair are considered: $K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(1)}) + K(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(2)}) + K(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}) + K(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(2)})$. Each of the kernel computations K can be broken down into the following: $K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = K_{\text{TK}}(\mathbf{t}^{(1)}, \mathbf{t}^{(2)}) + K_{\text{fvec}}(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, where K_{TK} computes a structural kernel and K_{fvec} is a kernel over feature vectors, e.g., linear, polynomial or RBF, etc. Further in the text we refer to structural tree kernel models as `TK` and explicit feature vector representation as `fvec`.

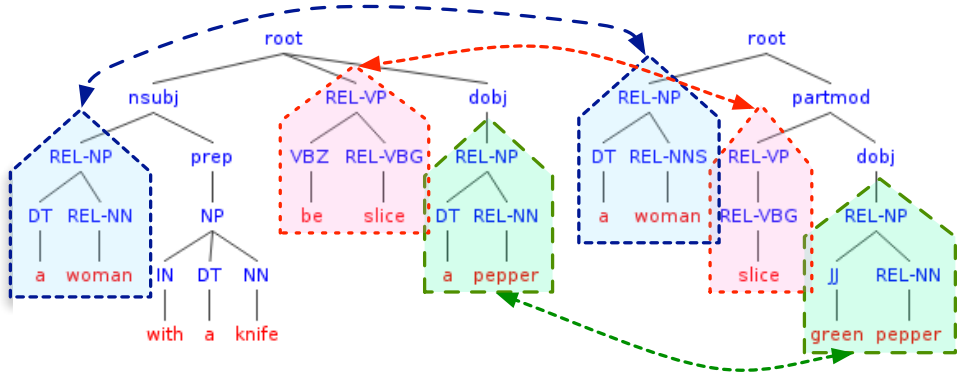


Figure 5.1: A phrase dependency-based structural representation of a text pair (s1, s2): *A woman with a knife is slicing a pepper* (s1) vs. *A women slicing green pepper* (s2) with a high semantic similarity (human judgement score 4.0 out of 5.0). Related tree fragments are linked with a `REL` tag.

Having defined a way to jointly model text pairs using structural `TK` representations along with the similarity features `fvec`, next we present our relational tree structures.

5.2.2 Syntactic models of text pairs

In this section, we define simple-to-build relational structures based on: (i) a shallow syntactic tree, (ii) constituency, (iii) dependency and (iv) phrase-dependency trees.

Shallow tree is a two-level syntactic hierarchy built from word lemmas (leaves), part-of-speech tags (preterminals) that are further organized into chunks. It was shown to significantly outperform feature vector baselines for modeling relationships between question answer pairs [133].

Constituency tree. While shallow syntactic parsing is very fast, here we consider using constituency structures as a potentially richer source of syntactic/semantic information.

Dependency tree. We propose to use dependency relations between words to derive an alternative structural representation. In particular, dependency relations are used to link words in a way that they are always at the leaf level. This reordering of the nodes helps to avoid the situation where nodes with words tend to form long chains. This is essential for PTK to extract meaningful fragments. We also plug part-of-speech tags between the word nodes and nodes carrying their grammatical role.

Phrase-dependency tree. We explore a phrase-dependency tree similar to the one defined in [180]. It represents an alternative structure derived from the dependency tree, where the dependency relations between words belonging to the same phrase (chunk) are collapsed in a unified node. Different from [180], the collapsed nodes are stored as a shallow subtree rooted at the unified node. This node organization is particularly suitable for PTK that effectively runs a sequence kernel on the tree fragments inside each chunk subtree. Fig 5.1 gives an example of our variation of a phrase dependency tree.

As a final consideration, if a document contains multiple sentences they are merged

in a single tree with a common root. To encode the structural relationships between documents in a pair a special `REL` tag is used to link the related structures. We adopt a simple strategy to establish such links: words from two documents that have a common lemma get their parents (POS tags) and grandparents, non-terminals, marked with a `REL` tag.

5.2.3 Pairwise similarity features

Along with the direct representation of input text pairs as structural objects our framework is also capable of encoding pairwise similarity feature vectors (`fvec`), which we describe below.

Baseline features (`base`). We adopt similarity features from two best performing systems of STS-2012, which were publicly released[1]: namely, the Takelab[2] system [172] and the UKP Lab's system[3] [14]. Both systems represent input texts with similarity features combining multiple text similarity measures of varying complexity.

UKP (`U`) provides metrics based on matching of character, word n-grams and common subsequences. It also includes features derived from Explicit Semantic Analysis [47] and aggregation of word similarity based on lexical-semantic resources, e.g., WordNet. In total it provides 18 features.

Takelab (`T`) includes n-gram matching of varying size, weighted word matching, length difference, WordNet similarity and vector space similarity where pairs of input sentences are mapped into Latent Semantic Analysis (LSA) space. The features are computed over several sentence representations where stop words are removed and/or lemmas are used in place of raw tokens. The total number of Takelab's features is 21. The combined system consists of 39 features.

Additional features. We also augment the `U` and `T` feature sets, with an additional set of features (`A`) which includes: a cosine similarity scores computed over (i) n-grams of part-of-speech tags (up to 4-grams), (ii) SuperSense tags [31], (iii) named entities, (iv) dependency triplets, and (v) PTK syntactic similarity scores computed between documents in a pair, where as input representations we use raw *dependency* and *constituency* trees. To alleviate the problem of domain adaptation, where datasets used for training and testing are drawn from different sources, we include additional features to represent the combined text of a pair: (i) bags (`B`) of lemmas, dependency triplets, production rules (from the constituency parse tree) and a normalized length of the entire pair; and (ii) a manually encoded corpus type (`M`), where we use a binary feature with a non-zero entry

[1] Note that only a subset of the features used in the final evaluation was released, which results in lower accuracy when compared to the official rankings.

[2] http://takelab.fer.hr/sts/

[3] https://code.google.com/p/dkpro-similarity-asl/wiki/SemEval2013

Experiment	U	T	A	S	C	D	P	STK	PTK	B	M	ALL	Mean	MSRp	MSRv	SMTe	OnWN	SMTn
`fvec` model	•											.7060	.6087	.6080	.8390	.2540	.6820	.4470
		•										.7589	.6863	.6814	.8637	.4950	.7091	.5395
	•	•										.8079	.7161	.7134	.8837	.5519	.7343	.5607
	•	•	•									.8187	.7137	.7157	.8833	.5131	.7355	.5809
`TK` models with STK and PTK	•	•	•	•				•				.8261	.6982	.7026	.8870	.4807	.7258	.5333
	•	•	•		•			•				.8326	.6970	.7020	.8925	.4826	.7190	.5253
	•	•	•			•		•				.8341	.7024	.7086	.8921	.4671	.7319	.5495
	•	•	•				•	•				.8211	.6693	.6994	.8903	.2980	.7035	.5603
	•	•	•	•					•			.8362	.7026	.6927	.8896	.5282	.7144	.5485
	•	•	•		•				•			.8458	.7047	.6935	.8953	.5080	.7101	.5834
	•	•	•			•			•			.8468	.6954	.6717	.8902	.4652	.7089	.6133
	•	•	•				•		•			.8326	.6693	.7108	.8879	.4922	.7215	.5156
`REL` tag	•	•	•		○							.8218	.6899	.6644	.8726	.4846	.7228	.5684
	•	•	•			○						.8250	.7000	.6806	.8822	.5171	.7145	.5769
domain adaptation	•	•	•		•					•		.8539	.7132	.6993	.9005	.4772	.7189	.6481
	•	•	•			•				•		.8529	.7249	.7080	.8984	.5142	.7263	.6700
	•	•	•		•	•				•		.8546	.7156	.6989	.8979	.4884	.7181	.6609
	•	•	•		•	•					•	.8810	.7416	.7210	.8971	.5912	.7328	.6778
	UKP (best system of STS-2012)											.8239	.6773	.6830	.8739	.5280	.6641	.4937

Table 5.1: Results on STS-2012. First set of experiments studies the combination of `fvec` models from UKP (U), Takelab (T) and (A). Next we show results for four structural representations: shallow (S), constituency (C), dependency (D) and phrase-dependency (P) trees with STK and PTK; next row set demonstrates the necessity of relational linking for two best structures, i.e. C and D (empty circle denotes a structures with no relational linking.); finally, domain adaptation via bags of features (B) of the entire pair and (M) manually encoded dataset type show the state of the art results.

corresponding to a dataset type. This helps the learning algorithm to learn implicitly the individual properties of each dataset.

Stacking. To integrate multiple `TK` representations into a single model we apply a classifier stacking approach [44]. Each of the learned `TK` models is used to generate predictions which are then plugged as features into the final `fvec` representation, s.t. the final model uses only explicit feature vector representation. To obtain prediction scores, we apply 5-fold cross-validation scheme, s.t. for each of the held-out folds we obtain independent predictions.

5.2.4 Experiments

We present the results of our structural learning model for STS when tested on the data from the Core STS task at SemEval 2012 [2].

Experimental setup

Data. To compare with the best systems of the STS-2012 we followed the same setup used in the final evaluation, where 3 datasets (*MSRpar*, *MSRvid* and *SMTeuroparl*) are used for training and 5 for testing (two "surprise" datasets were added: *OnWN* and

SMTnews). We use the entire training data to obtain a single model for making predictions on each test set.

Software. To encode TK models along with the similarity feature vectors into a single regression scoring model, we use an SVR framework implemented in SVM-Light-TK[4]. We use the following parameter settings `-t 5 -F 1 -W A -C +`, which specifies a combination of trees and feature vectors (`-C +`), STK over trees (`-F 1`) (`-F 3` for PTK) computed in all-vs-all mode (`-W A`) and polynomial kernel of degree 3 for the feature vector (active by default).

Metrics. We report the following metrics employed in the final evaluation: *Pearson* correlation for individual test sets[5] and *Mean* – an average score weighted by the test set size.

Results

Table 5.1 summarizes the results of combining TK models with a strong feature vector model. We test structures defined in Sec. 5.2.2 when using STK and PTK. The results show that: (i) combining all three features sets (`U`, `T`, `A`) provides a strong baseline system that we attempt to further improve with our relational structures; (ii) the generality of PTK provides an advantage over STK for learning more versatile models; (iii) constituency and dependency representations seem to perform better than shallow and phrase-dependency trees; (iv) using structures with no relational linking does not work; (v) `TK` models provide a far superior source of structural similarity than `U + T + A` that already includes PTK similarity scores as features, and finally (vi) the domain adaptation problem can be addressed by including corpus specific features, which leads to a large improvement over the previous best system.

5.3 Microblog Retrieval

Similarly to our structural learning model to tackle the STS task, we design a syntax-aware re-ranker for Mircroblog retrieval. It consists of two components: (i) a syntactic model that encodes tweets into shallow linguistic trees to ease feature extraction, and (ii) a tree kernel learning framework that computes similarities between query-tweet pairs. We also define a shallow tree kernel to enable efficient kernel computations.

[4] http://disi.unitn.it/moschitti/Tree-Kernel.htm
[5] we also report the results for a concatenation of all five test sets (`ALL`)

5.3.1 Learning

We employ a pointwise approach (see Sec. 2.6) to re-ranking where a binary classifier is used to learn a model to discriminate between relevant and non-relevant (query, tweet) pairs. The prediction scores from a classifier are then used to re-rank candidates. We use a our tree kernel function **Sh**allow syntactic **T**ree **K**ernel (SHTK) introduced in Sec. 4.2.2, which is as expressive as Partial Tree Kernel (PTK) [93], while it is more efficient to handle shallow tree structures. For feature vectors we use a linear kernel.

We represent each query-tweet pair $\mathbf{x}$ as a triple composed of a query tree $\mathbf{T}_q$ and a tweet tree $\mathbf{T}_{tw}$ together with a traditional feature vector $\mathbf{v}$, i.e., $\mathbf{x} = \langle \mathbf{T}_q, \mathbf{T}_{tw}, \mathbf{v} \rangle$. Given two (query, tweet) pairs $\mathbf{x}^i$ and $\mathbf{x}^j$, we define the following similarity kernel:

$$\begin{aligned} K(\mathbf{x}^i, \mathbf{x}^j) \;=\; & K_{TK}(\mathbf{T}^i_q, \mathbf{T}^j_q) + K_{TK}(\mathbf{T}^i_q, \mathbf{T}^j_{tw}) \\ & + K_{TK}(\mathbf{T}^i_{tw}, \mathbf{T}^j_{tw}) + K_{TK}(\mathbf{T}^j_q, \mathbf{T}^i_{tw}) \\ & + K_{\mathrm{v}}(\mathbf{v}^i, \mathbf{v}^j), \end{aligned} \tag{5.1}$$

where K_{TK} computes a tree kernel similarity between linguistic trees and K_{v} is a kernel over feature vectors. It computes an all-vs-all tree kernel similarity between two query-tweet pairs.

5.3.2 A syntactic model for tweets

Our approach to extract features from (query, tweet) pairs goes beyond traditional feature vectors. We employ structural syntactic models (`STRUCT`) that encode each tweet into shallow syntactic trees. The latter are input to tree kernel functions for generating structural features. Our structures are specifically adapted to the noisy tweets and encode important query/tweet relations.

In particular, our shallow tree structure (inspired by [133, 136, 138]) is a two-level syntactic hierarchy built from word lemmas (leaves) and part-of-speech tags that are grouped into chunks (Fig. 5.2). While full syntactic parsers would significantly degrade in performance on noisy texts such as tweets, our choice for shallow structure relies on simpler and more robust components: a part-of-speech (POS) tagger and a chunker. For POS tagging we use the CMU tagger [51] trained on Twitter data and an off-the-shelf OpenNLP chunker.

Fig. 5.2 provides an example of a candidate (query, tweet) pair each of which is encoded into a shallow linguistic structure. To upweight the tree fragments spanning words that are found in both the query and the tweet we introduce a special `REL` tag at the level of part-of-speech and chunk nodes. This step is important to generate syntactic patterns that carry additional semantics of sharing common terms between a query and a tweet. To find matching word pairs we lowercase and stem words and use plain string matching.

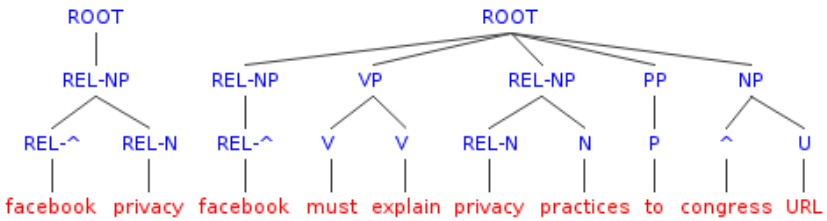


Figure 5.2: Shallow tree representation for an example query-tweet pair: (*"Facebook privacy", "Facebook Must Explain Privacy Practices to Congress http://sns.ly/2Qbry7"*) (in the original tree we use word stems). Part-of-speech tag ^ refers to a common noun. Note that an additional `REL` tag links the words (stems) common between the query and the candidate tweet, *Facebook* and *privacy*).

5.3.3 Experiments

To evaluate the utility of our structural syntactic re-ranker for microblog search we focus on the 2011 and 2012 editions of the ad-hoc retrieval task at TREC microblog tracks [103, 149]. Our main research question is: Does the use of relational syntactic features produced by our shallow syntactic parser, and the automatic feature engineering approach based on structural kernels lead to improvements in state-of-the-art L2R and retrieval algorithms?

To answer this question, we test our model in two settings. In the first, we re-implement an accurate recent L2R-based approach and add our features alongside its features. This will allow us to see directly if our features are complementary to the other features. We opted for the L2R approach in [16] ("the UvA model"), because of its comprehensiveness. It uses pseudo-test collections [12] to learn to fuse ten well-established retrieval algorithms and implements a number of query, tweet, and query-tweet features. It is a strong baseline, its performance ranks sixth and 26th in the 2011 and 2012 editions of the microblog track, respectively. In the second setting, we use the participant systems in the TREC microblog task as a black-box, and implement our model on top of them using only using their raw scores (ranks) as a single feature in our model. This allows us to see whether our features add information to the approaches these retrieval algorithms use.

Experimental setup

Dataset. Our dataset is the tweet corpus used in the TREC Microblog track in both 2011 (TMB2011) and 2012 (TMB2012). It consists of 16M tweets spread over two weeks, and a set of 49 (TMB2011) and 60 (TMB2012) timestamped topics. We minimally pre-process the tweets—we normalize elongations (e.g., sooo → so) and normalize URLs and author ids. For the second set of experiments, we also use the system runs submitted at TMB2011 and TMB2012, which contain 184 and 120 models, respectively.

Training and testing an L2R algorithm. For learning to rank we use SVM-light-TK[6] with no parameter tuning. In our first set of experiments, we train on TMB2011 topics, test on TMB2012 topics, and vice versa. In the second set, where we build upon the TREC participant runs, we train our system only on the runs submitted at TMB2011, and test on the TMB2012 runs. We focus on one direction only to avoid training bias, since TMB2011 topics were already used for learning systems in TMB2012.

Feature normalization. When combining our features with those of the UvA model, while training and testing we use the features of the latter model as $\mathbf{v}$ in Eq. 6.3; these features are already normalized. In contrast, we use the output of participant systems as follows. We use rank positions of each tweet rather than raw scores, since scores for each system are scaled differently, while ranks are uniform across systems. We apply the following transformation of the rank r: $1/\log(r+1)$. In the training phase, we take the top {10, 20, or 30} systems from the TMB2011 track (in terms of P@30). For each (query, tweet) pair we average the transformed rank over the top systems to yield a single score. This score is then used as a single feature in $\mathbf{v}$ from Eq. 6.3. In the testing phase, for each participant system we want to improve, we use the transformed rank of the (query, tweet) pairs as the single feature in $\mathbf{v}$.

Evaluation. We report on the official evaluation metric for the TREC 2012 Microblog track, i.e., precision at 30 (P@30), and also on mean average precision (MAP). Following [16, 103], we regard minimally and highly relevant documents as relevant and use the TMB2012 evaluation script. For significance testing, we use a pairwise t-test, where $\triangle$ and $\blacktriangle$ denote significance at $\alpha = 0.05$ and $\alpha = 0.01$, respectively. Triangles point up for improvement over the baseline, and down otherwise. We also report the improvement in the absolute rank (R) in the official TMB2012 ranking.

Results

Table 5.2 lists the outcome of our first set of experiments, where we use our syntactic features alongside the features of the UvA model. It shows the obtained MAP and P@30 scores when we train on TMB2011 and test on TMB2012 topics, and vice versa. The STRUCT model yields a significant improvement in P@30 and MAP scores on TMB2012 pushing up the system by 15 positions in the official ranking, and making it second best in TMB2011. The result support our claim that learning useful syntactic patterns from noisy tweets is possible and that relational syntactic features generated by our shallow syntactic tree kernel improve over a strong feature-based L2R baseline.

Table 7.9 reports on the application of our syntax-aware re-ranker on participant systems. It has results for re-ranking runs of the best 30 systems from TMB2012 (based

[6] http://disi.unitn.it/moschitti/Tree-Kernel.htm

Table 5.2: System performance (P@30, MAP; higher is better) and system rank (R; lower is better) for UvA's L2R system [16] (UvA), our re-implementation (UvA*), and a UVA* system using our STRUCT model (+STRUCT). We report on relative improvement (Impr) and statisical significance against UvA*.

Model	TMB2011			TMB2012		
	MAP	P@30	R	MAP	P@30	R
UVA	.3880	.4460	6	.2450	.3920	26
UVA*	.3845	.4456	6	.2467	.3870	28
+ STRUCT	**.3991**	**.4571**	**2**	**.2683**	**.4277**	**13**
Change	+3.8%$^{\triangle}$	+2.6%	+4	+8.8%$^{\blacktriangle}$	+10.5%$^{\blacktriangle}$	+15

on their P@30 score) when we train our system using the top {10, 20, or 30} runs from TMB2011. Our re-ranker improves P@30 for all systems with a relative improvement ranging from several points up to 10%—about 5% on average. This is remarkable, given that the pool of participants in TMB2012 was large, and the top systems are therefore likely to be very strong baselines. We observe that our syntactic model has a precision-enhancing effect. In cases where MAP drops a bit it can be seen that our model sometimes lowers relevant documents in the runs. It is possible that our model favors tweets with a higher syntactic quality, and that it down-ranks tweets that contain less syntactic structure but are nonetheless relevant. This is an interesting direction for analysis in future work.

Looking at the improvement in absolute position in the official ranking (R), we see that, on average, using our re-ranker boosts the absolute position in the official ranking for top 30 systems by 7 positions.

All in all, the results suggest that using syntactic features adds useful information to many state-of-the-art microblog search algorithms.

Finally, using aggregate scores from the best 10, 20 or 30 systems from TMB2011 does not reveal large differences, which suggests that our syntax-aware re-ranker is robust w.r.t. the exact retrieval models used in the training stage.

While improving the top systems from 2012 represents a challenging task, it is also interesting to assess the potential improvement for systems that ranked lower. For this purpose, we select 30 systems from the middle and the bottom of the official ranking. Table 7.10 summarizes the average improvement in P@30 for three groups of 30 systems each: top-30, middle-30, and bottom-30. We find that the improvement over underperforming systems is much larger than for stronger systems. In particular, for the bottom 30 systems, our approach achieves an average relative improvement of 20% in both MAP and P@30. These results further support our hypothesis that syntactic patterns automatically extracted and learned by our re-ranker can provide an additional benefit for learning to rank methods on microblog data.

Table 5.3: System performance on the top 30 runs from TMB2012, using the top 10, 20 or 30 runs from TMB2011 for training.

	TMB2012		TOP10			TOP20			TOP30		
# runs	MAP	P@30	MAP	P@30	R%	MAP	P@30	R%	MAP	P@30	R%
1 hitURLrun3	.3469	.4695	.3307 (-4.7%)▽	.4831 (2.9%)	0	.3378 (-2.6%)	.4864 (3.6%)△	0	.3328 (-4.1%)▽	.4774 (1.7%)	0
2 kobeMHC2	.3070	.4689	.3029 (-1.3%)	.4740 (1.1%)	1	.3065 (-0.2%)	.4768 (1.7%)	1	.3037 (-1.1%)	.4768 (1.7%)	1
3 kobeMHC	.2986	.4616	.2956 (-1.0%)	.4706 (2.0%)	2	.2989 (0.1%)	.4734 (2.6%)	2	.2965 (-0.7%)	.4718 (2.2%)	2
4 uwatgclrman	.2836	.4571	.3010 (6.1%)▲	.4729 (3.5%)△	3	.3032 (6.9%)▲	.4729 (3.5%)▲	3	.2995 (5.6%)▲	.4712 (3.1%)△	3
5 kobeL2R	.2767	.4429	.2734 (-1.2%)	.4452 (0.5%)	0	.2785 (0.7%)	.4514 (1.9%)	0	.2744 (-0.8%)	.4463 (0.8%)	0
6 hitQryFBrun4	.3186	.4424	.3102 (-2.6%)	.4554 (2.9%)	1	.3145 (-1.3%)	.4582 (3.6%)△	2	.3118 (-2.1%)	.4554 (2.9%)	2
7 hitLRrun1	.3355	.4379	.3200 (-4.6%)▽	.4508 (3.0%)	2	.3266 (-2.7%)	.4542 (3.7%)△	2	.3226 (-3.9%)▽	.4525 (3.3%)	2
8 FASILKOM01	.2682	.4367	.2827 (5.4%)▲	.4548 (4.1%)▲	3	.2841 (5.9%)▲	.4525 (3.6%)▲	3	.2820 (5.2%)▲	.4531 (3.8%)▲	3
9 hitDELMrun2	.3197	.4345	.3090 (-3.4%)▽	.4446 (2.3%)	4	.3142 (-1.7%)	.4458 (2.6%)	4	.3105 (-2.9%)	.4424 (1.8%)	4
10 tsqe	.2843	.4339	.2832 (-0.4%)	.4435 (2.2%)	5	.2865 (0.8%)	.4458 (2.7%)	5	.2836 (-0.3%)	.4441 (2.4%)	5
11 ICTWDSERUN1	.2715	.4299	.2873 (5.8%)▲	.4610 (7.2%)▲	7	.2885 (6.3%)▲	.4576 (6.4%)▲	7	.2862 (5.4%)▲	.4582 (6.6%)▲	7
12 ICTWDSERUN2	.2671	.4266	.2809 (5.2%)△	.4503 (5.6%)▲	7	.2808 (5.1%)△	.4508 (5.7%)▲	7	.2785 (4.3%)△	.4475 (4.9%)▲	7
13 cmuPrfPhrE	.3179	.4254	.3159 (-0.6%)	.4486 (5.5%)▲	8	.3190 (0.4%)	.4452 (4.7%)▲	8	.3172 (-0.2%)	.4469 (5.1%)▲	8
14 cmuPrfPhrENo	.3198	.4249	.3167 (-1.0%)	.4497 (5.8%)▲	9	.3201 (0.1%)	.4480 (5.4%)▲	9	.3179 (-0.6%)	.4486 (5.6%)▲	9
15 cmuPrfPhr	.3167	.4198	.3117 (-1.6%)	.4441 (5.8%)▲	10	.3154 (-0.4%)	.4407 (5.0%)▲	8	.3130 (-1.2%)	.4379 (4.3%)△	8
16 FASILKOM02	.2454	.4141	.2725 (11.0%)▲	.4497 (8.6%)▲	11	.2721 (10.9%)▲	.4497 (8.6%)▲	11	.2718 (10.8%)▲	.4508 (8.9%)▲	11
17 IBMLTR	.2630	.4136	.2734 (4.0%)△	.4424 (7.0%)▲	10	.2758 (4.9%)▲	.4412 (6.7%)▲	10	.2734 (4.0%)△	.4441 (7.4%)▲	10
18 otM12ihe	.2995	.4124	.2968 (-0.9%)	.4333 (5.1%)▲	7	.3015 (0.7%)	.4339 (5.2%)▲	7	.2969 (-0.9%)	.4322 (4.8%)▲	7
19 FASILKOM03	.2716	.4124	.2861 (5.3%)▲	.4407 (6.9%)▲	12	.2879 (6.0%)▲	.4469 (8.4%)▲	14	.2859 (5.3%)▲	.4452 (8.0%)▲	14
20 FASILKOM04	.2461	.4113	.2584 (5.0%)▲	.4362 (6.1%)▲	11	.2596 (5.5%)▲	.4322 (5.1%)▲	9	.2575 (4.6%)▲	.4294 (4.4%)▲	9
21 IBMLTRFuture	.2731	.4090	.2803 (2.6%)	.4384 (7.2%)▲	14	.2830 (3.6%)△	.4328 (5.8%)▲	10	.2808 (2.8%)	.4311 (5.4%)▲	10
22 uiucGSLIS01	.2445	.4073	.2574 (5.3%)△	.4271 (4.9%)△	10	.2612 (6.8%)▲	.4260 (4.6%)▲	9	.2575 (5.3%)▲	.4260 (4.6%)▲	9
23 PKUICST4	.2786	.4062	.2913 (4.6%)△	.4537 (11.7%)▲	18	.2931 (5.2%)▲	.4486 (10.4%)▲	18	.2909 (4.4%)△	.4514 (11.1%)▲	18
24 uogTrLsE	.2909	.4028	.2983 (2.5%)	.4282 (6.3%)▲	12	.3015 (3.6%)△	.4243 (5.3%)▲	9	.2977 (2.3%)	.4282 (6.3%)▲	9
25 otM12ih	.2777	.3989	.2807 (1.1%)	.4260 (6.8%)▲	12	.2839 (2.2%)	.4232 (6.1%)▲	10	.2810 (1.2%)	.4175 (4.7%)▲	10
26 ICTWDSERUN4	.1877	.3887	.1995 (6.3%)▲	.4136 (6.4%)▲	8	.1992 (6.1%)▲	.4164 (7.1%)▲	10	.1985 (5.8%)▲	.4164 (7.1%)▲	10
27 uwatrrfall	.2620	.3881	.2829 (8.0%)▲	.4158 (7.1%)▲	11	.2841 (8.4%)▲	.4136 (6.6%)▲	9	.2812 (7.3%)▲	.4136 (6.6%)▲	9
28 cmuPhrE	.2731	.3842	.2792 (2.2%)	.4130 (7.5%)▲	10	.2810 (2.9%)	.4164 (8.4%)▲	12	.2797 (2.4%)	.4136 (7.7%)▲	12
29 AIrun1	.2237	.3842	.2350 (5.1%)▲	.4085 (6.3%)▲	7	.2359 (5.5%)▲	.4056 (5.6%)▲	5	.2339 (4.6%)▲	.4102 (6.8%)▲	5
30 PKUICST3	.2118	.3825	.2320 (9.5%)▲	.4220 (10.3%)▲	15	.2324 (9.7%)▲	.4181 (9.3%)▲	14	.2318 (9.4%)▲	.4119 (7.7%)▲	14
Average			2.4%	5.4%	7.7	3.3%	5.3%	7.3	2.4%	5.0%	7.1

5.4 Summary

We have presented an approach where text pairs are directly treated as structural objects. This provides a much richer representation for the learning algorithm to extract useful syntactic and shallow semantic patterns.

We have provided an extensive experimental study of four different structural representations for modelling the STS problem, e.g. shallow, constituency, dependency and phrase-dependency trees using STK and PTK. The novelty of our approach is that it goes beyond a simple combination of tree kernels with feature vectors as: (i) it directly encodes input text pairs into relationally linked structures; (ii) the learned structural models are used to obtain prediction scores thus making it easy to plug into existing feature-based models, e.g. via stacking; (iii) to our knowledge, this work is the first to apply structural kernels and combinations in a regression setting; and (iv) our model achieves the state of the art in STS largely improving the best previous systems. Our structural learning approach to STS is conceptually simple and does not require additional linguistic sources other than off-the-shelf syntactic parsers. It is particularly suitable for NLP tasks where the input domain comes as pairs of objects, e.g., question answering, paraphrasing and

Table 5.4: System performance for top, middle (mid), and bottom (btm) 30 systems from TMB2012 system ranking and relative improvements using our method trained on top 20 (TOP20) performing systems in TMB2011.

	TMB2012		TOP20	
band	MAP	P@30	MAP	P@30
top	.2794	.4209	.2876 (3.3%)	.4430 (5.3%)
mid	.2193	.3460	.2461 (12.2%)	.3906 (12.9%)
btm	.1332	.2636	.1626 (22.1%)	.3298 (25.1%)

recognising textual entailment.

Regarding our model for reranking tweets, to the best of our knowledge, this work is the first to study the utility of syntactic patterns for microblog retrieval. We propose an efficient way to encode tweets into linguistic structures and use kernels for automatic feature engineering and learning. Our experimental findings show that our model: (i) improves in both MAP and P@30 when coupled with the features from a strong L2R baseline; (ii) provides a complementary source of features general enough to improve the best 30 systems from TMB2012; (iii) the performance gains are stable when we use run scores from the top 10, 20 or 30 best systems for learning; and (iv) the improvement becomes larger for under-performing systems achieving an average 20% of relative improvement in MAP and P@30 for bottom 30 systems.

Chapter 6

Modelling Question-Answer Pairs

In this chapter we focus on the design of relational syntactic tree structures to model input question-answer pairs for Question Answer (QA) reranking and answer extraction.

First, we show that learning to rank models can be applied to automatically learn complex patterns, such as relational semantic structures occurring in questions and candidate answers represented either by their supporting passages or single sentences. This is achieved by providing the learning algorithm with a tree representation derived from the syntactic trees of questions and answer candidates connected by relational tags, where the latter are again provided by the means of automatic classifiers, i.e., question and focus classifiers and Named Entity Recognizers. This way effective structural relational patterns are implicitly encoded in the representation and can be automatically utilized by our tree kernel learning framework.

Secondly, we show that our syntactic tree models are very effective in producing highly discriminative features for a different yet closely related task of answer extraction.

In this chapter we consider the following three QA tasks: reranking of candidate answer passages, answer sentence selection and answer extraction. The difference between the first and the second task is the granularity of the candidate answers where in the first case we deal with entire passages while in the latter case answer candidates are limited to a single sentence. This has a direct implication on the size of the resulting structures representing answer candidates, thus requiring to apply a pruning strategy in the case when we deal with answer passages. The goal of the answer extraction task is to identify a word or a phrase that is the answer keyphrase sought by a given factoid question. While it might be natural to treat this task a sequence labelling problem (Sec. 2.2.3), we opt for a simpler approach treating it as a classification problem, such that we can directly benefit from applying our tree kernel technology.

Although we address two different problems: reranking and answer extraction, the core focus and, hence, the contribution of this chapter is on how to design effective syntactic structures to represent question-answer pairs and encode relational information in a pair.

6.1 Overview

Question Answering (QA) systems are typically built from three main macro-modules: (i) search and retrieval of candidate passages; (ii) reranking or selection of the most promising passages; and (iii) answer extraction. The last two steps are the most interesting from a Natural Language Processing viewpoint since deep linguistic analysis can be carried out as the input is just a limited set of candidates.

Answer passage reranking (or sentence selection) refers to the task of selecting the answer candidate containing the correct answer among the different answer candidates retrieved by a search engine.

Answer extraction is a final step, required for factoid questions, consisting in extracting multiwords constituting the synthetic answer, e.g., *Barack Obama* for a question: *Who is the US president?*

Automated Question Answering (QA) is a complex task that often requires manual definition of rules and syntactic patterns to detect the relations between a question and its candidate answers in text fragments. Simple heuristics just refer to computing a textual similarity between the question and one of its answer candidates but the most accurate method is to manually design specific rules that are triggered when patterns in the question and in the passage are found. Such rules are based on syntactic and semantic patterns. For example, given a question[1]:

> *What is Mark Twain's real name?*

and a relevant passage, e.g., retrieved by a search engine:

> *Samuel Langhorne Clemens, better known as Mark Twain.*

the QA engineer usually applies a syntactic parser to obtain the parse trees of the above two sentences, e.g., like those in the top of Fig. 6.1 to derive rules like:

> **if** *the pattern* "`What is` NP_2's ADJ `name`" *is in the question* **and** *the pattern* "NP_1 `better known as` NP_2" *is in the answer passage* **then** *associate the passage with a high score*[2],

where the NPs are noun phrases and ADJ is an adjectival phrase recognized by the syntactic parser.

Previous work, e.g., carried out in TREC[3] [169–171], has shown that such an approach can lead to the design of accurate systems. However, it suffers from two major drawbacks: (i) being based on heuristics it does not provide a definitive methodology, since natural language is too complex to be characterized by a finite set of rules; and (ii) given the latter

[1] We use this question-answer pair from TREC QA as a running example in the rest of this chapter.
[2] If the point-wise answer is needed rather than the entire passage, the rule could end with: **returns** NP_1
[3] http://trec.nist.gov/data/qamain.html

claim, new domains and languages require the definition of new specific rules, which in turn require a large engineering effort.

An alternative to manual rule definition is the use of machine learning, which often shifts the problem to the easier task of feature engineering. This is very convenient for simple text categorization problems, such as document topic classification, where simple *bag-of-words* models have been shown very effective, e.g., [64]. Unfortunately, when the learning task is more complex such as in QA, features have to encode the combination of syntactic and semantic properties, which basically assume the shape of high-level rules: these are essential to achieve state-of-the-art accuracy. For example, the famous IBM Watson system [45] also uses a learning to rank algorithm fed with hundreds of features. For the extraction of some of the latter, articulated rules are required, which are very similar to those constituting typical manually engineered QA systems.

In this chapter, we show that learning to rank models can be applied to automatically learn structural patterns. These are relational structures occurring in question and answer passages and are based on the semantic information automatically derived by additional automatic classification modules. In particular, we first derive a representation of the question and answer passage (q/a) pair, where we follow our approach in [133] by engineering a pair of shallow syntactic trees connected with relational nodes (i.e., those matching the same words in the question and in the answer passages).

Secondly, we include a large sample of basic and advanced features that represent a strong baseline model for answer passage reranking. Additionally, we explore various methods in addressing the "lexical gap" problem: words in a question and its candidate answer can be semantically similar but having different surface forms. To establish relational links between a question and an answer, we exploit WordNet hierarchy [88], LDA topic models [27, 63], SuperSense tagging [31] and word alignments from translation models, e.g., [125, 152]. Their failure in improving our structural representation provides a strong indication that word generalization is not sufficient to improve on strong statistical-based retrieval systems, and more principled linking strategy is required.

To enable a more principled linking strategy, we model and implement question and focus classifiers based on kernel methods. Then, we use the output of such classifiers together with a named entity recognizer (NER) to establish relational links [137]. The focus classifier determines the constituent of the question to be linked to the named entities (NEs) of the answer passage. The target NEs are selected based on the compatibility of their category and the category of the question, e.g., an NE of type PERSON is compatible with a category of a question asking for a HUMAN.

In summary, to build our relational structures we (i) design a pair of shallow syntactic trees (one for the question and one for the answer candidate); (ii) connect them with relational nodes (i.e., those matching the same words in the question and in the answer passages); (iii) label the tree nodes with semantic information such as question category

and focus and NEs; and (iv) use the NE type to establish additional semantic links between the candidate answer, i.e., an NE, and the focus word of the question. Finally, for the task of answer extraction we also connect such semantic information to the answer sentence trees such that we can learn factoid answer patterns.

We show that highly discriminative features can be, in fact, automatically extracted and learned by using our kernel learning framework. Our approach does not require manual feature engineering to represent input structures. We do not fully rely on traditional similarity features that encode the degree of similarity between a question and its answer. We treat the input q/a pairs directly encoding them into linguistic trees. More powerful features can be encoded by injecting additional semantic information directly into the tree via special tags and additional tree nodes. We believe such way of engineering features is more intuitive and requires less effort, since the final patterns are automatically extracted by expressive tree kernels.

In the remainder of this chapter, Section 6.2 describes our syntactic tree structures to model question-answer pairs along with the classifiers used to generate semantic information. Section 6.3 describes three QA tasks that we tackle in this chapter. Section 6.4 reports on the baseline and more advanced feature sets. Section 6.5 describes our experimental setup and reports results and our findings on three tasks: (i) answer passage reranking (Section 6.5.1), (ii) answer sentence selection (Section 6.5.2), and answer extraction (Section 6.5.3). Section 6.6 reports on the related work; and finally, Section 6.7 summarizes our findings.

6.2 Structural models of question-answer pairs

In this section we present our structural models aimed at capturing structural similarities between a question and an answer. We use tree structures as our base representation since they provide sufficient flexibility in representation and allow for easier feature extraction than, for example, graph structures. In addition, coupled with the tree kernel learning framework, trees allow for efficient automatic feature engineering.

6.2.1 Basic structural representations

We first describe our shallow models that we explored in [133]. Next, we propose models to bridge the lexical gap between the words in the question and in the answer using various sources of semantic annotation.

Shallow tree structures

Our baseline structural model is the shallow tree representation we first proposed in [133]. This is essentially a shallow syntactic structure built from part-of-speech tags grouped

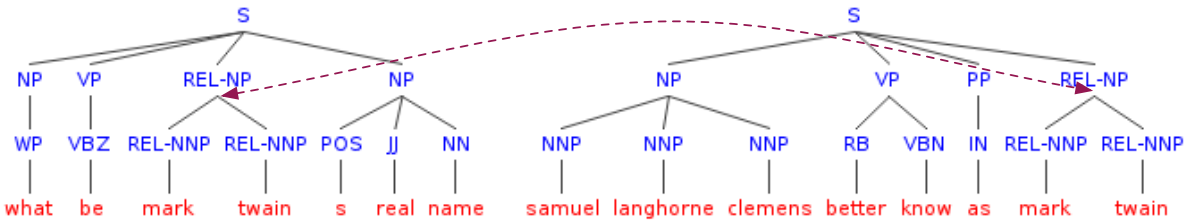


Figure 6.1: Basic structural representations using a shallow chunk tree for the q/a pair. *Q: What is Mark Twain's real name? A: Samuel Langhorne Clemens, better known as Mark Twain.* Arrows indicate the tree fragments in the question and its answer passage linked by the relational `REL` tag.

into chunks. Each question and its candidate answer passage are encoded into a tree where part-of-speech tags are found at the pre-terminal level and word lemmas at the leaf level. The sequences of part-of-speech (POS) tags are further organized into chunks. To encode the structural relationship for a given q/a pair, a special `REL` tag links the related structures. We adopt a simple strategy to determine such links: lexicals from a question and an answer that have a common lemma get their parents (POS tags) and grandparents, i.e., chunk labels, marked by prepending a `REL` tag. An example of a q/a pair encoded using shallow chunk trees is given in Figure 6.1 (top). Our empirical evaluation in [133] showed that such representation is superior to more simple bag-of-words and sequences of POS tags models.

Soft matching for relational linking

As shown in [133], the use of a special tag to mark the related fragments in the question and answer tree representations is the key to learn more accurate relational models. However, we just used a naïve hard matching between the word lemmas. To alleviate the word sparsity problem, where semantically similar words have non-matching surface forms, we explore WordNet, topic models, SuperSense tagging and word alignments from statistical machine translation.

Wordnet. To establish the matching between the structures in the question and a candidate answer we experiment with using synonym groups provided by WordNet. We also experiment with a similarity metric computed between words w_1 and w_2 using the WordNet concept hierarchy that defines hyponym/hypernym relations between the words:

$$sim_{WN}(w_1, w_2) = \frac{min(d(w_1, CP), d(w_2, CP))}{d(CP, R) + min(d(w_1, CP), d(w_2, CP))} \tag{6.1}$$

where d defines the distance in the hierarchy between two concepts, CP denotes a common parent of two words and R is the root of the WordNet hierarchy. We consider a match between two words if their sim_{WN} is below a specified threshold value. We tried various thresholds and found 0.2 to serve as a meaningful boundary. We did not use any sense

Table 6.1: Top 5 words from 5 randomly picked LDA topics extracted from the Aquaint corpus.

topic1	topic9	topic24	topic51	topic77
bank	music	china	family	baseball
financial	rock	chinese	home	game
government	band	beijing	people	yankees
economic	album	xinhua	day	team
banks	songs	province	years	league

disambiguation algorithms and opted for a simple strategy to take the most frequent sense (this seems the most effective approach, according to previous studies, e.g., [155]).

LDA. It has become a popular tool in discovering deeper relationships between q/a pairs, e.g., [27, 63]. It comes from a family of generative probabilistic models, where each document d in the collection D is viewed as a mixture of a fixed number of topics $z \in \mathcal{Z}$. Each topic z represents a distribution over the unique words $w \in V$ in the vocabulary V. More specifically, given a training corpus, an LDA model infers two distributions: $\phi_z^{(w)}$, the probability of a word w being assigned to a topic z, and $\theta^{(d)}$ which defines a distribution of topics for a document d.

Different from previous applications of LDA, where it is primarily used to compute various similarity scores for a given q/a pair, we consider using the obtained vector of topic assignments to each word in the document as a way to generalize hard matching on lemmas. Table 6.1 gives an example of several topics learnt from a large Aquaint corpus from TREC QA. It exemplifies that topics can be used to cluster words into semantically coherent groups. Hence, we explore them to link related fragments in a q/a pair.

SuperSense matching. We explore an alternative approach to link question and answer words that belong to the same semantic category. For this purpose we use a SuperSense tagger[4] [31] that annotates words in a text with the tagset of 41 Wordnet supersense classes for nouns and verbs, e.g., *act, event, relation, change, person, motion*, etc. Words in a question and answer that have the same tag are used to link the related structures.

Word alignment. Recently, the utility of translation models to alleviate the lexical gap between questions and answers has been explored by many QA systems, e.g., [125, 152, 156]. The typical approach is to learn question-to-answer and answer-to-question transformations using a translation model. The translation model is finally used to compute the similarity score relating a given q/a pair, which is then integrated as a similarity feature into the learning to rank model [156]. Different from the previous approaches, we explore

[4] http://sourceforge.net/projects/supersensetag/

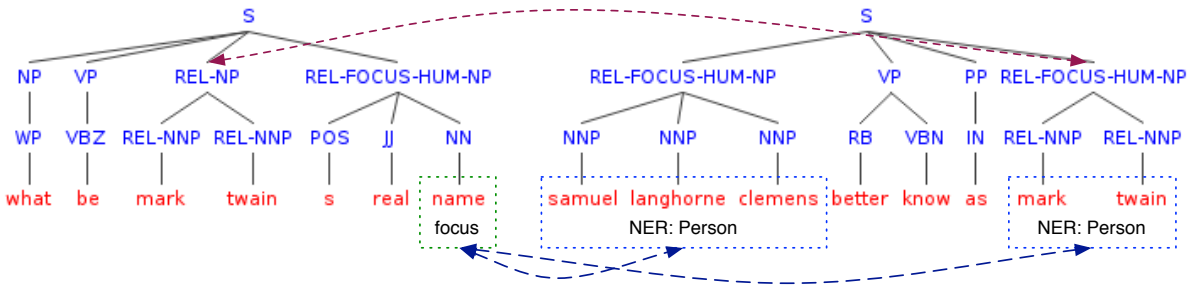


Figure 6.2: Using a typed relation tag `REL-FOCUS-HUM` to link a question focus word *name* with the named entities, whose type corresponds to the question category (HUM).

the utility of a translation model to align words in a given q/a pair for relational linking. To obtain the alignments, we use METEOR[5] monolingual word aligner [39] that includes flexible word and phrase matching using exact, synonym and paraphrase matches. Similar to the string matching strategy, we mark the structures spanning the aligned words with a relational tag.

6.2.2 Refining relational tag using question focus and question category

In this section we briefly describe our alternative strategy to establish relational links first proposed in [137]. We use question category to link the focus word of a question with the named entities extracted from the candidate answer. For this purpose, we briefly introduce our models for building a question focus detector and question category classifier.

Question focus detection

The question focus is typically a simple noun representing the entity or property being sought by the question [117]. It can be used to search for semantically compatible candidate answers in document passages, thus greatly reducing the search space [114]. While several machine learning approaches based on manual features and syntactic structures have been recently explored, e.g., [21, 35, 118], we opt for the latter and rely on the power of tree kernels to handle automatic feature engineering.

Question classification

Question classification is the task of assigning a question to one of the pre-specified categories. We use the coarse-grain classes described in [83]: six non overlapping classes: Abbreviations (ABBR), Descriptions (DESC, e.g., definitions or explanations), Entity (ENTY, e.g., animal, body or color), Human (HUM, e.g., group or individual), Location (LOC, e.g., cities or countries) and Numeric (NUM, e.g., amounts or dates). These categories can be used to determine the Expected Answer Type for a given question and find

[5] http://www.cs.cmu.edu/~alavie/METEOR

Table 6.2: Mapping from question classes to named entity types.

Question Category	Named Entity types
HUM	Person
LOC	Location
NUM	Date, Time, Money, Percentage
ENTY	Organization, Person

the appropriate entities found in the candidate answers. Imposing such constraints on the potential answer keys greatly reduces the search space.

Semantic linking

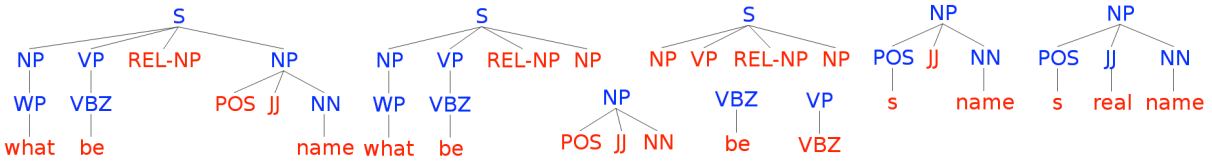
Question focus captures the target information need posed by the question but, to make this piece of information effective, the focus words need to be linked to the target candidate answer. They can be lexically matched with words present in an answer, or the match can be established using semantic information. One method exploiting the latter approach involves question classification.

Once the question focus and class are determined, we propose to link the focus word w_{focus} in the question, with all the named entities whose type matches the question class. Table 6.2 provides the correspondence between the question classes and named entity types. We perform tagging at the chunk level and use two types of the relational tag: plain `REL-FOCUS` and a tag typed with the question class, e.g., `REL-FOCUS-HUM`. Fig. 6.2 shows an example q/a pair where the typed relational tag is used to link the chunk containing the question focus word *name* with the named entities of the corresponding type *Person* (according to the mapping defined in Table 6.2).

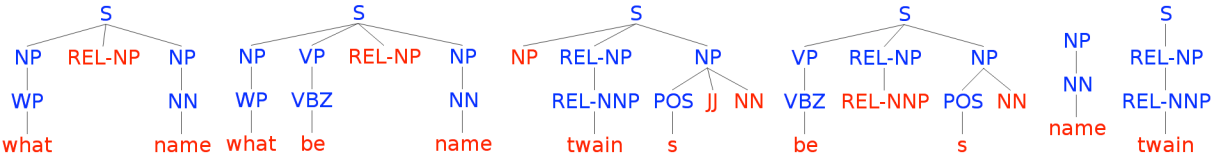
6.2.3 Tree kernels for automatic feature engineering

As pointed out in the introduction, engineering rules and features is the major bottleneck in the design of a QA system. To benefit from the automatic feature engineering offered by tree kernel learning framework also explored in the previous sections we again rely on tree kernels. A kernel function computes an implicit scalar product between input examples, typically in high dimensional spaces. In our case, they measure similarity between structural objects, e.g., parse trees, in terms of the number of common substructures. Different kernels map objects in different spaces. We make use of two types of tree kernels applied to several types of syntactic/semantic structures: STK [32] and PTK [93].

For example, given the parse tree of the question in Fig. 6.1, some of its syntactic fragments generated by tree kernels are shown in Fig. 6.8. It can be noted that each of



(a) Some of the fragments generated by STK



(b) Additional fragments only generated by PTK

Figure 6.3: Some tree fragments generated by STK and PTK tree kernels when applied to the question tree of Fig. 6.1

such fragment can be an important pattern indicating the type of question. Considering STK fragments (see Fig. 6.8 (a)), the first fragment encodes the first part of the manually designed rule described in the introduction, where `REL` is a relation tag. On the other hand, Fig. 6.8.(b) illustrates some fragments only generated by PTK: such patterns are more general and compact than those provided by STK.

6.3 QA tasks

This section describes our approach to the following problems in factoid question answering: (i) answer passage reranking where answer candidates are entire passages (we treat it as a *pairwise* reranking problem); (ii) answer sentence selection where answer candidates are limited to a single sentence (treated as a *pointwise* reranking problem); and (iii) answer extraction whose goal is to identify the answer keyword (treated as a classification problem).

6.3.1 Answer passage reranking

In the following we describe our approach to reranking answer passages (typically retrieved by a search engine) which we treat as a *pairwise* reranking problem. In the following we briefly describe the architecture of our QA pipeline and outline a kernelized version of the *pairwise* reranking approach introduced in Sec. 2.6.

QA architecture

Our QA system for answer passage reranking is based on a rather simple architecture displayed in Figure 6.4: given a question q, a search engine retrieves a list of candidate passages ranked by their relevancy. Next, the question together with its candidate answers are processed by a rich natural language processing pipeline that performs basic

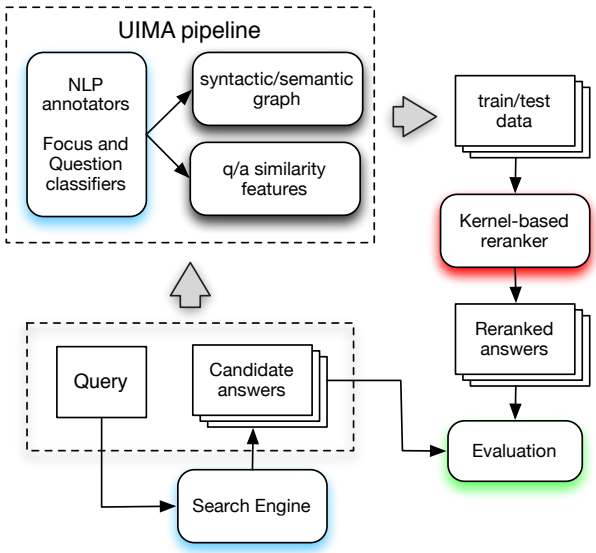


Figure 6.4: Kernel-based answer passage reranking system

tokenization, sentence splitting, lemmatization, stopword removal. Various NLP components embedded in the pipeline as UIMA[6] annotators perform more involved linguistic analysis, e.g., part-of-speech tagging, chunking, named entity recognition, constituency and dependency parsing, etc. These annotations are then used to produce structural models (described in Sec. 6.2), which are further used by a question focus detector and question type classifiers to establish relational links for a given q/a pair. The resulting tree pairs are then used to train a kernel-based reranker, which outputs the model to refine the initial ordering of the retrieved answer passages.

Pairwise reranking with kernels

To enable the use of kernels for learning to rank with SVMs, we use *pairwise* reranking approach (see Sec. 2.6), which can be reduced to the task of binary classification [68]. More specifically, the problem of learning to pick the correct candidate h_i from a candidate set $\{h_1, \ldots, h_k\}$ is reduced to a binary classification problem by creating *pairs*: positive training instances $\langle h_1, h_2\rangle, \ldots, \langle h_1, h_k\rangle$ and negative instances $\langle h_2, h_1\rangle, \ldots, \langle h_k, h_1\rangle$. This set can then be used to train a binary classifier. At classification time the standard one-versus-all binarization method is applied to form all possible pairs of hypotheses. These are ranked according to the number of *classifier votes* they receive: a positive classification of $\langle h_k, h_i\rangle$ gives a vote to h_k whereas a negative one votes for h_i.

A vectorial representation of such pairs is the difference between the vectors representing the hypotheses in a pair. However, this assumes that features are explicit and already available whereas we aim at automatically generating implicit patterns with kernel meth-

[6] http://uima.apache.org/

ods. Thus, for keeping implicit the difference between such vectors we use the following preference kernel:

$$P_K(\langle h_1, h_2\rangle, \langle h'_1, h'_2\rangle) = K(h_1, h'_1) + K(h_2, h'_2) - K(h_1, h'_2) - K(h_2, h'_1), \tag{6.2}$$

where h_i and h'_i refer to two sets of hypotheses associated with two rankings and K is a kernel applied to pairs of hypotheses. It should be noted that we represent the latter as pairs of question and answer passage trees. More formally, given two hypotheses, $h_i = \langle h_i(q), h_i(a)\rangle$ and $h_i = \langle h'_i(q), h'_i(a)\rangle$, whose members are the question and answer passage trees, we define $K(h_i, h'_i)$ as

$$K_{\text{TK}}(h_i(q), h'_i(q)) + K_{\text{TK}}(h_i(a), h'_i(a)),$$

where K_{TK} can be any tree kernel function, e.g., STK or PTK. Finally, it should be noted that, to add traditional feature vectors to the reranker, it is enough to sum the product $(\mathbf{x}_{h_1} - \mathbf{x}_{h_2}) \cdot (\mathbf{x}_{h'_1} - \mathbf{x}_{h'_2})$ to the structural kernel P_K, where $\mathbf{x}_h$ is the feature vector associated with the hypothesis h.

6.3.2 Answer Sentence Selection

To solve the answer sentence selection problem we apply a *pointwise* reranking method (also formally described in Sec. 2.6), which essentially treats it as a classification problem. In other words, each question $\mathbf{q}_i \in \mathbf{Q}$ is associated with a list of candidate answer sentences $\{(r_{i1}, \mathbf{s}_{i1}), \ldots, (r_{im}, \mathbf{s}_{im})\}$, with $r_{ij} \in \{-1, +1\}$ indicating if a given candidate answer sentence s_{ij} contains a correct answer $(+1)$ or not (-1). Our goal is to learn a classifier model to predict if a given pair of a question and an answer sentence is correct or not. We use a binary SVM with tree kernels[7] to train an answer sentence classifier. The prediction scores obtained from a classifier are used to rerank the answer candidates, s.t. the sentences that are more likely to contain correct answers will be ranked higher than incorrect candidates.

QA pair classification with tree kernels

We define each input question-answer pair $\mathbf{x}$ as a triple composed of a question tree $\mathbf{T}_q$ and answer sentence tree $\mathbf{T}_s$ and a similarity feature vector $\mathbf{v}$, i.e., $\mathbf{x} = \langle \mathbf{T}_q, \mathbf{T}_s, \mathbf{v}\rangle$. Given two triples $\mathbf{x}^i$ and $\mathbf{x}^j$, we define the following kernel:

$$\begin{aligned} K(\mathbf{x}^i, \mathbf{x}^j) &= K_{\text{TK}}(\mathbf{T}^i_q, \mathbf{T}^j_q) \\ &+ K_{\text{TK}}(\mathbf{T}^i_s, \mathbf{T}^j_s) \\ &+ K_{\text{v}}(\mathbf{v}^i, \mathbf{v}^j), \end{aligned} \tag{6.3}$$

[7] disi.unitn.it/moschitti/Tree-Kernel.htm

where K_{TK} computes a structural kernel, e.g., tree kernel, and K_{v} is a kernel over feature vectors, e.g., linear, polynomial, gaussian, etc. Structural kernels can capture the structural representation of a question/answer pair whereas traditional feature vectors can encode some sort of similarity, e.g., lexical, syntactic, semantic, between a question and its candidate answer.

We prefer to split the kernel computation over a question/answer pair into two terms since tree kernels are very efficient and there are no efficient graph kernels that can encode exhaustively all graph fragments. It should be noted that the tree kernel sum does not capture feature pairs. Theoretically, for such purpose, a kernel product should be used. However, our experiments revealed that using the product is actually worse in practice. In contrast, we solve the lack of feature pairing by annotating the trees with relational tags which are supposed to link the question tree fragments with the related fragments from the answer sentence.

Such relational information is very important to improve the quality of the pair representation as well as the implicitly generated features.

6.3.3 Answer Sentence Extraction

The goal of answer extraction is to extract a text span from a given candidate answer sentence. Such span represents a correct answer phrase for a given question. Different from previous work that casts the answer extraction task as a tagging problem and apply a CRF to learn an answer phrase tagger [186], we take on a simpler approach using a kernel-based classifier.

In particular, we rely on the shallow tree representation, where text spans identified by a shallow syntactic parser serve as a source of candidate answers. Algorithm 11 specifies the steps to generate training data for our classifier. In particular, for each example representing a triple $\langle a, \mathbf{T}_q, \mathbf{T}_s \rangle$ composed of the answer key a, the question and the answer sentence trees, we generate a set of training examples E with every candidate chunk marked with an `ANS` tag (one at a time). To reduce the number of generated examples for each answer sentence, we only consider `NP` chunks, since other types of chunks, e.g., `VP`, `ADJP`, typically do not contain factoid answers. Finally, an original untagged tree is used to generate a positive example (line 8), when the answer sentence contains a correct answer, and a negative example (line 10), when it does not contain a correct answer.

At the classification time, given a question and a candidate answer sentence, all NP nodes of the sentence are marked with `ANS` (one at a time) as the possible answer, generating a set of tree candidates. Then, such trees are classified (using the kernel from Eq. 6.3) and the one with the highest score is selected. If no tree is classified as positive example we do not extract any answer.

Algorithm 11 Generate training data for answer extraction

```
1: for each ⟨a, T_q, T_s⟩ ∈ D do
2:   E ← ∅
3:   for each chunk ∈ extract_chunks(T_s) do
4:     if not chunk == NP then
5:       continue
6:     T'_s ← tagAnswerChunk(T_s, chunk)
7:     if contains_answer(a, chunk) then
8:       label ← +1
9:     else
10:      label ← −1
11:    e ← build_example(T_q, T'_s, label)
12:    E ← E ∪ {e}
13: return E
```

6.4 Feature Vector Representation

While the primary focus of our study is on the structural representations and relations between the question-answer pairs we also include some basic and more advanced features widely applied in QA. We use several similarity functions between q and a, computed over various input representations to form a feature vector. These feature vectors are used along with the structural models.

6.4.1 Basic features

N-gram overlap features. We compute a cosine similarity over question and answer: $sim_{COS}(q, a)$, where the input vectors are composed of: (i) word lemmas, (ii) bi-grams, (iii) part-of-speech tags, (iv) topics, and (v) dependency triplets. For the latter, we simply hash the string value of the predicate defining the triple together with its argument, e.g., *poss(name, twain)*. We also generalize the arguments of the predicates by using topics instead of words.

Tree kernel similarity. For the structural representations we also define a similarity based on the PTK score: $sim_{PTK}(q, a) = PTK(q, a)$, where the input trees can be raw constituency trees and shallow chunk trees used in structural representations. Note that this similarity is computed between the members of a q/a pair, thus, it is very different from the one defined by Eq. 6.2. We also compute these features over the trees where the lexicals are replaced with their associated topics.

NER relatedness. We also compute a feature that represents a match between a question category and the related named entity types extracted from the candidate answer. We simply count the proportion of named entities in the answer that correspond to the question type returned by the question classifier.

LDA. The similarity between a question **q** and a candidate answer **a** can be captured by the similarity between their topic distributions $\theta^{(q)}$ and $\theta^{(a)}$. For this purpose, we use the

symmetrized KL divergence:

$$sim_{KL}(q,a) = \frac{1}{2}[KL(\theta^{(q)}||\theta^{(c)}) + KL(\theta^{(c)}||\theta^{(q)})]$$

LDA provides yet another way to compute the similarity between two documents using conditional probabilities of one document given the topic distribution of the other. For a question **q** and its candidate answer **a** it can be estimated as follows:

$$P(q|a) = \prod_{w \in q} P(w|a) = \prod_{w \in q} \sum_{t \in T} P(w|z=t)P(z=t|a)$$

where the last term $P(z = t|a)$ is simply the probability of a topic $z = t$ under the topic distribution of an answer **a**. Hence, we define two similarities : $sim^{LDA_1}(q,a) = P(q|a)$ and $sim^{LDA_2}(q,a) = P(a|q)$, which compute the probability of the question being generated from the topic distribution of the answer and vice versa. Differently from the features derived from translation-based language models [183], which extract knowledge from q/a pairs, topic models use the distribution of words over the entire collection.

The total number of our basic features is 24. Although far from being complete, our features represent a good sample of typical features used in many QA systems to rank candidate answers. Nevertheless, in our study feature vectors serve a complementary purpose, while the main focus is to study the virtue of structural representations for reranking. The effect of a more extensive number of features computed for the q/a pairs has been studied elsewhere, e.g., [156].

6.4.2 Advanced features

This section describes a set of more involved features to model the similarity of q/a pairs. The features below are largely inspired by the feature sets used by the top performing system [14] in Semantic Textual Similarity (STS) task [2].

Additional word-overlap measures. We include additional word-overlap similarity metrics over lemmas for a given q/a pair by computing the longest common substring (subsequence) measures, and Greedy String Tiling. The longest common substring measure [55] determines the length of the longest string which is also a substring shared by a pair of text fragments. The longest common subsequence measure [7] considers as subsequence also strings that differ from word insertions or deletions. Greedy String Tiling [178] detects similarity of reordered text parts as it is able to identify multiple shared contiguous substrings.

Knowledge-based word similarity. We compute Resnik similarity [122] which is based on the WordNet hypernymy hierarchy and on semantic relatedness between concepts. The hierarchy is used to find a path between two concepts. Then, the semantic relatedness is determined by the lowest common concept subsuming both of them. The specificity of

the subsuming concept affects the similarity measure: more specific concepts contributes more than generic ones. The aggregation strategy by Mihalcea et al. [86] is applied to scale the measure from pairs of words to sentences.

Explicit Semantic Analysis (ESA) Similarity. ESA [48] maps a document into a vector of concepts extracted from Wikipedia. Thus, the meaning of text fragment is modeled by a set of natural concepts, which are described and defined by humans. Moreover, we used WordNet and Wiktionary as additional concepts.

Lexical Substitution. A supervised word sense disambiguation system [18] finds substitutions for a wide selection of frequent English nouns. Resnik and ESA features are then computed adding the substitutions to the text. This feature enables additional matches aleviating the lexical gap between text fragments.

Translation model. We integrate two similarity score features obtained from the METEOR scorer when treating both a question and a candidate as a translation source.

This feature set adds 19 more features to our basic features defined above, which results in total 43 features for our advanced vector-based model (V_{adv}).

6.5 Experiments

We evaluate our syntactic relational tree models on three tasks: (i) answer passage reranking; (ii) answer sentence selection; and (iii) answer extraction.

6.5.1 Answer Passage Reranking

First we consider the case where the answer candidates are passages. We only consider the use of basic feature sets described in Sec. 6.4.1, while the use of an extended set of features (including advanced features) is considered in answer sentence selection experiments in Sec. 6.5.2.

Experimental setup

SVM re-ranker. To train our models, we use SVM-light-TK[8], which enables the use of structural kernels [93] in SVM-light [68]. We use default parameters as described in [133]. We choose PTK as the re-ranker kernel to establish pairwise similarities between tree structures, since PTK is the most general kernel able to generate a vast number of tree fragments. It is also particularly suitable for tree representations using dependency structures. For explicit feature vectors we use polynomial kernel of degree 3, as it has better discriminative power w.r.t. linear kernel.

[8] http://disi.unitn.it/moschitti/Tree-Kernel.htm

LDA model. To train an LDA model we use an implementation of a parallelized Gibbs sampler from MALLET[9] library. We fix the number of iterations of the Gibbs sampler to 1000 and fix the other parameters as their default values. Since the LDA implementation performs an automatic tuning of prior parameters for the document-topic and word-topic distributions, we fixed them at their default values. As an input training data we perform basic stopword removal and lemmatization. At the inference time on the unseen test documents we set the number of iterations of the Gibbs sampler to 300.

Metrics. To measure the impact of the re-ranker on the output of our QA system, we use metrics most commonly used to assess the accuracy of QA systems: Precision at rank 1 (P@1), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). P@1 is the percentage of questions with a correct answer ranked at the first position, MRR is computed as follows: $MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list. For a set of queries Q MAP is the mean over the average precision scores for each query: $\frac{1}{Q}\sum_{q=1}^{Q} AveP(q)$.

Pipeline. We built the entire processing pipeline on top of the UIMA Framework, which provides a convenient abstraction for developing NLP annotators for analyzing unstructured content.

We have included many off-the-shelf NLP tools wrapping them as UIMA annotators to perform sentence detection, tokenization, Named Entity Recognition, parsing, chunking and lemmatization. We included other tools such as the Stanford Parser, the Berkeley parser, the ClearTK dependency parser, the Illinois Chunker, the RitaWordnet Lemmatizer, Mallet for LDA and the Snowball stemmer. Moreover, we created annotators for building new sentences representations starting from tools' annotations. For example, the component producing a tree representations containing POS tags and chunks outputs the result as a UIMA annotation. Fully integrated into the pipeline, there are also the question focus and question classifiers.

Data

Our experiments are carried out on TREC QA data, which is widely used in the evaluation of QA systems. In this task, answer passages containing correct information nuggets, i.e. answer keys, have to be extracted from a given text corpus, typically a large newswire corpus. It should be noted that the passages may contain multiple sentences. This setting is thus rather different from the one we consider in Sec. 6.3.2. In our experiments, we opted for questions from 2002 and 2003 years (TREC 11-12), which totals to 824 questions. AQUAINT newswire corpus is used for searching the supporting answers.

[9] http://mallet.cs.umass.edu

Table 6.3: Search Engine baselines on TREC QA.

Model	MAP	MRR	P@1
Lucene	0.16	21.57	13.83
Whoosh	0.20	25.92	17.59
Terrier	0.22	27.91	18.08

We also experimented with a large sample of Answerbag QA corpus explored in [133]. This community-driven QA collection is attractive since it contains a large portion of questions that have "professionally researched" answers. Such answers are provided by the website moderators and allow for training high quality models, alleviating the problem of high noise in the CQA collections. From the original corpus containing 180k question/answer pairs, we use 1,000 randomly sampled questions for testing and 10k for training a cross-domain model.

To train Question Focus and Category classifiers we follow the same setup as described in [137].

Search engines

For the questions from TREC 11-12 the supporting corpus is AQUAINT which represents a large collection of newswire text (3Gb) containing about 1 million documents. We perform indexing at the paragraph level by splitting each document into a set of paragraphs which are then added to the search index. The resulting index contains about 12 million items. For the Answerbag we index the entire 180k answers. For both TREC and Answerbag we retrieve a list of 50 candidate answers for each question.

We tested 3 open-source search-engines: Lucene, Whoosh and Terrier. The results are given in Table 6.3. Lucene implements the most basic retrieval model based on the cosine similarity and represents the weakest baseline, while Whoosh and Terrier, which use a more accurate BM25 scoring model, demonstrate better performance. Hence, in our further experiments we use BM25 implemented by Terrier as our baseline model.

Relational learning on TREC

To evaluate the performance of our retrieval model, we evaluate the following models previously introduced in Sec. 6.2:

- **BM25**: initial ranking obtained by the BM25 search engine model.
- **CH**: our shallow chunk tree model in proposed [133].
- **V**: reranker model using the set of *basic* features defined in Sec. 6.4.1.

Table 6.4: Answer passage reranking on TREC QA. † shows significant improvement (using a two tailed t-test with $p < 0.05$) w.r.t. to our baseline model CH [133], while ‡ compares F and TF w.r.t. CH+V. Model +TF does not provide significant improvement over +F.

Model	MAP	MRR	P@1
IR baseline models			
BM25	0.22	28.02	18.17
V	0.25	31.82	21.61
Syntactic tree models			
CH [133]	0.28	35.63	24.88
CH+V	0.30†	37.45†	27.91†
Semantic linking			
CH+V+F	0.32‡	39.48‡	29.63‡
CH+V+TF	0.32‡	39.49‡	30.00‡

- **CH+V**: a combination of tree structures encoding q/a pairs with similarity scores stored in the feature vector.
- **F**: relational linking of the question focus word and named entities of the corresponding type using Focus and Question classifiers.
- **TF**: a typed relational link refined with the question category.

Table 6.4 reveals that using feature vectors provides a good improvement over BM25 baseline. Most interestingly, the structural representations give a bigger boost in the performance. Combining the structural and feature vector representation in a single model results in further improvement.

Soft Word Matching

Here, we explore the effect of using word similarity measures derived from WordNet, SuperSense tags, word alignments and LDA topic models on the performance of the reranker. We test WordNet synonym groups (SYN) and a distance (DIST) metric between two concepts to identify semantically close words. For the latter we fix the similarity threshold at 0.2. For the topic model we match words based on their topic assignments obtained when performing inference on the unseen documents. We train a set of LDA models using a fixed number of topics $Z \in \{50, 100, 250, 500\}$. Table 6.5 shows that using SSENSE or LDA topic matching actually results in lower performance w.r.t. plain string matching used by CH+V. Additionally, using WordNet and word alignment for relational linking does not provide any interesting improvement. Our intuition is that most of the answer candidates have a relatively high word overlap with the question (since search engine retrieves candidates based on metrics derived from word-overlap measures), hence

Table 6.5: Results on TREC QA using various relational tagging schemes: with Wordnet (WN), with synonym (SYN), and hierarchy distance (DIST) matching, LDA, SSENSE, and word alignment (WALIGN) matching. Basic CH+V structure is used. None of the linking strategies provide significant improvement over LEMMA matching of CH+V.

MODEL	MAP	MRR	P@1
LEMMA	0.30	37.45	27.91
WN-DIST (0.2)	0.30	37.41	27.20
WN-SYN	0.30	37.54	27.82
LDA50	0.23	29.51	18.29
LDA100	0.28	35.41	24.67
LDA250	0.24	33.01	22.25
LDA500	0.24	32.24	21.34
SSENSE	0.29	36.32	26.77
WALIGN	0.30	37.64	28.04

using a plain string match on lemmas results in a rather good coverage of words between question and answer. Differently, using coarser word classes from SuperSense tagger and LDA results in much higher linking but, at the same time, they add a considerable large amount of noise. Thus, the use of larger coverage linking strategies seems to requires a definition of a more principled approach.

We provided a possible solution in [137], where we explore a set of supervised components to semantically link important concepts between questions and answers. We include such strategy in our experiments to build an even stronger baseline. The next section briefly reports the results using such linking models.

Relational learning using Question Focus and Question Class

In the following set of experiments, we test another strategy for linking structures for a given q/a pair. We use an automatically detected question focus word and a question category obtained from the question classifier to link the focus word with the related named entities in the answer (namely model F). Then, we refine the relational link by typing it with the question category (namely model TF).

Table 6.4 summarizes the performance of the CH+V model when coupled with F and TF strategies to link structures in a given q/a pair. The structural representations with F yields an interesting improvement, while further refining the relational tag by adding a question category (TF) gives no improvement with CH structure.

Table 6.6: Answer passage reranking on Answerbag.

Model	MAP	MRR	P@1
Baseline			
BM25	0.64±0.03	64.32±3.90	54.20±3.78
Basic structural representations			
CH+V	0.67±0.03	66.95±3.74	57.81±3.60
Refined relational tag			
CH+V+TF	0.68±0.03	67.73±3.72	58.10±3.51

Table 6.7: Testing model robustness: training on Answerbag and testing on TREC QA.

Model	MAP	MRR	P@1
BM25	0.22	27.91	18.08
V	0.23	29.21	19.30
CH+V	0.25	31.31	21.28
CH+V+F	0.27	33.53	22.81

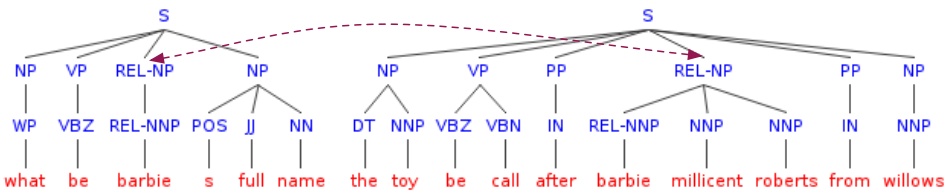


Figure 6.5: Basic CH representation of the example SV pair.

Learning cross-domain pairwise structural relationships

First, we provide the performance of the basic structural models and enriched with TF relational linking on the Answerbag (Table 6.6). In line with the results reported in [133] the CH+V improves the BM25. Using the typed focus linking strategy TF is giving a little further improvement.

Most interestingly, in our cross-domain experiment, when a model trained on the Answerbag, which is a completely different dataset coming from community QA, can be applied to the test questions from TREC (Table 6.7).

Error Analysis

Consider our running example q/a pair from Section 6.1. As the first candidate answer, the search engine retrieves the following incorrect passage: *"The autobiography of Mark Twain", Mark Twain.* It is relatively short and mentions the keywords {*Mark, Twain*} twice, which apparently results in a high score for the BM25 model. Instead, the search

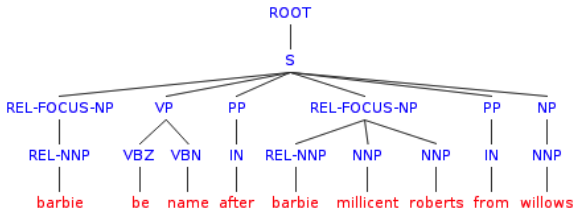


Figure 6.6: Structural representation of the answer from the SV pair with the relational focus tag.

engine ranks the correct answer at position 34. After re-ranking using the basic CH+V model the correct answer is promoted by 20 positions. While after using the CH+V+F model, where the `REL-FOCUS` relational tag is used to establish a connection between the focus word and the named entities in the answer, the correct answer advances to position 6. Below, we provide an intuition behind the merits of +F and +TF encoding question focus and question category into the basic models.

At test time the reranker classifies a list of the q/a pairs formed by a test question and its candidate answers. The list of prediction scores is then used to refine the initial ranking. The model learnt by the re-ranker contains q/a pairs from the training set, i.e. support vectors (SVs) which are matched against each candidate q/a pair. We isolated the following pair from SVs that has a high structural similarity with our running example:

Q: What is Barbie's full name?

A: The toy is called after Barbie Millicent Roberts from Willows.

As Fig. 6.5 reveals, despite differences in the surface forms of the words, it is similar to the structure in Fig. 6.1 (top). PTK extracts matching patterns, e.g. `[S NP [VP VBN] [PP IN] REL-NP]`, which yields a high similarity score boosting the rank of the correct candidate. However, we note that at the same time an incorrect candidate answer, e.g. *Mark Twain was accused of racist language.*, exhibits a similar pattern and also gets a high rank. The basic structural representation is not able to encode essential differences from the correct answer candidate. This poses a certain limitation on the discriminative power of CH and DEP representations.

Introducing a focus tag changes the structural representation of both q/a pairs, s.t. the correct q/a pair preserves the pattern (after identifying word *name* as focus and question category as HUM, it is transformed to `[S REL-FOCUS-NP [VP VBN] [PP IN] REL-FOCUS-NP]`), while it will be absent in the incorrect candidate. Thus, linking the focus word with the related named entities in the answer passage helps to discriminate between structurally similar yet semantically different candidates.

Another step towards a more fine-grained structural representation is to further specialize the relational focus tag achieved by TF model. We propose to augment the focus tag with the question category to avoid matches with other structurally similar but semantically different candidates. For example, a q/a pair found in the list of support

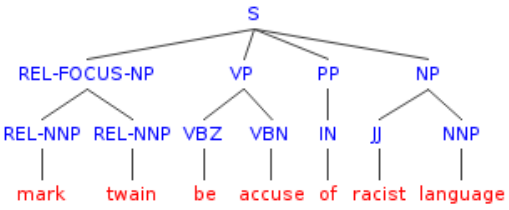


Figure 6.7: Incorrect answer with the relational focus tag.

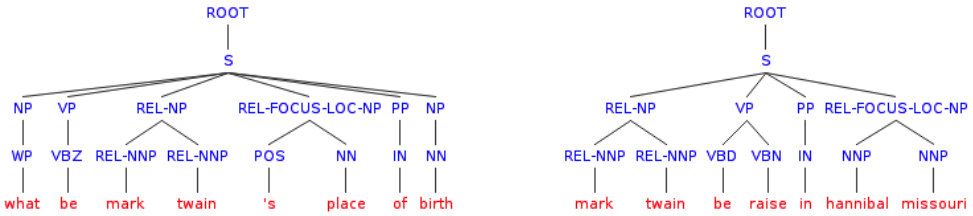


Figure 6.8: Example q/a pair from SVs not activated after using typed relation focus tag.

vectors:

Q: What is Mark Twain's place of birth?

A: Mark Twain was raised in Hannibal Missouri.

would exhibit high structural similarity even when relational focus is used (since the relational tag does not incorporate the question class LOC) but refining the focus tag with the question class will eliminate such cases.

6.5.2 Answer Sentence Selection

In this section, we test our structural CH and CH+F models on a sentence reranking task. For the latter several studies using a common dataset are available, thus it gives us the possibility to compare with several reranking systems on exactly the same test set. First, we briefly describe the experimental setup to replicate the setting of the previous work. Then, we propose an additional set of features to build a strong feature vector baseline model and finally, we compare our models to the previous state-of-the-art systems.

Experimental setup

We test our models on the manually curated[10] TREC QA dataset[11] from Wang et al. [175], which has been widely used for comparison of answer rerankers in previous work[12].

[10]Manual judgement of candidate answer sentences was carried out for the entire TREC 13 set and for the first 100 questions from TREC 8-12. The motivation behind this annotation effort is that TREC provides only the answer patterns to identify if a given passage contains a correct answer key or not. This results in many unrelated candidate answers marked as correct simply because they happen to contain a regex match with the answer key.

[11]http://cs.stanford.edu/people/mengqiu/data/qg-emnlp07-data.tgz

[12]Different from the experiments above, where we learn a candidate answer reranker over paragraphs, here the retrieved answer candidates represent a single sentence.

Table 6.8: Summary of the TREC data for answer sentence selection.

data	questions	candidates	correct
TRAIN	94	4718	348
ALL	1229	53417	6410
TEST	89	1517	284

In their setup, 100 manually judged questions from TREC 8-12 are used for training, while questions from TREC-13 are used for testing. Additionally, Wang et. al [175] provide a "noisy setting" experiment where 2,393 questions from the entire TREC 8-12 collection is used for training. This results in a lower performance of their system, which is explained by the inclusion of erroneous candidate answers treated as correct by TREC. Nevertheless, we compare with their best results that they obtained using their manually judged training set.

This setup enables direct comparison with previous work on answer sentence selection. Table 7.1 summarizes the data used for training and testing.

Results

Table 6.9 compares V_{adv} and CH structural model coupled with F relational linking strategy with the previous state-of-the-art systems[13]. In particular, we compare to four most recent state-of-the-art reranker models [58,174,175,186] that report their performance on the same questions and candidate sets from TREC 13 as provided by [175].

First note that, our combined set of basic and advanced features V_{adv} already represents a rather strong baseline model. Furthermore, combining it with CH representations gives state-of-the-art performance providing an improvement over the previous work with a large margin[14]. Finally, augmenting CH representation with F linking strategy yields additional 2 points in MAP and one point in MRR. This strongly indicates on the utility of using supervised components, e.g., question focus and question category classifiers coupled with NERs, to establish semantic mapping between words in a q/a pair.

Our kernel-based learning to rank approach is conceptually simpler than approaches in the previous work, as it relies on the structural kernels, e.g., PTK, to automatically extract salient syntactic patterns relating questions and answers. Moreover, the computational complexity of previous approaches limited their application to reranking of answer sentences, while our approach is demonstrated to work well also at the paragraph level.

[13] P@1 metric is omitted since it is not reported in the previous work.

[14] Given the fact that our system was trained on a smaller subset of TREC 8-12, we expect a potential increase in accuracy when trained on the full data.

Table 6.9: Answer sentence reranking on TREC 13. † indicates that improvement delivered by adding F linking to CH+V_{adv} model is significant ($p < 0.05$).

Model	MAP	MRR
Previous work		
Wang et al., 2007 [175]	0.6029	0.6852
Heilman & Smith, 2010 [58]	0.6091	0.6917
Wang & Manning, 2010 [174]	0.5951	0.6951
Yao et al., 2013 [186]	0.6307	0.7477
Feature vector models (Sec. 6.4)		
V	0.5331	0.5974
V_{ADV}	0.5627	0.6294
Basic features (Sec. 6.4.1)		
CH+V	0.6485	0.7244
CH+V+F	0.6781	0.7358
Advanced features (Sec. 6.4.2)		
CH+V_{ADV}	0.6611	0.7419
CH+V_{ADV}+F	**0.6829**†	**0.7520**†

6.5.3 Answer Extraction

Our experiments on answer extraction replicate the setting of [186], which is the most recent work on answer extraction reporting state-of-the-art results.

Recovering answers

Table 6.10 reports the accuracy of our model in recovering correct answers from a set of candidate answer sentences for a given question. Here the focus is on the ability of an answer extraction system to recuperate as many correct answers as possible from each answer sentence candidate. The set of extracted candidate answers can then be used to select a single best answer, which is the final output of the QA system for factoid questions. Recall (R) encodes the percentage of correct answer sentences for which the system correctly extracts an answer (for TREC 13 there are a total of 284 correct answer sentences), while Precision (P) reflects how many answers extracted by the system are actually correct. Clearly, having a high recall system, allows for correctly answering more questions. On the other hand, a high precision system would attempt to answer less questions (extracting no answers at all) but get them right.

We compare our results to a CRF model of [186] augmented with WordNet features (without forced voting) [15]. Unlike the CRF model which obtains higher values of precision, our system acts as a high recall system able to recover most of the answers from the correct

[15] We could not replicate the results obtained in [186] with the forced voting strategy. Thus such result is not included in Table 6.10.

answer sentences. Having higher recall is favorable to high precision in answer extraction since producing more correct answers can help in the final voting scheme to come up with a single best answer. To solve the low recall problem of their CRF model, Yao et al. [186] apply fairly complex outlier resolution techniques to force answer predictions, thus aiming at increasing the number of extracted answers.

To further boost the number of answers produced by our system we exclude negative examples (answer sentences not containing the correct answer) from training, which slightly increases the number of pairs with correctly recovered answers. Nevertheless, it has a substantial effect on the number of questions that can be answered correctly (assuming perfect single best answer selection). Clearly, our system is able to recover a large number of answers from the correct answer sentences, while low precision, i.e., extracting answer candidates from sentences that do not contain a correct answer, can be overcome by further applying various best answer selection strategies, which we explore in the next section.

Best Answer Selection

Since the final step of the answer extraction module is to select for each question a single best answer from a set of extracted candidate answers, an answer selection scheme is required.

We adopt a simple majority voting strategy, where we aggregate the extracted answers produced by our answer extraction model. Answers sharing similar lemmas (excluding stop words) are grouped together. The prediction scores obtained by the answer extraction classifier are used as votes to decide on the final rank to select the best single answer.

Table 6.11 shows the results after the majority voting is applied to select a single best answer for each candidate. A rather naïve majority voting scheme already produces satisfactory outcome demonstrating better results than the previous work. Our voting scheme is similar to the one used by [186], yet it is much simpler since we do not perform any additional hand tuning to account for the weight of the "forced" votes or take any additional steps to catch additional answers using outlier detection techniques applied in the previous work.

Discussion and Error Analysis

There are several sources of errors affecting the final performance of our answer extraction system: (i) chunking, (ii) named entity recognition and semantic linking, (iii) answer extraction, (iv) single best answer selection.

Chunking. Our system uses text spans identified by a chunker to extract answer candidates, which makes it impossible to extract answers that lie outside the chunk boundaries. Nevertheless, we found this to be a minor concern since for 279 out of total 284 candidate

Table 6.10: Results on answer extraction. P/R - precision and recall; pairs - number of QA pairs with a correctly extracted answer, q - number of questions with at least one correct answer extracted, F1 sets an upper bound on the performance assuming the selected best answer among extracted candidates is always correct. *-marks the setting where we exclude incorrect question answer pairs from training.

set	P	R	pairs	q	F1
Yao et al. [186]	25.7	23.4	73	33	-
+ WN	26.7	24.3	76	35	-
TRAIN	29.6	64.4	183	58	65.2
TRAIN*	15.7	71.8	204	66	74.1
Yao et al. [186]	35.2	35.1	100	38	-
+ WN	34.5	34.7	98	38	-
ALL	29.4	74.6	212	69	77.5
ALL*	15.8	76.7	218	73	82.0

Table 6.11: Results on finding the best answer with voting.

system	set	P	R	F1
Yao et al. [186]	TRAIN	55.7	43.8	49.1
+ forced		54.5	53.9	54.2
+ WN		55.2	53.9	54.5
this work		**66.2**	**66.2**	**66.2**
Yao et al. [186]	ALL	67.2	50.6	57.7
+ forced		60.9	59.6	60.2
+ WN		63.6	62.9	63.3
this work		**70.8**	**70.8**	**70.8**

sentences from TREC 13 the answers are recoverable within the chunk spans.

Semantic linking. Our structural model relies heavily on the ability of NER to identify the relevant entities in the candidate sentence that can be further linked to the focus word of the question. While our answer extraction model is working on all the NP chunks, the semantic tags from NER serve as a strong cue for the classifier that a given chunk has a high probability of containing an answer. Typical off-the-shelf NER taggers have good precision and low recall, s.t. many entities as potential answers are missed. In this respect, a high recall entity linking system, e.g., linking to wikipedia entities [121], is required to boost the quality of candidates considered for answer extraction. Finally, improving the accuracy of question and focus classifiers would allow for having more accurate input representations fed to the learning algorithm.

Answer Extraction. Our answer extraction model acts as a high recall system, while it suffers from low precision in extracting answers for many incorrect sentences. Improving the precision without sacrificing the recall would ease the successive task of best answer

selection, since having less incorrect answer candidates would result in a better final performance. Introducing additional constraints in the form of semantic tags to allow for better selection of answer candidates could also improve our system.

Best Answer Selection. We apply a naïve majority voting scheme to select a single best answer from a set of extracted answer candidates. This step has a dramatic impact on the final performance of the answer extraction system resulting in a large drop of recall, i.e., from 82.0 to 70.8 before and after voting respectively. Hence, a more involved model, i.e., performing joint answer sentence re-ranking and answer extraction, is required to yield a better performance.

6.6 Related Work

Work in QA shows that semantics and syntax are essential to retrieve answers, e.g., [59, 148]. However, most approaches in TREC were based on many complex heuristics and fine manual tuning, which require large effort for system engineering and often made the result not replicable. In contrast, our passage re-ranker is adaptable to any domain and can be also used as front end of more complex QA systems.

Previous studies closely to ours carry out passage reranking by exploiting structural information. In this perspective, a typical approach is to use subject-verb-object relations, e.g., as in [11]. Unfortunately, the large variability of natural language makes such triples rather sparse thus different methods explore soft matching (i.e., lexical similarity) based on answer types and named entity types, e.g., see [6]. Passage reranking using classifiers of question and answer pairs were proposed in [62, 120]. In this context, several approaches focused on reranking the answers to definition/description questions, e.g., [99, 141, 156]. [1] propose a cascading learning to rank approach, where the ranking produced by one ranker is used as input to the next stage.

Regarding kernel methods, the work in [91,95,96,98,99,133] were the first to exploit tree kernels for modeling answer re-ranking. However, their methods lack the use of important relational information between a question and a candidate answer, which is essential to learn accurate relational patterns. In this respect, a solution based on enumerating relational links was given in [194, 195] for the textual entailment task but it is computationally too expensive for the large dataset of QA. Some faster versions were provided in [92, 193], which may be worth to try. In contrast, we design our re-ranking models with shallow trees encoding the output of question and focus classifiers connected to the NE information derived from the answer passage. This provides more effective relational information, which allows our model to significantly improve on previous rerankers.

Our relational structures are based on a shallow tree from [133] and we reuse our semantic linking strategy from [137], where question focus is linked to the related named entities (as identified by the question category). Additionally, this chapter studies a

number of linking strategies to establish connections between related tree fragments from questions and answer passages. Regarding the experimental evaluation, different from our previous work, where experiments were conducted for an answer passage reranking task on a subset of TREC QA data, in this work, we also include experiments for the answer sentence selection task on a public TREC 13 benchmark, such that we can compare to the previous state-of-the-art methods. We show that highly discriminative features can be, in fact, automatically extracted and learned by using our kernel learning framework. Our approach does not require manual feature engineering to encode input structures via similarity features. We treat the input q/a pairs directly encoding them via linguistic trees, and more powerful features can be encoded by injecting additional semantic information directly into the tree nodes.

Regarding previous state of the art in answer sentence rerankers, Wang et al., 2007 [175] use quasi-synchronous grammar to model relations between a question and a candidate answer with the syntactic transformations. Heilman & Smith, 2010 [58] develop an improved Tree Edit Distance (TED) model for learning tree transformations in a q/a pair. They search for a good sequence of tree edit operations using complex and computationally expensive Tree Kernel-based heuristic. Wang & Manning, 2010 [174] develop a probabilistic model to learn tree-edit operations on dependency parse trees. They cast the problem into the framework of structured output learning with latent variables. The model of Yao et al., 2013 [186] applies linear chain CRFs with features derived from TED to automatically learn associations between questions and candidate answers.

6.7 Summary

In this chapter we demonstrate the effectiveness of handling the input structures representing QA pairs directly vs. using explicit feature vector representations, which typically require substantial feature engineering effort. Our approach relies on a kernel-based learning framework, where structural kernels, e.g., tree kernels, are used to handle automatic feature engineering. It is enough to specify the desired type of structures, e.g., shallow, constituency, dependency trees, representing question and its candidate answer sentences and let the kernel learning framework learn to use discriminative tree fragments for the target task.

An important feature of our approach is that it can effectively combine together different types of syntactic and semantic information, also generated by additional automatic classifiers, e.g., focus and question classifiers. We augment the basic structures with additional relational and semantic information by introducing special tag markers into the tree nodes. Using the structures directly in the kernel learning framework makes it easy to integrate additional relational constraints and semantic information directly in the structures.

The comparison with previous work on a public benchmark from TREC suggests that our approach is very promising as we can improve the state of the art in both answer selection and extraction by a large margin (up to 22% of relative improvement in F1 for answer extraction). Our approach makes it relatively easy to integrate other sources of semantic information, among which the use of Linked Open Data can be the most promising to enrich the structural representation of q/a pairs.

To achieve state-of-the-art results in answer sentence selection and answer extraction, it is sufficient to provide our model with a suitable tree structure encoding relevant syntactic information, e.g., using shallow, constituency or dependency formalisms. Moreover, additional semantic and relational information can be easily plugged in by marking tree nodes with special tags. We believe this approach greatly eases the task of tedious feature engineering that will find its applications well beyond QA tasks.

Finally, the inefficacy of using topic models, WordNet, SuperSense and word alignment studied in this chapter suggests that information produced by unsupervised methods has still to be carefully considered. Therefore, studying ways to utilize semantic resources at their best is a natural future extension of this paper.

Chapter 7

Deep Learning models of input texts

In the previous chapters we have demonstrated the effectiveness of our syntactic tree representations to model input texts, e.g. Youtube comments in Chapter 4, text pairs for Semantic Textual Similarity and Microblog retrieval in Chapter 5, and question-answer pairs in Chapter 6. Thanks to the expressiveness of tree kernels that implicitly generate rich feature spaces it is possible to learn discriminative syntactic patterns thus largely reducing the feature engineering effort. Additionally, injecting semantic information into the tree nodes leads to significant boosts in accuracy increasing the informativeness of the generated features.

However, similarly to much simpler bag-of-words models, one of the main limitations of these approaches remains largely unaddressed: words in the input texts are treated as atomic units, e.g., matching between the words is always a hard match. While syntactic tree structures have the benefit of encoding the syntactic context around words through the generated tree fragments (that do not contain lexicals) thus partly allowing for the soft matching between input texts through matching between their syntactic contexts, still the problem of semantic matching between word remains problematic. Additionally, the definition of a tree kernel function together with its hyper-parameters remains fixed throughout the training process, thus it cannot be directly tuned on the training data.

On the other hand, distributional approaches that model words as vectors are much more effective in establishing a semantic match between words or phrases. Recently, deep learning approaches have been applied to generalize the distributional word matching problem to matching sentences and take it one step further by learning the optimal sentence representations for a given task. Deep neural networks have already claimed state-of-the-art performance in many computer vision, speech recognition, and natural language tasks.

In this chapter, we apply convolutional neural networks for modelling input texts and text pairs. We design a state-of-the-art architecture for Twitter Sentiment Analysis addressing two subtasks that treat the problem of sentiment classification at different

levels of granularity: term-level and message-level. We also propose a three-step process to initialize its weights which is a key to obtain the best results. Our experimental evaluation on a recent benchmark from Semeval-2015 [129] shows that our model ranks 1st on the term-level subtask (among 11 submissions) and 2nd on the message-level subtask (among 40 systems).

Considering our models of the input text pairs, we present a novel deep learning architecture for Microblog retrieval and Question Answering. Similarly to our idea of relational syntactic representations for modelling question-answer pairs (Chapter 6), where we tag tree nodes covering matching words in a pair, we also encode into the input to the network the fact that certain words in a pair co-occur. The main difference w.r.t. the previous apporach is that this information is represented, which can be learned from the training data. Effectively, we are augmenting word embeddings with additional parameters (dimensions) to represent the fact that words in a pair are related. These parameters are then tuned by the network. Modelling overlapping words in a pair allows the network to better capture the interactions between questions and answers resulting in a significant boost in the accuracy.

We test our deep learning system on two popular retrieval tasks from TREC: Question Answering and Microblog Retrieval. Our model demonstrates strong performance on the first task beating previous state-of-the-art systems by about 5% absolute points in MAP and 3% in MRR and shows comparable results on tweet reranking, while enjoying the benefits of no manual feature engineering and no additional syntactic parsers.

7.1 Overview

In this section we provide a brief overview of our approach to model short texts (tweets) to build a state-of-the-art sentiment analysis system and our deep learning architecture to tackle a more complex problem of modelling text pairs for Microblog retrieval and answer sentence selection.

7.1.1 Modelling tweets

In this chapter we describe our deep convolutional neural network for sentiment analysis of tweets. Its architecture is most similar to the deep learning systems presented in [72,76] that have recently established new state-of-the-art results on various NLP sentence classification tasks also including sentiment analysis. Convolutional neural networks have been also successfully applied in various IR applications, e.g., [143, 144]. While already demonstrating excellent results, training a convolutional neural network that would beat hand-engineered approaches that also rely on multiple manual and automatically constructed lexicons, e.g. [89, 182], requires careful attention. This becomes an even harder

problem especially in cases when the amount of labelled data is relatively small, e.g., thousands of examples.

It turns out that providing the network with good initialisation parameters can have a significant impact on the accuracy of the trained model. To address this issue, we propose a three-step process to train our deep learning model for sentiment classification. Our approach can be summarized as follows: (i) word embeddings are initialized using a neural language model [87, 128], which is trained on a large unsupervised collection of tweets; (ii) we use a convolutional neural network to further refine the embeddings on a large distant supervised corpus [52]; (iii) the word embeddings and other parameters of the network obtained at the previous stage are used to initialize the network with the same architecture, which is then trained on a supervised corpus from Semeval-2015 [129].

We apply our deep learning model on two subtasks of Semeval-2015 Twitter Sentiment Analysis (Task 10) challenge: phrase-level (subtask A) and message-level (subtask B). Our system achieves high results on the official tests sets of the phrase-level and on the message-level subtasks. In addition to the above test sets, we also used the so-called progress test set, which consists of five test sets, where our system again outperforms most of the systems participated in the challenge. In particular, if we ranked all systems (including ours) according to their accuracy on each of the six test sets and compute their average ranks, our model would be ranked first in both subtasks, A and B.

7.1.2 Modelling text pairs

Encoding query-document pairs into discriminative feature vectors that are input to a learning-to-rank algorithm is a critical step in building an accurate reranker. The core assumption is that relevant documents have high semantic similarity to the queries and, hence, the main effort lies in mapping a query and a document into a joint feature space where their similarity can be efficiently established.

The most widely used approach is to encode input text pairs using many complex lexical, syntactic and semantic features and then compute various similarity measures between the obtained representations.

The first step is to map input texts to a representation using various lexical, syntactic and semantic units such as words, ngrams, part-of-speech tags, dependency chains, predicate-argument relations, etc. These units are then translated into feature vectors using one-hot encoding scheme. Having mapped an input query and a document into a joint feature space, their similarity can be easily computed by an inner product between their feature vectors. The obtained similarity scores computed over various representations encode the input query-document pairs into a feature vector which is fed to a learning to rank classifier.

Many state-of-the-art approaches to reranking follow that schema. For example, in answer passage reranking [157] employ complex linguistic features, modelling syntactic

and semantic information as bags of syntactic and semantic role dependencies and build similarity and translation models over these representations.

However, the choice of representations and features is a completely empirical process, driven by the intuition, experience and domain expertise. Moreover, although using syntactic and semantic information has been shown to improve performance, it can be computationally expensive and require a large number of external tools — syntactic parsers, lexicons, knowledge bases, etc. Furthermore, adapting to new domains requires additional effort to tune feature extraction pipelines and adding new resources that may not even exist.

Recently, it has been shown that the problem of semantic text matching can be efficiently tackled using distributional word matching, where a large number of lexical semantic resources are used for matching questions with a candidate answer [161].

Deep learning approaches generalize the distributional word matching problem to matching sentences and take it one step further by learning the optimal sentence representations for a given task. Deep neural networks are able to effectively capture the compositional process of mapping the meaning of individual words in a sentence to a continuous representation of the sentence. In particular, it has been recently shown that convolutional neural networks are able to efficiently learn to embed input sentences into low-dimensional vector space preserving important syntactic and semantic aspects of the input sentence, which leads to state-of-the-art results in many NLP tasks [72, 76, 189]. Perhaps one of the greatest advantages of deep neural networks is that they are trained in an end-to-end fashion, thus removing the need for manual feature engineering and greatly reducing the need for adapting to new tasks and domains.

In this chapter, we describe a novel deep learning architecture for reranking short texts, where questions and documents are limited to a single sentence. The main building blocks of our architecture are two distributional sentence models based on convolutional neural networks. These underlying sentence models work in parallel, mapping queries and documents to their distributional vectors, which are then used to learn the semantic similarity between them.

The distinctive properties of our model are: (i) we use a state-of-the-art distributional sentence model for learning to map input sentences to vectors, which are then used to measure the similarity between them; (ii) our model encodes query-document pairs in a richer representation using not only their similarity score but also their intermediate representations; (iii) the architecture of our network makes it straightforward to include any additional feature vectors to the model; and finally (iv) no expensive pre-processing steps are necessary, i.e., our model does not require manual feature engineering or external resources. We only require to initialize word embeddings from some large unsupervised corpora. However, given a large training corpora our network can also optimize the embeddings directly for the task, thus omitting the need for pre-training of the word

embeddings.

Our sentence model is based on a convolutional neural network architecture that has recently showed state-of-the-art results on many NLP sentence classification tasks [72,76]. However, our model uses it only to generate intermediate representation of input sentences for computing their similarity. To compute the similarity score we use an approach used in the deep learning model of [189], which recently established new state-of-the-art results on answer sentence selection task. However, their model operates only on unigram or bigrams, while our architecture learns to extract and compose n-grams of higher degrees, thus allowing for capturing longer range dependencies. Additionally, our architecture uses not only the intermediate representations of questions and answers to compute their similarity but also includes them in the final representation, which constitutes a much richer representation of the question-answer pairs. Finally, our model is trained end-to-end, while in [189] they use the output of their deep learning model to learn a logistic regression classifier.

We test our model on two popular retrieval tasks from TREC: answer sentence selection and Microblog retrieval. Our model shows a considerable improvement on the first task beating recent state-of-the-art system by 3% absolute points in both MAP and MRR. On the second task, our model demonstrates that previous state-of-the-art retrieval systems can benefit from using our deep learning model.

7.1.3 Our contributions

In the following we describe our contributions:

- We present a state-of-the-art convolutional neural network for sentiment classification of tweets at the term-level and message-level.
- We propose a three-step process to pre-initialize the model weights of our convolutional neural network that is the key to achieve state-of-the-art results on Twitter Sentiment Analysis tasks.
- We present a novel deep learning architecture for modelling text pairs demonstrating its effectiveness on two popular benchmarks from TREC: Microblog retrieval and answer sentence selection.
- We show that a powerful sentence model has a large impact on the model accuracy as superior intermediate representation of input sentences in a pair helps to establish more accurate similarity.
- Our experimental findings verify that modelling overlapping words in a pair has a significant role in obtaining good results. Encoding this information directly into word embeddings works better than using a word overlap feature vector, as these parameters can be tuned during the training process.

In the remainder of this chapter, Section 7.2 provides a brief overview of the main components of a typical convolutional neural network that is the core component of our deep learning systems. Next, Sec. 7.3 describes our Convolutional Neural Network for modelling tweets to build a state-of-the-art sentiment classifier. In Sec. 7.4 we describe our deep learning architecture of modelling text pairs for Microblog retrieval and answer sentence selection for Question Answering. Sec. 7.6 describes the related work and Sec. 7.7 summarizes our findings.

7.2 Preliminaries: Convolutional Neural Networks

In this section we provide a brief overview of the main components (layers) of a typical convolutional neural network (ConvNet) for dealing with natural language texts, e.g., sentence matrix, convolutional layer, pooling, activation units, hidden layer, etc.

7.2.1 Sentence matrix

The input to the network are raw words that need to be translated into real-valued feature vectors to be processed by subsequent layers of the network.

ConvNets treat an input sentence $\mathbf{s}$ as a sequence of words: $[w_i, .., w_{|s|}]$, where each word is drawn from a vocabulary V. Words are represented by distributional vectors $\mathbf{w} \in \mathbb{R}^d$ looked up in a word embeddings matrix $\mathbf{W} \in \mathbb{R}^{d \times |V|}$ which is formed by concatenating embeddings of all words in V. For convenience and ease of lookup operations in $\mathbf{W}$, words are mapped to indices $1, \ldots, |V|$.

For each input sentence $\mathbf{s}$ we build a sentence matrix $\mathbf{S} \in \mathbb{R}^{d \times |s|}$, where each column i represents a word embedding $\mathbf{w}_i$ at the corresponding position i in a sentence (see Fig. 7.1):

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \ldots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

To learn to capture and compose features of individual words in a given sentence from low-level word embeddings into higher level semantic concepts, the neural network applies a series of transformations to the input sentence matrix $\mathbf{S}$ using convolution, non-linearity and pooling operations, which we describe next.

7.2.2 Convolution feature maps

The aim of the convolutional layer is to extract patterns, i.e., discriminative word sequences found within the input sentences that are common throughout the training instances.

More formally, the convolution operation $*$ between two vectors $\mathbf{s} \in \mathbb{R}^{1\times|s|}$ and $\mathbf{f} \in \mathbb{R}^{1\times m}$ (called a filter of size m) results in a vector $\mathbf{c} \in \mathbb{R}^{|s|+m-1}$ where each component is as follows:

$$\mathbf{c}_i = (\mathbf{s} * \mathbf{f})_i = \mathbf{s}^T_{[i-m+1:i]} \cdot \mathbf{f} = \sum_{k=i}^{i+m-1} s_k f_k \tag{7.1}$$

The range of allowed values for i defines two types of convolution: *narrow* and *wide*. The *narrow* type restricts i to be in the range $[1, |s| - m + 1]$, which in turn restricts the filter width to be $\leq |s|$. To compute the *wide* type of convolution i ranges from 1 to $|s|$ and sets no restrictions on the size of m and s. The benefits of one type of convolution over the other when dealing with text are discussed in detail in [72]. In short, the *wide* convolution is able to better handle words at boundaries giving equal attention to all words in the sentence, unlike in *narrow* convolution, where words close to boundaries are seen fewer times. More importantly, *wide* convolution also guarantees to always yield valid values even when $\mathbf{s}$ is shorter than the filter size m. Hence, we use *wide* convolution in our sentence model. In practice, to compute the *wide* convolution it is enough to pad the input sequence with $m - 1$ zeros from left and right.

Given that the input to our ConvNet are sentence matrices $\mathbf{S} \in \mathbb{R}^{d\times|s|}$, the arguments of Eq. 7.2 are matrices and a convolution filter is also a matrix of weights: $\mathbf{F} \in \mathbb{R}^{d\times m}$. More formally, the convolution operation $*$ between an input matrix $\mathbf{s} \in \mathbb{R}^{d\times|s|}$ and a filter $\mathbf{F} \in \mathbb{R}^{d\times m}$ of width m results in a vector $\mathbf{c} \in \mathbb{R}^{|s|+m-1}$ where each component is computed as follows:

$$\mathbf{c}_i = (\mathbf{S} * \mathbf{F})_i = \sum_{k,j} (\mathbf{S}_{[:,i-m+1:i]} \otimes \mathbf{F})_{kj} \tag{7.2}$$

where $\otimes$ is the element-wise multiplication and $\mathbf{S}_{[:,i-m+1:i]}$ is a matrix slice of size m along the columns. Note that the convolution filter is of the same dimensionality d as the input sentence matrix. As shown in Fig. 7.1, it slides along the column dimension of $\mathbf{S}$ producing a vector $\mathbf{c} \in \mathbb{R}^{1\times(|s|-m+1)}$ in output. Each component c_i is the result of computing an element-wise product between a column slice of $\mathbf{S}$ and a filter matrix $\mathbf{F}$, which is then summed to a single value.

An alternative way of computing a convolution was explored in [72], where a series of convolutions are computed between each row of a sentence matrix and a corresponding row of the filter matrix. Essentially, it is a vectorized form of 1d convolution applied between corresponding rows of $\mathbf{S}$ and $\mathbf{F}$. As a result, the output feature map is a matrix $\mathbf{C} \in \mathbb{R}^{d\times|s|-m+1}$ rather then a vector as above. While, intuitively, being a more general way to process the input matrix $\mathbf{S}$, where individual filters are applied to each respective dimension, it introduces more parameters to the model and requires a way to reduce the dimensionality of the resulting feature map. To address this issue, the authors apply a folding operation, which sums every two rows element-wise, thus effectively reducing the

size of the representation by 2.

So far we have described a way to compute a convolution between the input sentence matrix and a single filter. In practice, deep learning models apply a set of filters that work in parallel generating muliple feature maps (also shown on Fig. 7.1). The resulting filter bank $\mathbf{F} \in \mathbb{R}^{n \times d \times m}$ produces a set of feature maps of dimension $n \times (|s| - m + 1)$.

In practice, we also add a bias vector $\mathbf{b}$ [1] $\in \mathbb{R}^n$ to the result of a convolution – a single b_i value for each feature map $\mathbf{c}_i$.

7.2.3 Activation units

To allow the network learn non-linear decision boundaries, each convolutional layer is typically followed by a non-linear activation function $\alpha()$ applied element-wise to the output of the preceding layer. Among the most common choices of activation functions are the following: sigmoid (or logistic), hyperbolic tangent *tanh*, and a rectified linear (ReLU) function defined as simply $max(0, \mathbf{x})$ to ensure that feature maps are always positive. The choice of activation function has been shown to affect the convergence rate and the quality of obtained the solution. In particular, [100] shows that rectified linear unit has significant benefits over sigmoid and *tanh* overcoming some of the their shortcomings.

7.2.4 Pooling

The output from the convolutional layer (passed through the activation function) are then passed to the pooling layer, whose goal is to aggregate the information and reduce the representation. The result of the pooling operation is:

$$\mathbf{c}_{\text{pooled}} = \begin{bmatrix} \text{pool}(\alpha(\mathbf{c}_1 + b_1 * \mathbf{e})) \\ \dots \\ \text{pool}(\alpha(\mathbf{c}_n + b_n * \mathbf{e})) \end{bmatrix}$$

where $\mathbf{c}_i$ is the ith convolutional feature map with added bias (the bias is added to each element of $\mathbf{c}_i$ and $\mathbf{e}$ is a unit vector of the same size as $\mathbf{c}_i$) and passed through the activation function $\alpha()$.

There are two conventional choices for the pool$(\cdot)$ operation: average and max. Both operations apply to columns of the feature map matrix, by mapping them to a single value: $\text{pool}(\mathbf{c}_i) : \mathbb{R}^{1 \times (|s|+m-1)} \rightarrow \mathbb{R}$. This is also demonstrated in Fig. 7.1.

Both *average* and *max* pooling methods exhibit certain disadvantages: in average pooling, all elements of the input are considered, which may weaken strong activation values. This is especially critical with *tanh* non-linearity, where strong positive and negative activations can cancel each other out.

[1] bias is needed to allow the network learn an appropriate threshold

The *max* pooling is used more widely and does not suffer from the drawbacks of average pooling. However, as shown in [197], it can lead to strong overfitting on the training set and, hence, poor generalization on the test data. To mitigate the overfitting issue of max pooling several variants of stochastic pooling have been proposed in [197].

Recently, *max* pooling has been generalized to k-max pooling [72], where instead of a single max value, k values are extracted in their original order. This allows for extracting several largest activation values from the input sentence. As a consequence deeper architectures with several convolutional layers can be used. In [72], the authors also propose dynamic k-max pooling, where the value of k depends on the sentence size and the level in the convolution hierarchy [72].

Convolutional layer passed through the activation function together with pooling layer acts as a non-linear feature extractor. Given that multiple feature maps are used in parallel to process the input, deep learning networks are able to build rich feature representations of the input.

7.2.5 Hidden layers

Often additional hidden layers right before the softmax layer (described next) are used to allow for modelling interactions between the components of the output from the previous layer. It computes the following transformation:

$$\alpha(\mathbf{w}_h \cdot \mathbf{x} + b),$$

where $\mathbf{w}_h$ is the weight vector of the hidden layer and $\alpha()$ is the non-linearity.

7.2.6 Softmax

The output of the penultimate convolutional and pooling layers $\mathbf{x}$ is passed to a fully connected softmax layer. It computes the probability distribution over the labels:

$$\begin{aligned} P(y = j|\mathbf{x}, \mathbf{s}, \mathbf{b}) &= \text{softmax}_j(\mathbf{x}^T\mathbf{w} + \mathbf{b}) \\ &= \frac{e^{\mathbf{x}^T\mathbf{w}_j + b_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T\mathbf{w}_k + b_k}}, \end{aligned}$$

where $\mathbf{w}_k$ and b_k are the weight vector and bias of the k-th class. $\mathbf{x}$ can be thought of as a final abstract representation of the input example obtained by a series of transformations from the input layer through a series of convolutional and pooling operations.

7.2.7 Training

ConvNets are typically trained to minimise the cross-entropy (negative conditional log-likelihood of the data) cost function:

$$\begin{aligned} \mathcal{C} &= -\log \prod_{i=1}^{N} p(y_i|\mathbf{q}_i, \mathbf{d}_i) + \lambda\|\theta\|_2^2 \\ &= -\sum_{i=1}^{N}[y_i \log \mathbf{a}_i + (1-y_i)\log(1-\mathbf{a}_i)] + \lambda\|\theta\|_2^d, \end{aligned} \tag{7.3}$$

where $\mathbf{a}$ is the output from the softmax layer.

The parameters of the network are optimized with stochastic gradient descent (SGD) using backpropogation algorithm to compute the gradients. To speedup the convergence rate of SGD various modifications to the update rule have been proposed: *momentum*, *Adagrad* [41], *Adadelta* [196], etc. *Adagrad* scales the learning rate of SGD on each dimension based on the $l2$ norm of the history of the error gradient. *Adadelta* uses both the error gradient history like *Adagrad* and the weight update history. It has the advantage of not having to set a learning rate at all.

7.2.8 Regularization

While neural networks have a large capacity to learn complex decision functions they tend to easily overfit especially on small and medium sized datasets. To mitigate the overfitting issue the cost function is augmented with l_2-norm regularization terms for the parameters of the network.

Another popular and effective technique to improve regularization of the ConvNets is dropout [153]. Dropout prevents feature co-adaptation by setting to zero (dropping out) a portion of hidden units during the forward phase when computing the activations at the softmax output layer. As suggested in [53] dropout acts as an approximate model averaging.

7.3 Modelling tweets

This section describes the architecture of our convolutional neural network and our parameter initialization process process we follow to train it.

7.3.1 Network architecture

The architecture of our convolutional neural network for sentiment classification is shown on Fig. 7.1. It is mainly inspired by the architectures used in [72,76] for performing various sentence classification tasks. Given that our training process (described in Sec. 7.3.3) requires to run the network on a rather large corpus, our design choices are mainly driven by the computational efficiency of our network. Hence, different from [72] that presents

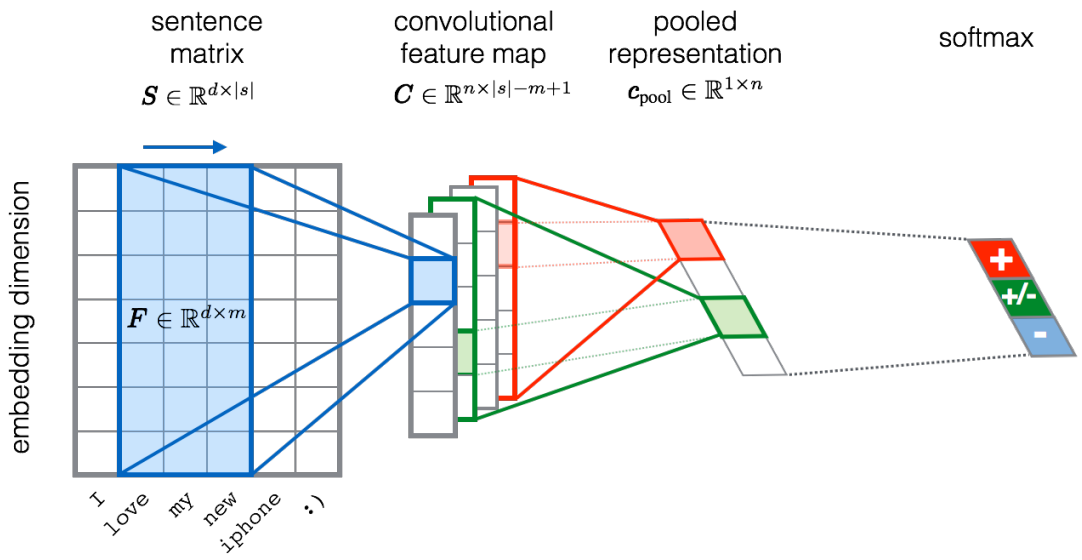


Figure 7.1: The architecture of our deep learning model for sentiment classification.

an architecture with several layers of convolutional feature maps, we adopt a single level architecture, which has been shown in [76] to perform equally well.

Our network is composed of an input sentence matrix, a single convolutional layer followed by a ReLU non-linearity, *max* pooling and a soft-max classification layer.

7.3.2 Phrase-level sentiment analysis

To perform phrase-level sentiment analysis, we feed the network with an additional input sequence indicating the location of the target phrase in a tweet. The elements are encoded using only two word types: tokens spanning the phrase to be predicted are encoded with 1s and all the other with 0s. Each word types is associated with its own embedding. So, when tackling the phrase-level sentiment classification, we form a sentence matrix $\mathbf{S}$ as follows: for each token in a tweet we have to look up its corresponding word embedding in the word matrix $\mathbf{W}$, and the embedding for one of the two word types. Hence, the input sentence matrix is augmented with an additional set of rows from the word type embeddings. Other than that, the architecture of our network remains unchanged.

This ends the description of our convolutional neural network for sentiment classification of tweets.

7.3.3 Initializing the model parameters

Convolutional neural networks live in the world of non-convex function optimization leading to locally optimal solutions. Hence, starting the optimization from a good point can be crucial to train an accurate model. We propose the following 3-step process to initialize

the parameter weights of the network:

1. Given that the largest parameter of the network is the word matrix $\mathbf{W}$, it is crucial to feed the network with the high quality embeddings. We use a popular `word2vec` neural language model [87] to learn the word embeddings on an unsupervised tweet corpus. For this purpose, we collect 50M tweets over the two-month period. We perform minimal preprocessing tokenizing the tweets, normalizing the URLs and author ids. To train the embeddings we use a skipgram model with window size 5 and filtering words with frequency less than 5.

2. When dealing with small amounts of labelled data, starting from pre-trained word embeddings is a large step towards successfully training an accurate deep learning system. However, while the word embeddings obtained at the previous step should already capture important syntactic and semantic aspects of the words they represent, they are completely clueless about their sentiment behaviour. Hence, we use a distant supervision approach [52] using our convolutional neural network to further refine the embeddings.

3. Finally, we take the the parameters θ of the network obtained at the previous step and use it to initialize the network which is trained on a supervised training corpus from Semeval-2015 [129].

7.4 Modelling text pairs

This section explains our deep learning model for reranking short text pairs. Its main building blocks are two distributional *sentence models* based on convolutional neural networks (ConvNets). These underlying sentence models work in parallel mapping queries and documents to their distributional vectors, which are then used to learn the semantic similarity between them.

In the following, we first describe our *sentence model* for mapping queries and documents to their intermediate representations and then describe how they can be used for learning semantic matching between input query-document pairs.

7.4.1 Sentence model

The architecture of our ConvNet for mapping sentences to feature vectors is similar to that shown on Fig. 7.1 (omitting the output softmax layer). It is mainly inspired by the architectures used in [72,76] for performing various sentence classification tasks. However, different from previous work the goal of our distributional sentence model is to learn intermediate representations of the queries and documents, which are then used for computing their semantic matching.

Hence, similarly to the ConvNet architecture for modelling tweets (Sec. 7.3) our sentence model is composed of a single *wide* convolutional layer followed by a ReLU nonlinearity and a simple *max* pooling.

7.4.2 Encoding overlapping words

Each dimension of the word embeddings represents some latent aspect of a word and as shown in [87] word embeddings learned by the neural language model capture various syntactic and semantic aspects of the words they represent.

When modelling text pairs it can be beneficial to augment word embeddings with additional dimensions, for example, to represent the fact that certain words in a pair are overlapping or semantically similar. For this purpose, we follow an approach of [33], where for each word w in the input sentence we associate an additional *word overlap* indicator feature $o \in \{0, 1\}$, where 1 corresponds to words that overlap in a given pair and 0 otherwise. To decide if the words overlap we apply simple string matching. Hence, we require an additional lookup table layer for the word overlap features $LT_{\mathbf{W}^o}(\cdot)$ with parameters $\mathbf{W}^o \in \mathbb{R}^{d_o \times 2}$, where $d_o \in \mathbb{N}$ is the number of dimensions to encode word overlap features and is a hyper-parameter of the model.

Given a word w_i its final word embedding $\mathbf{w}_i \in \mathbb{R}^d$ (where $d = d_w + d_o$) is obtained by concatenating the output of two lookup table operations $LT_{\mathbf{W}}(w_i)$ and $LT_{\mathbf{W_o}}(w_i)$. Finally, for each input sentence $\mathbf{s}$ we build a sentence matrix $\mathbf{S} \in \mathbb{R}^{d \times |s|}$ where each i-th column corresponds to a word embedding $\mathbf{w}_i$.

7.4.3 Our architecture for matching text pairs

The architecture of our model for matching query-document pairs is presented in Fig. 7.2. Our sentence models based on ConvNets learn to map input sentences to vectors, which can then be used to compute their similarity. These are then used to compute a query-document similarity score, which together with the query and document vectors are joined in a single representation.

In the following we describe how the intermediate representations produced by the sentence model can be used to compute query-document similarity scores and give a brief explanation of the remaining layers, e.g. hidden and softmax, used in our network.

7.4.4 Matching query and documents

Given the output of our sentence ConvNets for processing queries and documents, their resulting vector representations $\mathbf{x}_q$ and $\mathbf{x}_d$, can be used to compute a query-document similarity score. We follow the approach of [8] that defines the similarity between $\mathbf{x}_q$ and $\mathbf{x}_d$ vectors as follows:

$$sim(\mathbf{x}_q, \mathbf{x}_d) = \mathbf{x}_q^T \mathbf{M} \mathbf{x}_d, \tag{7.4}$$

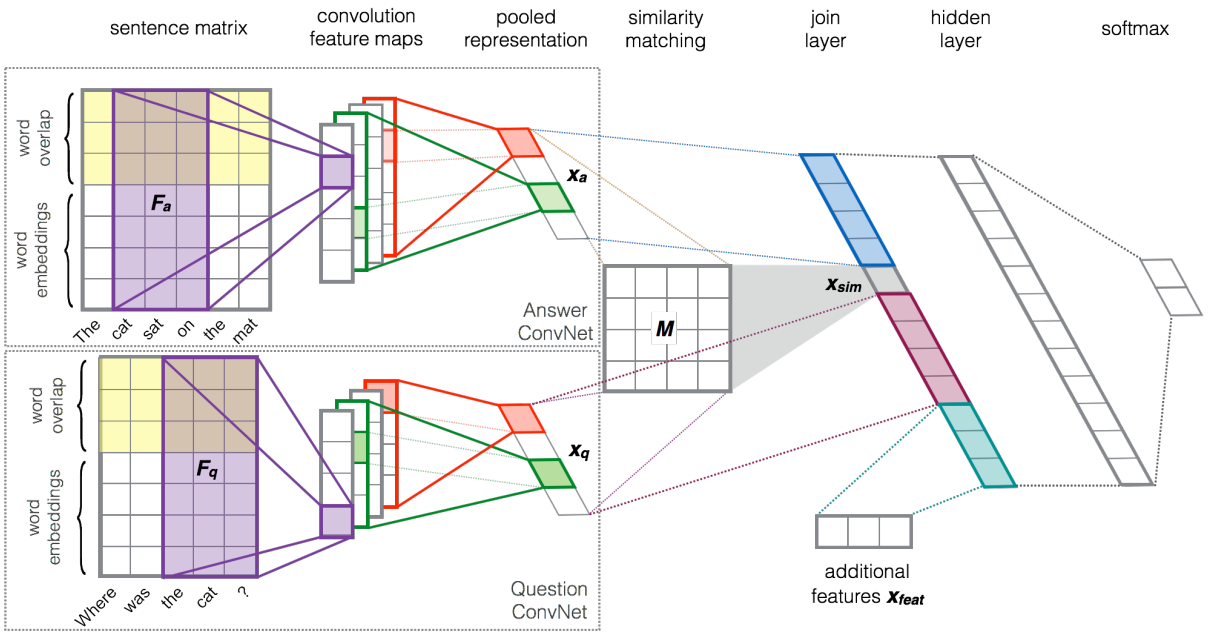


Figure 7.2: Our deep learning architecture for reranking short text pairs.

where $\mathbf{M} \in \mathbb{R}^{d \times d}$ is a similarity matrix. The Eq. 7.4 can be viewed as a model of the noisy channel approach from machine translation, which has been widely used as a scoring model in information retrieval and question answering [42]. In this model, we seek a transformation of the candidate document $\mathbf{x}'_d = \mathbf{M}\mathbf{x}_d$ that is the closest to the input query $\mathbf{x}_q$. The similarity matrix $\mathbf{M}$ is a parameter of the network and is optimized during the training.

7.4.5 The information flow

Here we provide a full description of our deep learning network (shown on Fig. 7.2) that maps input sentences to class probabilities.

The output of our sentence models are distributional representations of a query $\mathbf{x}_q$ and a document $\mathbf{x}_d$. These are then matched using a similarity matrix $\mathbf{M}$ according to Eq. 7.4. This produces a single score x_{sim} capturing various aspects of similarity (syntactic and semantic) between the input queries and documents. Note that it is also straight-forward to add additional features $\mathbf{x}_{\text{feat}}$ to the model.

The join layer concatenates all intermediate vectors, the similarity score and any additional features into a single vector:

$$\mathbf{x}_{\text{join}} = [\mathbf{x}_q^T; x_{\text{sim}}; \mathbf{x}_d^T; \mathbf{x}_{\text{feat}}^T]$$

This vector is then passed through a fully connected hidden layer, which allows for mod-

elling interactions between the components of the joined representation vector. Finally, the output of the hidden layer is further fed to the softmax classification layer, which generates a distribution over the class labels.

7.5 Experiments

This section reports on our experiments with deep learning models to represent input texts and text pairs. In particular, we evaluate our deep learning sentiment classifier on a recent benchmark from Semeval-2015. We test our deep learning architecture for modelling text pairs on two popular retrieval benchmarks from TREC: answer sentence selection and TREC microblog retrieval.

7.5.1 Twitter Sentiment Analysis

This section reports on the results of testing our ConvNet for twitter sentiment analysis on a recent benchmark from Semeval-2015 [129].

Experimental setup

We test our model on two subtasks from Semeval-2015 Task 10: phrase-level (subtask A) and message-level (subtask B). The datasets use in Semeval-2015 are summarized in Table 7.1. We use train and dev from Twitter'13 for training and Twitter'13-test as a validation set. The other datasets are used for testing, whereas Twitter'15 is used to establish the official ranking of the systems.

Additionally, to pre-train the weights of our network, we use a large unsupervised corpus containing 50M tweets for training the word embeddings and a 10M tweet corpus for distant supervision. The latter corpus was built similarly to [52], where tweets with positive emoticons, like ':)', are assumed to be positive, and tweets with negative emoticons, like ':(', are labeled as negative. The dataset contains equal number of positive and negative tweets.

The parameters of our model were (chosen on the validation set) as follows: the width m of the convolution filters is set to 5 and the number of convolutional feature maps is 300. We use ReLU activation function and a simple max-pooling. The dimensionality of the word embeddings d is set to 100. For the phrase-level subtask the size of the word type embeddings, which encode tokens that span the target phrase or not. is set to 10.

Pre-training the network

To train our deep learning model we follow our 3-step process as described in Sec. 7.3.3. We report the results for training the network on the official supervised dataset from Semeval'15 using parameters that were initialized: (i) completely at random (`Random`); (ii)

Table 7.1: Semeval-2015 data

Dataset	Subtask A	Subtask B
Twitter'13-train	5,895	9,728
Twitter'13-dev	648	1,654
Twitter'13-test	2,734	3,813
LiveJournal'14	660	1,142
SMS'13	1,071	2,093
Twitter'14	1,807	1,853
Sarcasm'14	82	86
Twitter'15	3,092	2,390
# Teams	11	40

Table 7.2: Testing the model on the progress test sets from Semeval-2015 with different parameter initializion schemes: `Random` (random word embeddings); `Unsup` (word2vec embeddings); `Distant` (all parameters from a network trained on a distant supervised dataset).

Dataset	Random	Unsup	Distant
LiveJournal'14	63.58	73.09	72.48
SMS'13	58.41	65.21	68.37
Twitter'13	64.51	72.35	72.79
Twitter'14	63.69	71.07	73.60
Sarcasm'14	46.10	52.56	55.44

using word embeddings from the neural language model trained on a large unsupervised dataset (`Unsup`) with the `word2vec` tool and (iii) initializing all the parameters of our model with the parameters of the network which uses the word embeddings from the previous step and are further tuned on a distant supervised dataset (`Distant`).

Table 7.2 summarizes the performance of our model on five test sets using three parameter initialization schemas. First, we observe that training the network with all parameters initialized completely at random results in a rather mediocre performance. This is due to a small size of the training set. Secondly, using embeddings pre-trained by a neural language model considerably boosts the performance. Finally, using a large distant supervised corpus to further tune the word embeddings to also capture the sentiment aspect of the words they represent results in a further improvement across all test sets (except for a small drop on LiveJournal'14).

Official rankings

The results from the official rankings for both subtasks A and B are summarized in Table 7.3. As we can see our system performs particularly strong on subtask A ranking 1st on the official Twitter'15 set, while also showing excellent performance on all other

Table 7.3: Results on Semeval-2015 for phrase and tweet-level subtasks. Rank shows absolute position of our system in the final ranking on each test set. AveRank is the averaged rank across all test sets, where Rank shows the absolute standing of a system according to this metric.

Dataset	Score	Rank
Phrase-level subtask A		
LJournal'14	84.46	2
SMS'13	88.60	2
Twitter'13	90.10	1
Twitter'14	87.12	1
Sarcasm'14	73.65	5
Twitter'15	**84.79**	**1**
AveRank	2.0	1
Message-level subtask B		
LJournal'14	72.48	12
SMS'13	68.37	2
Twitter'13	72.79	3
Twitter'14	73.60	2
Sarcasm'14	55.44	5
Twitter'15	**64.59**	**2**
AveRank	4.3	1

test sets.

On subtask B our system ranks 2nd also showing strong results on the other test sets (apart from the LiveJournal'14). In fact, no single system at Semeval-2015 performed equally well across all test sets. For example, a system that ranked 1st on the official Twitter'15 dataset performs much worse on the progress test sets ranking $\{14, 14, 11, 7, 12\}$ on {`LiveJournal'14`, `SMS'13`, `Twitter'13`, `Twitter'14`, and `Sarcasm'14`} correspondingly. It has an AveRank of 9.8, which is only 6th best result if systems were ranked according to this metric. In contrast, our system shows robust results across all tests having the best AveRank of 4.3 among all 40 submissions.

7.5.2 Answer Sentence Selection

Our first experiment where we deal with text pairs is on answer sentence selection dataset, where answer candidates are limited to a single sentence. Given a question with its list of candidate answers the task is to rank the candidate answers based on their relatedness to the question.

Experimental setup

This section describes the dataset and our experimental setup, also giving details about how we obtain the word embeddings matrix $\mathbf{W}$ and train our network.

Table 7.4: Summary of TREC QA datasets for answer reranking.

Data	# Questions	# QA pairs	% Correct
TRAIN-ALL	1,229	53,417	12.0%
TRAIN	94	4,718	7.4%
DEV	82	1,148	19.3%
TEST	100	1,517	18.7%

Data. We test our model on the manually curated TREC QA dataset[2] from Wang et al. [175], which appears to be one of the most widely used benchmarks for answer sentence reranking. The dataset contains a set of factoid questions, where candidate answers are limited to a single sentence. The set of questions are collected from TREC QA tracks 8-13. The manual judgement of candidate answer sentences is provided for the entire TREC 13 set and for the first 100 questions from TREC 8-12. The motivation behind this annotation effort is that TREC provides only the answer patterns to identify if a given passage contains a correct answer key or not. This results in many unrelated candidate answers marked as correct simply because regular expressions cannot always match the correct answer keys.

To enable direct comparison with the previous work, we use the same `train`, `dev` and `test` sets. Table 7.4 summarizes the datasets used in our experiments. An additional training set TRAIN-ALL provided by Wang et. al [175] contains 1,229 questions from the entire TREC 8-12 collection and comes with automatic judgements. This set represents a more noisy setting, nevertheless, it provides many more QA pairs for learning.

Word overlap features. In contrast to the word embeddings matrix, the size of the vocabulary V_o to encode word overlap features is tiny. Given such a small parameter space it is possible to tune the vectors even on small sized datasets. Hence, we keep this as a parameter optimized by our network. We randomly initialize the entries of the matrix $\mathbf{W}_o$ by sampling from the uniform distribution $U[-0.25, 0.25]$.

Evaluation. The two metrics used to evaluate the quality of our model are Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR). We use the official `trec_eval` scorer to compute the above metrics.

Word embeddings

While our model allows for learning the word embeddings directly, we keep the word matrix parameter $\mathbf{W}$ static. This is due to a common experience that a minimal size of the dataset required for tuning the word embeddings for a given task should be at least in the order of hundred thousands, while in our case the number of question-answer

[2] http://cs.stanford.edu/people/mengqiu/data/qg-emnlp07-data.tgz

pairs is one order of magnitude smaller. Hence, similar to [38, 76, 189] we keep the word embeddings fixed and initialize the word matrix $\mathbf{W}$ from an unsupervised neural language model.

We run `word2vec` tool [87] on the English Wikipedia dump and the AQUAINT corpus[3] containing roughly 375 million words. We opt for a skipgram model with window size 5 and filtering words with frequency less than 5. We set the dimensionality of our word embeddings to 50 to be on the line with [189]. The resulting model contains 50-dimensional vectors for about 3.5 million words. Embeddings for words not present in the `word2vec` model are randomly initialized with each component sampled from the uniform distribution $U[-0.25, 0.25]$.

We minimally preprocess the data only performing tokenization and lowercasing all words. To reduce the size of the resulting vocabulary V, we also replace all digits with 0. The size of the word vocabulary V for experiments using TRAIN set is 17,023 with approximately 95% of words initialized using `wor2vec` embeddings and the remaining 5% words are initialized at random. For the TRAIN-ALL setting the $|V| = 56,953$ with about 90% words found in the `word2vec` model.

Training and hyperparameters

The parameters of our deep learning model were (chosen on a `dev` set) as follows: the width m of the convolution filters is set to 5 and the number of convolutional feature maps is 100. We use ReLU activation function and a simple max-pooling. The size of the hidden layer is equal to the size of the $\mathbf{x}_{\text{join}}$ vector obtained after concatenating question and answer sentence vectors from the distributional models, similarity score and additional features (if used).

To train the network we use stochastic gradient descent with shuffled mini-batches. We eliminate the need to tune the learning rate by using the Adadelta update rule [196]. The batch size is set to 50 examples. The network is trained for 25 epochs with early stopping, i.e., we stop the training if no update to the best accuracy on the `dev` set has been made for the last 5 epochs. The accuracy computed on the `dev` set is the MAP score. At test time we use the parameters of the network that were obtained with the best MAP score on the `dev` set, i.e., we compute the MAP score after each 10 mini-batch updates and save the network parameters if a new best `dev` MAP score was obtained. In practice, the training converges after a few epochs. We set a value for L2 regularization term to $1e-5$ for the parameters of convolutional layers and $1e-4$ for all the others.

[3] https://catalog.ldc.upenn.edu/LDC2002T31

Size of the model

Given that the dimensionality of the word embeddings is fixed to 50, the number of parameters in the convolution layer of each sentence model is $100 \times 5 \times 50$. Hence, the total number of parameters in each of the two convolutional networks that map sentences to vectors is 25k. The similarity matrix is $\boldsymbol{M} \in \mathbb{R}^{100\times100}$, which adds another 10k parameters to the model. The fully connected hidden layer is and a softmax add about 40k parameters. Hence the total number of parameters in the network is about 100k, which is a rather compact model.

Results and discussion

We compare our model to the results from a recent deep learning system in [189] that has established the new state-of-the-art results in the same setting. Our goal is to evaluate the impact of using: (i) a more powerful convolutional network for sentence modelling; (ii) distributional representations of questions and answers in addition to the similarity score; and (iii) our approach to model overlapping words by augmenting word embeddings with additional dimensions vs. providing the network with a pre-computed feature vector of overlapping word counts as in [189].

Distributional sentence models. Table 7.5 summarises the results for the setting when the network is trained using only input question-answer pairs without using any additional features, i.e., we omit the word overlap features.

First, we report the results of our model when using only a similarity score x_{sim}. Our model shows stronger performance then the system of Yu et al. [189]. Their deep learning model, similarly to ours, relies on a convolutional neural network to learn intermediate representations. However, their convolutional neural network operates only on unigram or bigrams, while in our architecture we use a larger width of the convolution filter, thus allowing for capturing longer range dependencies.

Additionally, along with the question-answer similarity score from Eq. 7.4, our architecture includes intermediate representations of the question and the answer $\mathbf{x}_q$ and $\mathbf{x}_a$ into the final vector representation $\mathbf{x}_{\text{join}}$, which together constitute a much richer representation for computing the final score. Our representation results in a large improvement of about 6% absolute points in MAP for TRAIN and 10% when trained with more data from TRAIN-ALL. This emphasizes the importance of learning high quality sentence models.

Relational modelling via word overlap features. Yu et al. [189] shows that combining the output of their deep learning system with a simple feature vector that includes word overlap counts in a logistic regression model, provides a significant boost in accuracy and yields new state-of-the-art results.

Table 7.5: Results on TRAIN and TRAIN-ALL from TREC QA using only similarity score and also including the distributional representation of question and answer sentences.

Model	MAP	MRR
TRAIN		
Yu et al., 2014 (unigram)	.5387	.6284
Yu et al., 2014 (bigram)	.5476	.6437
Our model (only sim score)	.5884	.6036
Our model	**.6258**	**.6591**
TRAIN-ALL		
Yu et al., 2014 (unigram)	.5470	.6329
Yu et al., 2014 (bigram)	.5693	.6613
Our model (only sim score)	.6521	.7010
Our model	**.6709**	**.7280**

Table 7.6 provides the results when we include the information about overlapping words in two modes: (i) *feature vector* (fvec) mode – when we include overlapping word counts replicating [189], which is represented by a feature vector $\mathbf{x}_{\text{feat}}$ that is plugged into the final representation $\mathbf{x}_{\text{join}}$ (see Fig. 7.2); and (ii) *embeddings* mode – when we augment the representation of input words with additional word *overlap* indicator features. First, we note that the results are significantly better than in Table 7.5 when no overlap information is used. Adding word overlap information in the form of a feature vector $\mathbf{x}_{\text{feat}}$ results in a considerable generalization improvement of the network. As argued by Yu et al. [189], distributional word embeddings have certain shortcomings especially when dealing with proper nouns and cardinal numbers, which are frequent in factoid questions.

In contrast, our approach to encode the relational information about overlapping words in a pair (*embeddings*) directly into word embeddings shows even larger improvement on TRAIN-ALL with a MAP score of 76.42% and 81.26% MRR and comparable results on TRAIN. We conjecture that small size of the TRAIN set may negatively impact the quality of the embeddings $\mathbf{W}_o$ for the word overlap features that are learned on a smaller set. Nevertheless, training our model on TRAIN-ALL, which contains an order of magnitude more pairs, results in a 5% absolute improvement in MAP and 3% in MRR, which produces the best MAP and MRR scores.

Comparing to previous state-of-the-art. Finally, we compare our results with the previously published systems on this task (Table 7.7), where our system sets new state-of-the-art scores with a large margin. The results are very promising considering that our system requires no manual feature engineering (other than simple word overlap features), no expensive preprocessing using various NLP parsers, and no external semantic resources other than using pre-initialized word embeddings that can be easily trained provided a large amount of unsupervised text.

Table 7.6: Results on TREC QA when augmenting the deep learning model with relational information about overlapping words.

Model	MAP	MRR
TRAIN		
Yu et al., 2014 (unigram)	.6889	.7727
Yu et al., 2014 (bigram)	.7058	**.7800**
Our model (fvec)	**.7275**	.7796
Our model (embeddings)	.7207	.7741
TRAIN-ALL		
Yu et al., 2014 (unigram)	.6934	.7677
Yu et al., 2014 (bigram)	.7113	.7846
Our model (fvec)	.7459	.8078
Our model (embeddings)	**.7642**	**.8126**

Table 7.7: Survey of the results on the TREC QA answer selection task.

Model	MAP	MRR
Wang et al. (2007)	.6029	.6852
Heilman and Smith (2010)	.6091	.6917
Wang and Manning (2010)	.5951	.6951
Yao et al. (2013)	.6307	.7477
Severyn & Moschitti (2013)	.6781	.7358
Yih et al. (2013)	.7092	.7700
Yu et al. (2014)	.7113	.7846
Our model (TRAIN-ALL)	**.7642**	**.8126**

In the spirit, our system is most similar to a recent deep learning architecture from Yu et al. (2014) [189]. However, we employ a more expressive convolutional neural network for learning intermediate representations of the query and the answer. This allows for performing a more accurate matching between question-answer pairs. Additionally, our architecture includes intermediate question and answer representations in the model, which result in a richer representation of question-answer pairs. Finally, we train our system in an end-to-end fashion, while [189] use the output of their deep learning system as a feature in a logistic regression classifier.

7.5.3 TREC Microblog Retrieval

To assess the effectiveness and generality of our deep learning model for text matching, we apply it on tweet reranking task. We focus on the 2011 and 2012 editions of the ad-hoc retrieval task at TREC microblog tracks [103, 149]. We follow the setup in [135], where they represent query-tweet pairs with a shallow syntactic models to learn a tree kernel reranker. In contrast, our model does not rely on any syntactic parsers and requires virtually no preprocessing other than tokenizaiton and lower-casing. Our main research

Table 7.8: Summary of TREC Microblog datasets.

Data	# Topic	# Tweet pairs	% Correct	# Runs
TMB2011	49	60,129	5.1%	184
TMB2012	59	73,073	8.6%	120

question is: Can our neural network that requires no manual feature engineering and expensive pre-processing steps improve on top of the state-of-the-art learning-to-rank and retrieval algorithms?

To answer this question, we test our model in the following settings: we treat the participant systems in the TREC microblog tasks as a black-box, and implement our model on top of them using only their raw scores (ranks) as a single feature in our model. This allows us to see whether our model is able to learn information complementary to the approaches used by such retrieval algorithms. Our setup replicates the experiments in [135] to allow for comparing to their model.

Experimental setup

Data and setup. Our dataset is the tweet corpus used in both TREC Microblog tracks in 2011 (TMB2011) and 2012 (TMB2012). It consists of 16M tweets spread over two weeks, and a set of 49 (TMB2011) and 59 (TMB2012) timestamped topics. We minimally preprocess the tweets—we normalize elongations (e.g., sooo → so), normalize URLs and author ids. Additionally, we use the system runs submitted at TMB2011 and TMB2012, which contain 184 and 120 models, respectively. This is summarized in Table 7.8.

Word embeddings. We used the `word2vec` tool to learn the word embeddings from the provided 16M tweet corpus, with the following setting: (i) we removed non-english tweets, which reduces the corpus to 8.4M tweets and (ii) we used the skipgram model with window size 5 and filtering words with frequency less than 5. The trained model contains 330k words. We use word embeddings of size 50 — same as for the previous task. To build the word embedding matrix $\mathbf{W}$, we extract the vocabulary from all tweets present in TMB2011 and TMB2012. The resulting vocabulary contains 150k words out of which only 60% are found in the word embeddings model. This is due to a very large number of misspellings and words occurring only once (hence they are filted by the `word2vec` tool). This has a negative impact on the performance of our deep learning model since around 40% of the word vectors are randomly initialized. At the same time it is not possible to tune the word embeddings on the training set, as it will overfit due to the small number of the query-tweet pairs available for training. For the same reasons as in our experiments on answer sentence selection we keep the word embedding matrix parameter fixed during the training.

Training. We train our system on the runs submitted at TMB2011, and test it on the TMB2012 runs. We focus on one direction only to avoid training bias, since TMB2011 topics were already used for learning systems in TMB2012.

Submission run as a feature. We use the output of participant systems as follows: we use rank positions of each tweet rather than raw scores, since scores for each system are scaled differently, while ranks are uniform across systems. We apply the following transformation of the rank r: $1/\log(r+1)$. In the training phase, we take the top 30 systems from the TMB2011 track (in terms of P@30). For each query-tweet pair we compute the average transformed rank over the top systems. This score is then used as a single feature $\mathbf{x}_{\text{feat}}$ by our model. In the testing phase, we generate this feature as follows: for each participant system that we want to improve, we use the transformed rank of the query-tweet taken from their submission run.

Evaluation. We report on the official evaluation metric for the TREC 2012 Microblog track, i.e., precision at 30 (P@30), and also on mean average precision (MAP). Following [16, 103], we regard minimally and highly relevant documents as relevant and use the TMB2012 evaluation script. For significance testing, we use a pairwise t-test, where $\triangle$ and $\blacktriangle$ denote significance at $\alpha = 0.05$ and $\alpha = 0.01$, respectively. Triangles point up for improvement over the baseline, and down otherwise. We also report the improvement in the absolute rank (R) in the official TMB2012 ranking.

Results and discussion

Table 7.9 reports the results for re-ranking runs of the best 30 systems from TMB2012 (based on their P@30 score) when we train our system using the top 30 runs from TMB2011.

First, we note that our model improves P@30 for the majority of the systems with a relative improvement ranging from several points up to 10% with about 6% on average. This is remarkable, given that the pool of participants in TMB2012 was large, and the top systems are therefore likely to be very strong baselines.

Secondly, we note that the relative improvement of our model is on the par with the STRUCT model from [135], which relies on using syntactic parsers to train a tree kernel reranker. In contrast, our model requires no manual feature engineering and virtually no preprocessing and external resources. Similar to the observation made in [135], our model has a precision-enhancing effect. In cases where MAP drops a bit it can be seen that our model sometimes lowers relevant documents in the runs. It is possible that our model favours query-tweet pairs that exhibit semantic matching of higher quality, and that it down-ranks tweets that are of lower quality but are nonetheless relevant. Another important aspect is the fact that a large portion of the word embeddings (about 40%) used

Table 7.9: System performance on the top 30 runs from TMB2012, using the top 10, 20 or 30 runs from TMB2011 for training.

#	runs	TMB2012		STRUCT [135]			Our model		
		MAP	P@30	MAP	P@30	R%	MAP	P@30	R%
1	hitURLrun3	0.3469	0.4695	0.3328 (-4.1%)$^{\nabla}$	0.4774 (1.7%)	0	0.3326 (-4.1%)$^{\nabla}$	0.4836 (3.0%)	0
2	kobeMHC2	0.3070	0.4689	0.3037 (-1.1%)	0.4768 (1.7%)	1	0.3052 (-0.6%)	0.4899 (4.5%)$^{\triangle}$	1
3	kobeMHC	0.2986	0.4616	0.2965 (-0.7%)	0.4718 (2.2%)	2	0.2999 (0.4%)	0.4830 (4.6%)$^{\triangle}$	2
4	uwatgclrman	0.2836	0.4571	0.2995 (5.6%)$^{\blacktriangle}$	0.4712 (3.1%)$^{\triangle}$	3	0.2738 (-3.5%)$^{\nabla}$	0.4516 (-1.2%)	-1
5	kobeL2R	0.2767	0.4429	0.2744 (-0.8%)	0.4463 (0.8%)	0	0.2677 (-3.3%)$^{\nabla}$	0.4409 (-0.5%)	-2
6	hitQryFBrun4	0.3186	0.4424	0.3118 (-2.1%)	0.4554 (2.9%)	2	0.3220 (1.1%)$^{\blacktriangle}$	0.4849 (9.6%)$^{\blacktriangle}$	5
7	hitLRrun1	0.3355	0.4379	0.3226 (-3.9%)$^{\nabla}$	0.4525 (3.3%)	2	0.3188 (-5.0%)$^{\nabla}$	0.4610 (5.3%)$^{\blacktriangle}$	3
8	FASILKOM01	0.2682	0.4367	0.2820 (5.2%)$^{\blacktriangle}$	0.4531 (3.8%)$^{\blacktriangle}$	3	0.2622 (-2.2%)$^{\nabla}$	0.4346 (-0.5%)	-1
9	hitDELMrun2	0.3197	0.4345	0.3105 (-2.9%)	0.4424 (1.8%)	4	0.3246 (1.5%)	0.4723 (8.7%)$^{\blacktriangle}$	8
10	tsqe	0.2843	0.4339	0.2836 (-0.3%)	0.4441 (2.4%)	5	0.2917 (2.6%)	0.4660 (7.4%)$^{\blacktriangle}$	7
11	ICTWDSERUN1	0.2715	0.4299	0.2862 (5.4%)$^{\blacktriangle}$	0.4582 (6.6%)$^{\blacktriangle}$	7	0.2765 (1.8%)$^{\blacktriangle}$	0.4484 (4.3%)$^{\triangle}$	6
12	ICTWDSERUN2	0.2671	0.4266	0.2785 (4.3%)$^{\triangle}$	0.4475 (4.9%)$^{\blacktriangle}$	7	0.2786 (4.3%)$^{\triangle}$	0.4478 (5.0%)$^{\triangle}$	7
13	cmuPrfPhrE	0.3179	0.4254	0.3172 (-0.2%)	0.4469 (5.1%)$^{\blacktriangle}$	8	0.3321 (4.5%)$^{\triangle}$	0.4585 (7.8%)$^{\blacktriangle}$	9
14	cmuPrfPhrENo	0.3198	0.4249	0.3179 (-0.6%)	0.4486 (5.6%)$^{\blacktriangle}$	9	0.3359 (5.0%)$^{\triangle}$	0.4591 (8.1%)$^{\blacktriangle}$	10
15	cmuPrfPhr	0.3167	0.4198	0.3130 (-1.2%)	0.4379 (4.3%)$^{\triangle}$	8	0.3282 (3.6%)$^{\triangle}$	0.4572 (8.9%)$^{\blacktriangle}$	11
16	FASILKOM02	0.2454	0.4141	0.2718 (10.8%)$^{\blacktriangle}$	0.4508 (8.9%)$^{\blacktriangle}$	11	0.2489 (1.4%)	0.4201 (1.5%)$^{\triangle}$	1
17	IBMLTR	0.2630	0.4136	0.2734 (4.0%)$^{\triangle}$	0.4441 (7.4%)$^{\blacktriangle}$	10	0.2703 (2.8%)$^{\triangle}$	0.4346 (5.1%)$^{\blacktriangle}$	8
18	otM12ihe	0.2995	0.4124	0.2969 (-0.9%)	0.4322 (4.8%)$^{\blacktriangle}$	7	0.2900 (-3.2%)$^{\nabla}$	0.4239 (2.8%)$^{\triangle}$	3
19	FASILKOM03	0.2716	0.4124	0.2859 (5.3%)$^{\blacktriangle}$	0.4452 (8.0%)$^{\blacktriangle}$	14	0.2740 (0.9%)	0.4270 (3.5%)$^{\blacktriangle}$	7
20	FASILKOM04	0.2461	0.4113	0.2575 (4.6%)$^{\blacktriangle}$	0.4294 (4.4%)$^{\blacktriangle}$	9	0.2414 (-1.9%)$^{\nabla}$	0.4220 (2.6%)$^{\triangle}$	5
21	IBMLTRFuture	0.2731	0.4090	0.2808 (2.8%)	0.4311 (5.4%)$^{\blacktriangle}$	10	0.2785 (2.0%)$^{\triangle}$	0.4415 (8.0%)$^{\blacktriangle}$	14
22	uiucGSLIS01	0.2445	0.4073	0.2575 (5.3%)$^{\blacktriangle}$	0.4260 (4.6%)$^{\blacktriangle}$	9	0.2478 (1.4%)	0.4233 (3.9%)$^{\triangle}$	7
23	PKUICST4	0.2786	0.4062	0.2909 (4.4%)$^{\triangle}$	0.4514 (11.1%)$^{\blacktriangle}$	18	0.2832 (1.7%)$^{\blacktriangle}$	0.4491 (10.6%)$^{\blacktriangle}$	18
24	uogTrLsE	0.2909	0.4028	0.2977 (2.3%)	0.4282 (6.3%)$^{\blacktriangle}$	9	0.3131 (7.6%)$^{\blacktriangle}$	0.4484 (11.3%)$^{\blacktriangle}$	19
25	otM12ih	0.2777	0.3989	0.2810 (1.2%)	0.4175 (4.7%)$^{\blacktriangle}$	10	0.2752 (-0.9%)	0.4119 (3.3%)$^{\triangle}$	5
26	ICTWDSERUN4	0.1877	0.3887	0.1985 (5.8%)$^{\blacktriangle}$	0.4164 (7.1%)$^{\blacktriangle}$	10	0.2040 (8.7%)$^{\blacktriangle}$	0.4220 (8.6%)$^{\blacktriangle}$	11
27	uwatrrfall	0.2620	0.3881	0.2812 (7.3%)$^{\blacktriangle}$	0.4136 (6.6%)$^{\blacktriangle}$	9	0.2942 (12.3%)$^{\triangle}$	0.4314 (11.2%)$^{\blacktriangle}$	16
28	cmuPhrE	0.2731	0.3842	0.2797 (2.4%)	0.4136 (7.7%)$^{\blacktriangle}$	12	0.2972 (8.8%)$^{\triangle}$	0.4352 (13.3%)$^{\blacktriangle}$	19
29	AIrun1	0.2237	0.3842	0.2339 (4.6%)$^{\blacktriangle}$	0.4102 (6.8%)$^{\blacktriangle}$	5	0.2285 (2.2%)$^{\triangle}$	0.4157 (8.2%)$^{\blacktriangle}$	13
30	PKUICST3	0.2118	0.3825	0.2318 (9.4%)$^{\blacktriangle}$	0.4119 (7.7%)$^{\blacktriangle}$	14	0.2363 (11.6%)$^{\blacktriangle}$	0.4415 (15.4%)$^{\blacktriangle}$	23
	Average			3.3%	5.3%	7.3	2.0%	6.2%	7.8

by the network are initialized at random, which has a negative impact on the accuracy of our model.

Looking at the improvement in absolute position in the official ranking (R), we see that, on average, our deep learning model boosts the absolute position in the official ranking for top 30 systems by about 7.8 positions.

All in all, the results suggest that our deep learning model with no changes in its architecture is able to capture additional information and can be useful when coupled with many state-of-the-art microblog search algorithms.

While improving the top systems from 2012 represents a challenging task, it is also interesting to assess the potential improvement for systems that ranked lower. We follow [135] and report our results on the 30 systems from the middle and the bottom of the official ranking. Table 7.10 summarizes the average improvements for three groups of 30

Table 7.10: Comparison of the averaged relative improvements for the top, middle (mid), and bottom (btm) 30 systems from TMB2012.

	STRUCT [135]		Our model	
band	MAP	P@30	MAP	P@30
top	3.3%	5.3%	2.0%	6.2%
mid	12.2%	12.9%	9.8%	13.7%
btm	22.1%	25.1%	18.7%	24.3%

systems each: top-30, middle-30, and bottom-30.

We find that the improvement over underperforming systems is much larger than for stronger systems. In particular, for the bottom 30 systems, our approach achieves an average relative improvement of 20% in both MAP and P@30. The performance of our model is on the par with the STRUCT model [135].

We expect that learning word embeddings on a larger corpora such that the percentage of the words present in the word embedding matrix $\mathbf{W}$ should help to improve the accuracy of our system. Moreover, similar to the situation observed with answer selection experiments, we expect that using more training data would improve the generalization of our model. As one possible solution to getting more training data, it could be interesting to experiment with training our model on much larger pseudo test collections similar to the ones proposed in [16]. We leave it for the future work.

7.6 Related Work

Most of the previous work to tackle the answer sentence selection task use various approaches to model transformations of syntactic trees between a question and its candidate answer sentence, e.g., Wang et al. [175] use quasi-synchronous grammar, Heilman & Smith [58] develop an improved Tree Edit Distance (TED) model, Wang & Manning [174] develop a probabilistic model to learn tree-edit operations on dependency parse trees, while Yao et al. [186] applies linear chain CRFs with features derived from TED. Severyn and Moschitti [134] applied SVM with tree kernels to shallow syntactic representations. Yih et al. [161] use distributional models based on lexical semantics to match semantic relations of aligned words in QA pairs.

Deep learning methods generalize the idea of distributional word vectors by providing the possibility to learn the compositional rules directly from the data. Recently, [15] shown that the word vectors learned by unsupervised neural language models are superior to the count-based distributional semantic models. Convolutional neural networks that are at the core of many deep learning approaches have been successfully applied to various sentence classification tasks, e.g., [72, 76], and for modelling text pairs, e.g. [60, 84].

The work closest to ours is [189], where they apply deep learning to learn to match

question and answer sentences. However, their sentence model to map questions and answers to vectors operates only on unigrams or bigrams. Our sentence model is based on a convolutional neural network with the state-of-the-art architecture, we use a larger width of the convolution filter, thus allowing the network to capture longer range dependencies. Moreover, our architecture along with the similarity score also encodes vector representations of questions and answers used to compute the final score. Hence, our model constructs and learns a richer representation of the question-answer pairs, which results in superior results on the answer sentence selection dataset. Moreover, we use a completely different way to encode relational information about words that overlap in a pair. Finally, our deep learning model is trained end-to-end, while in [189] they use the output of their neural network in a separate logistic scoring model.

Regarding learning to rank systems applied to TREC microblog datasets, recently [135] have shown that richer linguistic representations of tweets can improve upon state of the art systems in TMB-2011 and TMB-2012. We directly compare with their system, showing that our deep learning model without any changes to its architecture (we only pre-train word embeddings) is on the par with their reranker. This is remarkable, since different from [135], which requires additional pre-proccesing using syntactic parsers to construct syntactic trees, our model requires no expensive pre-processing and does not rely on any external resources.

7.7 Summary

We described our deep learning approach to sentiment analysis of tweets for predicting polarities at both message and phrase levels. We give a detailed description of our 3-step process to train the parameters of the network that is the key to our success. The resulting model sets a new state-of-the-art on the phrase-level and is 2nd on the message-level subtask. Considering the average rank across all test sets our system is 1st on both subtasks.

Our network initialization process includes the use of distant supervised data (noisy labels are inferred using emoticons found in the tweets) to further refine the weights of the network passed from the completely unsupervised neural language model. Thus, our solution successfully combines together two traditionally important aspects of IR: unsupervised learning of text representations (word embeddings from neural language model) and learning on weakly supervised data.

We also propose a novel deep learning architecture for modelling text pairs. It has the benefits of requiring no manual feature engineering or external resources, which may be expensive or not available. The model with the same architecture can be successfully applied to other domains and tasks.

Our experimental findings show that our deep learning model: (i) greatly improves

on the previous state-of-the-art systems and a recent deep learning approach in [189] on answer sentence selection task showing a 5% absolute improvement in MAP and 3% MRR; (ii) our system is able to improve even the best system runs from TREC Microblog 2012 challenge; (iii) is comparable to the syntactic reranker in [135], while our system requires no external parsers or resources.

This is largely due to our use of more expressive models for the input question and answer sentences, and our approach to inject relational information directly in the word embeddings. However, our word overlap indicator features are based on simple string matching, which is clearly a very coarse way to model relatedness between words in a question-answer pair.

Recently, deep learning architectures have been successfully applied to learn word alignments in machine translation, e.g., [185]. It sounds promising to allow the network to learn to dynamically align the related words in a question and its answer. This, in turn, requires to maximize over the latent alignment configurations, thus making the optimization problem highly non-convex. Our preliminary experiments show that far larger number of text pairs are required to train such architectures. We leave it for the future work.

Chapter 8

Summary and Future Work

Learning with kernels poses two major problems when approaching NLP tasks: (i) the training time scales quadratically in the number of input examples; and (ii) plain syntactic structures produced by running off-the-shelf syntactic parsers often lacks the representation power required to build state-of-the-art models.

In this thesis we address these problems by: first, proposing novel algorithms that greatly speed up training of SVMs with structural kernels (Chapter 3), and secondly, demonstrate the effectiveness of modelling the textual inputs using carefully designed syntactic/semantic structures (Chapter 4, 5, and 6) where tree kernel functions play the role of automatically generating rich feature spaces, thus greatly reducing the manual feature engineering effort. An important aspect of our approach is that it can effectively combine together different types of syntactic and semantic information, also generated by additional automatic classifiers, e.g., focus and question classifiers. We augment the basic structures with additional relational and semantic information by introducing special tag markers into the tree nodes.

Finally, in the last Chapter 7 we explore the deep learning approaches that address the problem of modelling words (previously treated as atomic units) by representing them as vectors. At the same time deep neural networks make it possible to automatically learn the compositional rules to combine the word vectors into a vector representation for word phrases and sentences. We show that some of our findings applied to model text pairs using syntactic structural representations (explored in Chapters 5, 6) can be efficiently encoded into the deep learning architectures. We augment the word embeddings with additional dimensions to encode the information about word overlapping in a pair, which results in a considerably more accurate models producing state-of-the-art results.

We show the effectiveness of our approach on a large number of tasks: starting from our large-scale experiments on Semantic Role Labeling to train boundary detection classifiers to modelling texts with more advances syntactic/semantic structures to encode Youtube comments for building sentiment classifiers, modelling text pairs for Semantic Textual

Similarity, Microblog retrieval and answer passage reranking, answer sentence selection and answer key extraction in Question Answering.

8.1 Fast SVMs with structural kernels

In this chapter we have presented several techniques to make learning with SVMs and convolution tree kernels applicable to a larger set of real-world applications. Firstly, we have defined a generalized theory and methods for using DAG model compression in the CPA algorithm with sampling. Our extensive experiments on several tasks provide a number of insights about the nature of the obtained speedups.

Next, we build on our idea of DAG model compression within the CPA learning algorithm to define a novel linearization approach based on the reverse kernel engineering. Our findings reveal that on a high-level semantic task such as SRL the naïve approach of enumerating all possible sub-structures becomes intractable and hashed feature vectors fail to achieve both the same accuracy of tree kernels and high efficiency. In contrast, SDAG allows for achieving the same accuracy as SVMs and makes learning practical. However, the classification and learning time may still not be appealing for large-scale experiments. Finally, linearization with RKE is rather effective as again there is almost no loss in accuracy and it benefits from extracting complex and highly discriminative features derived from learning in kernel spaces.

As a result, we derive an efficient approach to kernel learning: applying reverse-kernel engineering directly on the SDAG model. This alleviates the major computational bottleneck of the original approach, where traditional SVM training was used. Interestingly, the extracted features have high overlap with the baseline SVM. Additionally, we achieve a significant speedup with almost no loss in accuracy by splitting the data into smaller subsets. This allows for more efficient kernel space learning in a fully distributed manner.

Summing up, we can train an accurate tree kernel model on 4 million instances from SRL, in less than 20 minutes using 10 CPUs. We achieved F1 of 84.5% on Section 23, which is the state-of-the-art performance of the binary classifier for boundary detection without using ensembles of learners and relying only a single source of the syntactic information from the parse trees.

Our study opens several **future research directions**: application of tree kernels to many tasks, where large data size has prevented their use. This surely regards SRL in many languages but also parse tree re-ranking [32] and question answering applications. Also applications to other data mining tasks would be interesting, e.g., XML tree classification.

From the algorithmic perspective, it would be promising to explore approaches to prune the DAGs for achieving higher compression rates without any loss in accuracy. Finally, the ultimate goal would be to use tree kernels for structured output prediction.

8.2 Syntactic structures for modelling input texts: Youtube comments

In this chapter we defined novel structural models and a tree kernel to model noisy social media data such as YouTube comments. We carried out a systematic study on OM from YouTube comments by training a set of supervised multi-class classifiers distinguishing between video and product related opinions. We use standard feature vectors augmented by shallow syntactic trees enriched with additional conceptual information.

In this chapter we make the following contributions: (i) we show that effective OM can be carried out with supervised models trained on high quality annotations; (ii) we introduce a novel annotated corpus of YouTube comments, which we make available for the research community; (iii) we define novel structural models and kernels, which can improve over feature vectors, e.g., up to 30% of relative improvement in type classification, when little data is available, and demonstrates that the structural model scales well to other domains.

In the **future**, it seems promising to develop a joint model to classify all the comments of a given video, s.t. it is possible to exploit latent dependencies between entities and the sentiments of the comment thread. Additionally, building a hierarchical multi-label classifiers for the full task should further boost the accuracy (in place of a flat multi-class learner).

8.3 Syntactic structures for modelling text pairs

In this chapter we explore the design of linguistic tree structures to represent text pairs. This provides a rich representation for the learning algorithm to extract useful syntactic and shallow semantic patterns.

We have provided an extensive experimental study of four different structural representations for modelling the STS problem, e.g. shallow, constituency, dependency and phrase-dependency trees using STK and PTK. The novelty of our approach is that it goes beyond a simple combination of tree kernels with feature vectors as: (i) it directly encodes input text pairs into relationally linked structures; (ii) the learned structural models are used to obtain prediction scores thus making it easy to plug into existing feature-based models, e.g. via stacking; (iii) to our knowledge, this work is the first to apply structural kernels and combinations in a regression setting; and (iv) our model achieves the state of the art in STS largely improving the best previous systems. Our structural learning approach to STS is conceptually simple and does not require additional linguistic sources other than off-the-shelf syntactic parsers. It is particularly suitable for NLP tasks where the input domain comes as pairs of objects, e.g., question answering, paraphrasing and recognising textual entailment.

Regarding our model for reranking tweets, to the best of our knowledge, this work is the first to study the utility of syntactic patterns for microblog retrieval. We propose an efficient way to encode tweets into linguistic structures and use kernels for automatic feature engineering and learning. Our experimental findings show that our model: (i) improves in both MAP and P@30 when coupled with the features from a strong L2R baseline; (ii) provides a complementary source of features general enough to improve the best 30 systems from TMB2012; (iii) the performance gains are stable when we use run scores from the top 10, 20 or 30 best systems for learning; and (iv) the improvement becomes larger for under-performing systems achieving an average 20% of relative improvement in MAP and P@30 for bottom 30 systems.

As a **future work**, it would be interesting to explore linearization approaches to extract and encode some of the most discriminative syntactic patterns useful for modelling text pairs.

8.4 Semantic syntactic structures for Question Answering

In this chapter we demonstrate the effectiveness of handling the input structures representing QA pairs directly vs. using explicit feature vector representations, which typically require substantial feature engineering effort. Our approach relies on a kernel-based learning framework, where structural kernels, e.g., tree kernels, are used to handle automatic feature engineering. It is enough to specify the desired type of structures, e.g., shallow, constituency, dependency trees, representing question and its candidate answer sentences and let the kernel learning framework learn to use discriminative tree fragments for the target task.

An important feature of our approach is that it can effectively combine together different types of syntactic and semantic information, also generated by additional automatic classifiers, e.g., focus and question classifiers. We augment the basic structures with additional relational and semantic information by introducing special tag markers into the tree nodes. Using the structures directly in the kernel learning framework makes it easy to integrate additional relational constraints and semantic information directly in the structures.

The comparison with previous work on a public benchmark from TREC suggests that our approach is very promising as we can improve the state of the art in both answer selection and extraction by a large margin (up to 22% of relative improvement in F1 for answer extraction). Our approach makes it relatively easy to integrate other sources of semantic information, among which the use of Linked Open Data can be the most promising to enrich the structural representation of q/a pairs.

To achieve state-of-the-art results in answer sentence selection and answer extraction, it is sufficient to provide our model with a suitable tree structure encoding relevant syn-

tactic information, e.g., using shallow, constituency or dependency formalisms. Moreover, additional semantic and relational information can be easily plugged in by marking tree nodes with special tags. We believe this approach greatly eases the task of tedious feature engineering that will find its applications well beyond QA tasks.

Finally, the inefficacy of using topic models, WordNet, SuperSense and word alignment studied in this chapter suggests that information produced by unsupervised methods has still to be carefully considered. Therefore, studying ways to utilize semantic resources at their best is a natural **future extension** of this research line.

8.5 Deep Learning architectures for modelling input texts and text pairs

In this chapter we described our deep learning approach to sentiment analysis of tweets for predicting polarities at both message and phrase levels. We gave a detailed description of our 3-step process to train the parameters of the network that is the key to our success. The resulting model sets a new state-of-the-art on the phrase-level and is 2nd on the message-level subtask. Considering the average rank across all test sets our system is 1st on both subtasks.

Our network initialization process includes the use of distant supervised data (noisy labels are inferred using emoticons found in the tweets) to further refine the weights of the network passed from the completely unsupervised neural language model. Thus, our solution successfully combines together two traditionally important aspects of IR: unsupervised learning of text representations (word embeddings from neural language model) and learning on weakly supervised data.

We also proposed a novel deep learning architecture for modelling text pairs. It has the benefits of requiring no manual feature engineering or external resources, which may be expensive or not available. The model with the same architecture can be successfully applied to other domains and tasks.

Our experimental findings show that our deep learning model: (i) greatly improves on the previous state-of-the-art systems and a recent deep learning approach in [189] on answer sentence selection task showing a 5% absolute improvement in MAP and 3% MRR; (ii) our system is able to improve even the best system runs from TREC Microblog 2012 challenge; (iii) is comparable to the syntactic reranker in [135], while our system requires no external parsers or resources.

This is largely due to our use of more expressive models for the input question and answer sentences, and our approach to inject relational information directly in the word embeddings. However, our word overlap indicator features are based on simple string matching, which is clearly a very coarse way to model relatedness between words in a question-answer pair.

Recently, deep learning architectures have been successfully applied to learn word alignments in machine translation, e.g., [185]. It sounds promising to allow the network to learn to dynamically align the related words in a question and its answer. This in turn requires to maximize over the latent alignment configurations, thus making the optimization problem highly non-convex. Our preliminary experiments show that far larger number of text pairs are required to train such architectures. We leave it for **the future work**.

Bibliography

[1] Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, David Gondek, and Richard Lawrence. Learning to rank for robust question answering. In *CIKM*, 2012.

[2] Eneko Agirre, Daniel Cer, Mona Diab, and Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In *First Joint Conference on Lexical and Computational Semantics (*SEM)*, 2012.

[3] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In **SEM 2013: The Second Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, 2013.

[4] Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti, and Alessandro Moschitti. Fast on-line kernel learning for trees. In *ICDM*, pages 787–791, 2006.

[5] Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti, and Alessandro Moschitti. Efficient kernel-based learning for trees. In *CIDM*, pages 308–315, 2007.

[6] Elif Aktolga, James Allan, and David A. Smith. Passage reranking for question answering using syntactic structures and answer types. In *ECIR*, 2011.

[7] L Allison and T I Dix. A bit-string longest-common-subsequence algorithm. *Inf. Process. Lett.*, 23(6):305–310, December 1986.

[8] Jason Weston Antoine Bordes and Nicolas Usunier. Open question answering with weakly supervised embedding models. In *ECML*, Nancy, France, September 2014.

[9] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008.

[10] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. In *SDM*, 2002.

[11] Giuseppe Attardi, Antonio Cisternino, Francesco Formica, Maria Simi, and Ro Tommasi. Piqasso: Pisa question answering system. In *TREC*, pages 599–607, 2001.

[12] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: An analysis using six European languages. In *SIGIR*, 2007.

[13] Timothy Baldwin, Paul Cook, Marco Lui, Andrew MacKinlay, and Li Wang. How noisy social media text, how diffrnt social media sources? In *IJCNLP*, 2013.

[14] Daniel Bar, Chris Biemann, Iryna Gurevych, and Torsten Zesch. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval*, 2012.

[15] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*, 2014.

[16] Richard Berendsen, Manos Tsagkias, Wouter Weerkamp, and Maarten de Rijke. Pseudo test collections for training and tuning microblog rankers. In *SIGIR*, 2013.

[17] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1):39–71, 1996.

[18] Chris Biemann. Creating a system for lexical substitutions from scratch using crowdsourcing. *Lang. Resour. Eval.*, 47(1):97–122, March 2013.

[19] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

[20] Richard Bödi, Katrin Herr, and Michael Joswig. Algorithms for symmetric linear and integer programs. *Mathematical Programming, Series A, Online First*, Jan 2011. Comments: 21 pages, 1 figure; sums up and extends results from 0908.3329 and 0908.3331.

[21] Razvan Bunescu and Yunfeng Huang. Towards a general model of answer typing: Question focus identification. In *CICLing*, 2010.

[22] Erik Cambria and Amir Hussain. *Sentic Computing: Techniques, Tools, and Applications*. Springer, 2012.

[23] N. Cancedda, E. Gaussier, C. Goutte, and J. M. Renders. Word sequence kernels. *J. Mach. Learn. Res.*, 3:1059–1082, 2003.

[24] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA, 2007. ACM.

[25] Xavier Carreras and Lluís Màrquez. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, CONLL '05, pages 152–164, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[26] Paula Carvalho, Luís Sarmento, Mário J Silva, and Eugénio de Oliveira. Clues for detecting irony in user-generated contents: oh...!! it's so easy;-). In *1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, 2009.

[27] Asli Celikyilmaz, Dilek Hakkani-Tur, and Gokhan Tur. Lda based similarity modeling for question answering. In *NAACL HLT Workshop on Semantic Search*, 2010.

[28] Olivier Chapelle. Training a support vector machine in the primal. *Neural Comput.*, 19(5):1155–1178, May 2007.

[29] Eugene Charniak. A maximum-entropy-inspired parser. In *ANLP*, pages 132–139, 2000.

[30] Yun Chi, Yirong Yang, and Richard R. Muntz. Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical form. In *SSDBM*, pages 11–20, 2004.

[31] Massimiliano Ciaramita and Yasemin Altun. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *EMNLP*, 2006.

[32] Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL02*, 2002.

[33] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[34] Chad Cumby and Dan Roth. Kernel Methods for Relational Learning. In *ICML*, 2003.

[35] Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In *LREC*, 2010.

[36] Hal Daumé, III. Frustratingly easy domain adaptation. *ACL*, 2007.

[37] Hal Daumé III and Daniel Marcu. A tree-position kernel for document compression. In *DUC*, 2004.

[38] Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. Modelling, visualising and summarising documents with a single convolutional neural network. Technical report, University of Oxford, 2014.

[39] Michael Denkowski and Alon Lavie. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*, 2011.

[40] Ludovic Denoyer and Patrick Gallinari. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. *SIGIR Forum*, 41:79–90, June 2007.

[41] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[42] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *ACL*, 2003.

[43] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[44] Andrew S. Fast and David Jensen. Why stacked models perform effective collective classification. In *ICDM*, 2008.

[45] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3), 2010.

[46] Vojtech Franc and Sören Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *ICML*, pages 320–327, 2008.

[47] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, 2007.

[48] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[49] Kuzman Ganchev and Mark Dredze. Small statistical models by random feature mixing. In *In workshop on Mobile NLP at ACL*, 2008.

[50] Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28:245–288, 2002.

[51] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *ACL*, 2011.

[52] Alex Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. In *CS224N Project Report, Stanford*, 2009.

[53] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *ICML*, pages 1319–1327, 2013.

[54] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*, 2004.

[55] Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[56] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[57] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[58] Michael Heilman and Noah A. Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *NAACL*, 2010.

[59] A. Hickl, J. Williams, J. Bensley, K. Roberts, Y. Shi, and B. Rink. Question answering with lcc chaucer at trec 2006. In *TREC*, 2006.

[60] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.

[61] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *KDD*, 2004.

[62] Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. Finding similar questions in large question and answer archives. In *CIKM*, 2005.

[63] Zongcheng Ji, Fei Xu, Bin Wang, and Ben He. Question-answer topic model for question retrieval in community question answering. In *CIKM*, 2012.

[64] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML)*, pages 137–142, Berlin, 1998. Springer.

[65] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[66] T. Joachims. A support vector method for multivariate performance measures. In *International Conference on Machine Learning (ICML)*, pages 377–384, 2005.

[67] T. Joachims and Chun-Nam John Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2-3):179–193, 2009. ECML.

[68] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[69] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[70] Richard Johansson and Alessandro Moschitti. Relational features in fine-grained opinion analysis. *Computational Linguistics*, 39(3):473–509, 2013.

[71] Partly Sunny with a Chance of Hashtags competition at kaggle.com. https://www.kaggle.com/c/crowdflower-weather-twitter, 2013.

[72] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, June 2014.

[73] Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *ACL*, July 2006.

[74] Jun'ichi Kazama and Kentaro Torisawa. Speeding up training with tree kernels for node relation labeling. In *EMNLP*, 2005.

[75] Jason S. Kessler, Miriam Eckert, Lyndsie Clark, and Nicolas Nicolov. The 2010 ICWSM JDPA sentiment corpus for the automotive domain. In *ICWSM-DWC*, 2010.

[76] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, Doha, Qatar, October 2014.

[77] Klaus Krippendorf. *Content Analysis: An Introduction to Its Methodology, second edition*, chapter 11. Sage, Thousand Oaks, CA, 2004.

[78] Rui Kuang, Eugene Ie, Ke Wang, Kai Wang, Mahira Siddiqi, Yoav Freund, and Christina S. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *3rd International IEEE Computer Society Computational Systems Bioinformatics Conference (CSB 2004)*, pages 152–160, 2004.

[79] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *ACL*, 2003.

[80] Taku Kudo, Jun Suzuki, and Hideki Isozaki. Boosting-based parse reranking with subtree features. In *ACL*, 2005.

[81] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[82] Christina Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–76, 2004.

[83] Xin Li and Dan Roth. Learning question classifiers. In *COLING*, 2002.

[84] Zhengdong Lu and Hang Li. A deep architecture for matching short texts. In *NIPS*, 2013.

[85] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[86] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press, 2006.

[87] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, 2013.

[88] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[89] Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. In *7th International Workshop on Semantic Evaluation (Semeval'13)*, 2013.

[90] Andrés Montoyo, Patricio Martínez-Barco, and Alexandra Balahur. Subjectivity and sentiment analysis: An overview of the current state of the area and envisaged developments. *Decision Support Systems*, 53(4):675–679, 2012.

[91] A. Moschitti and S. Quarteroni. Linguistic Kernels for Answer Re-ranking in Question Answering Systems. *Information Processing & Management*, 2010:1–36, 2010.

[92] A. Moschitti and F. Zanzotto. Fast and effective kernels for relational learning from texts. In Zoubin Ghahramani, editor, *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, 2007.

[93] Alessandro Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, 2006.

[94] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *EACL*, pages 113–120, 2006.

[95] Alessandro Moschitti. Kernel methods, syntax and semantics for relational text categorization. In *CIKM*, 2008.

[96] Alessandro Moschitti. Syntactic and semantic kernels for short text pair categorization. In *EACL*, 2009.

[97] Alessandro Moschitti, Daniele Pighin, and Roberto Basili. Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224, 2008.

[98] Alessandro Moschitti and Silvia Quarteroni. Kernels on linguistic structures for answer extraction. In *ACL*, 2008.

[99] Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *ACL*, 2007.

[100] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[101] Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. Semeval-2013 task 2: Sentiment analysis in twitter. In *SemEval*, 2013.

[102] Eric W. Noreen. *Computer-Intensive Methods for Testing Hypotheses : An Introduction*. Wiley-Interscience, 1989.

[103] Iadh Ounis, Craig Macdonald, Jimmy Lin, and Ian Soboroff. Overview of the TREC-2011 microblog track. In *TREC*, 2011.

[104] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT*, 2013.

[105] Sebastian Padó. *User's guide to* `sigf`*: Significance testing by approximate randomisation*, 2006.

[106] Martha Palmer, Paul Kingsbury, and Daniel Gildea. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

[107] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL*, 2004.

[108] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[109] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *EMNLP*, pages 79–86, 2002.

[110] Polina Panicheva, John Cardiff, and Paolo Rosso. Identifying subjective statements in news titles using a personal sense annotation framework. *Journal of the American Society for Information Science and Technology*, 64(7):1411–1422, 2013.

[111] J. Pei, J. Han, Mortazavi B. Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In *ICDE*, 2001.

[112] Slav Petrov and Ryan McDonald. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, 2012.

[113] Daniele Pighin and Alessandro Moschitti. Efficient linearization of tree kernel functions. In *CONLL*. ACL, 2009.

[114] Christopher Pinchak. A probabilistic answer type model. In *In EACL*, 2006.

[115] Barbara Plank and Alessandro Moschitti. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *ACL*, 2013.

[116] Soujanya Poria, Erik Cambria, Grégoire Winterstein, and Guang-Bin Huang. Sentic patterns: Dependency-based rules for concept-level sentiment analysis. *Knowledge-Based Systems, Special Issue on Big Data for Social Analysis*, pages 1–32, 2014.

[117] John M. Prager. Open-domain question-answering. *Foundations and Trends in Information Retrieval*, 1(2):91–231, 2006.

[118] Silvia Quarteroni, Vincenzo Guerrisi, and Pietro La Torre. Evaluating multi-focus natural language queries over data services. In *LREC*, 2012.

[119] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[120] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 239–248, 2005.

[121] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, 2011.

[122] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995.

[123] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in Twitter. *Language Resources and Evaluation*, 47(1):239–268, 2013.

[124] Konrad Rieck, Tammo Krueger, Ulf Brefeld, and Klaus-Robert Mueller. Approximate tree kernels. *Journal of Machine Learning Research*, 11:555–580, 2010.

[125] Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. Statistical machine translation for query expansion in answer retrieval. In *ACL*, 2007.

[126] Ellen Riloff, Siddharth Patwardhan, and Janyce Wiebe. Feature subsumption for opinion analysis. In *EMNLP*, 2006.

[127] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: an experimental study. In *ACL*, 2011.

[128] Jason Weston Ronan Collobert. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, 2008.

[129] Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M Mohammad, Alan Ritter, and Veselin Stoyanov. Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, SemEval '2015, Denver, Colorado, June 2015. Association for Computational Linguistics.

[130] H. Saigo, J.P. Vert, T. Akutsu, and N. Ueda. Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, 2004.

[131] A. Severyn and A. Moschitti. Fast support vector machines for structural kernels. In *ECML/PKDD*, 2011.

[132] Aliaksei Severyn and Alessandro Moschitti. Large-scale support vector learning with structural kernels. In *ECML/PKDD (3)*, pages 229–244, 2010.

[133] Aliaksei Severyn and Alessandro Moschitti. Structural relationships for large-scale learning of answer re-ranking. In *SIGIR*, 2012.

[134] Aliaksei Severyn and Alessandro Moschitti. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[135] Aliaksei Severyn, Alessandro Moschitti, Manos Tsagkias, Richard Berendsen, and Maarten de Rijke. A syntax-aware re-ranker for microblog retrieval. In *SIGIR*, 2014.

[136] Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. Building structures from classifiers for passage reranking. In *CIKM*, 2013.

[137] Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. Learning adaptable patterns for passage reranking. In *CoNLL*, 2013.

[138] Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. Learning semantic textual similarity with structural representations. In *ACL*, 2013.

[139] Dennis Shasha, Jason Tsong-Li Wang, and Sen Zhang. Unordered tree mining with applications to phylogeny. In *ICDE*, pages 708–719, 2004.

[140] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[141] D. Shen and M. Lapata. Using semantic roles to improve question answering. In *EMNLP-CoNLL*, 2007.

[142] L. Shen, A. Sarkar, and A. Joshi. Using LTAG Based Features in Parse Reranking. In *EMNLP*, 2003.

[143] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. *CIKM*, 2014.

[144] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Gregoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW*, 2014.

[145] Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. *Proceedings of Advances in Neural Information Processing Systems*, 2009.

[146] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander J. Smola, and S. V. N. Vishwanathan. Hash kernels for structured data. *JMLR*, 10:2615–2637, 2009.

[147] Stefan Siersdorfer, Sergiu Chelaru, Wolfgang Nejdl, and Jose San Pedro. How useful are your comments?: Analyzing and predicting YouTube comments and comment ratings. In *WWW*, 2010.

[148] S. Small, T. Strzalkowski, T. Liu, S. Ryan, R. Salkin, N. Shimizu, P. Kantor, D. Kelly, and N. Wacholder. Hitiqa: Towards analytical question answering. In *COLING*, 2004.

[149] Ian Soboroff, Iadh Ounis, J Lin, and I Soboroff. Overview of the TREC-2012 microblog track. In *TREC*, 2012.

[150] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[151] Anders Søgaard and Anders Johannsen. Robust learning in random subspaces: Equipping nlp for oov effects. In *COLING*, 2012.

[152] Radu Soricut and Eric Brill. Automatic question answering using the web: Beyond the factoid. *Inf. Retr.*, 9(2):191–206, 2006.

[153] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[154] Ingo Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 4:1071–1105, 2003.

[155] Christopher Stokoe, Michael P. Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. In *SIGIR*, 2003.

[156] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers on large online QA collections. In *Proceedings of ACL-HLT*, 2008.

[157] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Comput. Linguist.*, 37(2):351–383, June 2011.

[158] Jun Suzuki and Hideki Isozaki. Sequence and Tree Kernels with Statistical Feature Mining. In *NIPS*, 2005.

[159] Oscar Täckström and Ryan McDonald. Semi-supervised latent variable models for sentence-level sentiment analysis. In *ACL*, 2011.

[160] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Neural Information Processing Systems*, 2003.

[161] Wen tau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question answering using enhanced lexical semantic models. In *ACL*, August 2013.

[162] Alexandre Termier, Marie-Christine Rousset, and Michèle Sebag. Dryade: A new approach for discovering closed frequent trees in heterogeneous tree databases. In *ICDM*, pages 543–546, 2004.

[163] Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, pages 127–132, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[164] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[165] Francesca Trentini, Markus Hagenbuchner, Alessandro Sperduti, and Franco Scarselli. A self-organising map approach for clustering of xml documents. In *IJCNN*, pages 1805–1812. IEEE, 2006.

[166] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[167] Olga Uryupina, Barbara Plank, Aliaksei Severyn, Agata Rotondi, and Alessandro Moschitti. SenTube: A corpus for sentiment analysis on YouTube social media. In *LREC*, 2014.

[168] K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proceedings of the IJCAI*, pages 55–60, 1999.

[169] E. M. Voorhees. Overview of the TREC 2001 Question Answering Track. In *Proceedings of TREC*, 2001.

[170] E. M. Voorhees. Overview of TREC 2003. In *TREC*, 2003.

[171] E. M. Voorhees. Overview of the trec 2004 question answering track. In *TREC*, 2004.

[172] Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. Takelab: Systems for measuring semantic text similarity. In *SemEval*, 2012.

[173] Chen Wang, Mingsheng Hong, Jian Pei, Haofeng Zhou, Wei Wang, and Baile Shi. Efficient pattern-growth methods for frequent tree pattern mining. In *PAKDD*, pages 441–451, 2004.

[174] Mengqiu Wang and Christopher D. Manning. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *ACL*, 2010.

[175] Mengqiu Wang, Noah A. Smith, and Teruko Mitaura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP*, 2007.

[176] Sida Wang and Christopher Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, 2012.

[177] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *EMNLP*, 2005.

[178] Michael J. Wise. Yap3: improved detection of similarities in computer program and other texts. In *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, SIGCSE '96, pages 130–134, New York, NY, USA, 1996. ACM.

[179] G Wu and EY Chang. Class-boundary alignment for imbalanced dataset learning. *ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC*, pages 49–56, 2003.

[180] Yuanbin Wu, Qi Zhang, Xuanjing Huang, and Lide Wu. Phrase dependency parsing for opinion mining. In *EMNLP*, 2009.

[181] Yi Xia and Yirong Yang. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202, 2005. Student Member-Yun Chi and Fellow-Richard R. Muntz.

[182] Saif M. Mohammad Xiaodan Zhu, Svetlana Kiritchenko. Nrc-canada-2014: Recent improvements in sentiment analysis of tweets, and the Voted Perceptron. In *Eighth International Workshop on Semantic Evaluation Exercises (SemEval-2014)*, 2014.

[183] Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. Retrieval models for question and answer archives. In *SIGIR*, 2008.

[184] Liang Huai Yang, Mong-Li Lee, Wynne Hsu, and Xinyu Guo. 2pxminer: an efficient two pass mining of frequent xml query patterns. In *KDD*, pages 731–736, 2004.

[185] Nan Yang, Shujie Liu, Mu Li, Ming Zhou, and Nenghai Yu. Word alignment modeling with context dependent deep neural network. In *ACL*, pages 166–175, Sofia, Bulgaria, August 2013.

[186] Xuchen Yao, Benjamin Van Durme, Peter Clark, and Chris Callison-Burch. Answer extraction as sequence tagging with tree edit distance. In *NAACL*, 2013.

[187] Ainur Yessenalina, Yisong Yue, and Claire Cardie. Multi-level structured models for document sentiment classification. In *EMNLP*, 2010.

[188] Chun-Nam John Yu and T. Joachims. Training structural svms with kernels using sampled cuts. In *KDD*, 2008.

[189] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *CoRR*, 2014.

[190] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of ICDM*, 2003.

[191] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *Knowledge and Data Engineering, IEEE Transactions on*, 17(8):1021–1035, 2005.

[192] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2002. ACM Press.

[193] F. M. Zanzotto, L. Dell'Arciprete, and A. Moschitti. Efficient graph kernels for textual entailment recognition. *FUNDAMENTA INFORMATICAE*, 2010, 2010.

[194] F. M. Zanzotto and A. Moschitti. Automatic Learning of Textual Entailments with Cross-Pair Similarities. In *COLING*, 2006.

[195] Fabio Massimo Zanzotto, Marco Pennacchiotti, and Alessandro Moschitti. A Machine Learning Approach to Recognizing Textual Entailment. *Natural Language Engineering*, Volume 15 Issue 4, October 2009:551–582, 2009.

[196] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, 2012.

[197] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.