PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Towards structured representation of academic search results

Daniil Mirylenka

Advisor:

Prof. Andrea Passerini

Università degli Studi di Trento

February 2015

Abstract

Searching for scientific publications is a tedious task, especially when exploring an unfamiliar domain. Typical scholarly search engines produce lengthy unstructured result lists, which are difficult to comprehend, interpret and browse. An informative visual summary could convey useful information about the returned results as a whole, without the need to sift through individual publications.

The first contribution of this thesis is a novel method of representing academic search results with concise and informative topic maps. *The method consists of two steps: i) extracting interrelated topics from the publication titles and abstracts, and ii) summarizing the resulting topic graph. In the first step we map the returned publications to articles and categories of Wikipedia, constructing a graph of relevant topics with hierarchical relations. In the second step we sequentially build a summary of the topic graph that represents the search results in the most informative way. We rely on sequential prediction to automatically learn to build informative summaries from examples. The summarized topic maps share the most of the benefits and avoid most of the drawbacks of the current methods for grouping documents, such as clustering, topic models, and predefined taxonomies. Specifically, the topic maps are dynamic, fine-grained, of flexible granularity, with up-to-date topics connected with informative relations and having meaningful concise labels.*

The second contribution of this thesis is a method for bootstrapping domain-specific ontologies from the categories of Wikipedia. The method performs three steps: i) selecting the set of categories relevant to the domain, ii) classifying the categories into classes and individuals, and iii) classifying the sub-category relations into "subclass-of", "instance-of", "part-of" and "related-to". In each step we rely on binary classification, which makes the method flexible and easily extensible with new features. For the purpose of academic search, the proposed method advances the creation of semantically rich topic maps. In general, the method semi-automates the construction of large-scale domain ontologies, benefiting multiple potential applications.

Providing ground truth data for structured prediction of large objects, such as topic map summaries or domain ontologies, is tedious. The last contribution of this thesis is an initial investigation into reducing the labeling effort in structured prediction tasks. First, we present a labeling interface that suggests topics to be added to the ground truth topic map summary. We modify a state of the art sequential prediction method to iteratively learn from the summaries one topic at a time, while retaining the convergence guarantees. Second, we present an interactive learning method for selecting the categories of Wikipedia relevant to a given domain. The method reduces the number of required labels by actively selecting the queries to the annotator and learning one label at a time.

Keywords
academic search; search result clustering; structured representation; topic maps; Wikipedia; domain ontologies; structured prediction; sequential prediction; learning to search;

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As researchers, we look for publications on a daily basis—to keep up with our research fields, to expand our competences, and to find related work. The number of papers being published is far beyond what a scientist can consume, so we have to be very selective in what we read or even look through. Digital libraries and search engines greatly simplify this task by enabling effective search over large collections of published contributions. With now traditional search interfaces, *finding a specific paper* has become a matter of few keystrokes, provided that we know beforehand some of the metadata associated with the paper, such as the keywords in the title, the authors or the publication venue. The way the search results are presented—the infamous ten blue links—proves insufficient for more complex, yet typical, scholarly search tasks.

1.1 Motivation

1.1.1 Exploratory search

We often use academic search engines to perform *exploratory search* [59, 105], when we do not know in advance what we want to find. For instance, when learning about an unfamiliar research domain, a great deal of exploration is usually needed just to find out what specific publications we are looking for. We start with a vague understanding of our information need, and search our way to reducing the uncertainty. We interact with the search engine in a loop, in which we formulate the search queries, examine the returned

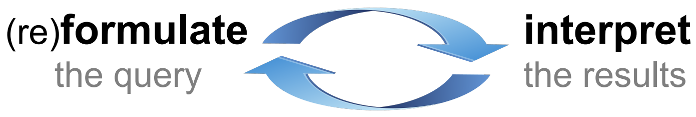


Figure 1.1: Exploratory search interaction loop.

results, and re-formulate the queries based on what we learned. In this interaction, our ability to comprehend the search results becomes a bottleneck.

The results are typically presented as plain ranked lists thousands of items long, clearly exceeding what we can even look through, so we typically view only a few of the first result pages. We skim through the titles and abstracts of the returned papers, one by one, trying to understand whether they could be relevant to our search task, and which of them we should examine further (e.g. download). The excessive number of results, along with the necessity to decide which of them should be viewed in the limited time cause anxiety and frustration [16, 38, 48, 72]. In addition to analyzing individual papers, we try to form an opinion of the returned results as a whole, in order to understand, in general, a) how relevant the results are to our task, b) how far in the result list we should go, c) whether the current search query describes our information need correctly, and d) how we can re-formulate the query to get better or additional results. Comprehending the search results by skimming through individual items is a cognitively demanding task. A useful presentation of the search results should alleviate the cognitive load on the user as much as possible. Additionally, this representation should allow the user to interact with the search results, for example, by focusing on different subsets, or adjusting the search query, or grouping them with various levels of granularity. These interactive capabilities should help the users sharpen the understanding of their information needs and improve the exploratory search experience.

1.1.2 Graduate student scenario

Consider a more concrete scenario of a graduate student who is starting to investigate an unfamiliar research topic. The likely information-seeking tasks the student might have are related to understanding the *structure of the topic*, including:

- the relevant concepts and relations between them,
- the various sub-topics and related topics,
- the main research problems (solved and unsolved),
- the main methods,
- the most influential results,
- the current research directions and trends,
- the most prominent researchers/communities working on the topic,
- the most relevant papers, including
  - introductory/survey papers,
  - methodological papers,
  - experimental comparisons,
  - empirical evaluations,
  - tool (e.g. software) descriptions.

Arguably, the simplest way to accomplish these tasks is to read a comprehensive up-to-date survey of the topic, which will contain the answers to most of the presented questions. However, the surveys are only being written for well-established and relatively large topics. In our scenario, typical for the graduate students, the user is looking for a new research direction, which is novel, unestablished, small, and at first vaguely-defined. In this case, he/she will likely turn to an academic search engine to find the answers.

Imagine for a moment that the search engine returned all of the relevant publications on the topic of interest in response to the first query. The presented results should convey the answer to the user's questions with minimal effort on the user's side. Ideally, the results should be presented in a way that exposes the *semantic structure of the topic* in terms of the elements described above, such as sub-topics, relevant concepts, research problems, methods, and so on. In this way the ideal representation of the query results will be analogous to a *survey* that presents the most important papers in the context of, and according to, the semantic structure of the underlying topic.

In a more realistic case, the first returned result list will not cover all of the relevant papers, and the user will have to iteratively refine the search query. In this case the structured representation should additionally help in understanding how well the current search results match the user's information need. As mentioned in the previous section, the useful representation should allow interaction with the search results by aggregating them at various levels of detail, focusing on various subsets, and *navigating* to the results for related queries.

1.2 Problem formulation

Motivated by the graduate student scenario, as well as exploratory academic search in general, we formulate the following problem:

Problem (Structured representation of the academic search results)**.** *Can we construct a visual representation that conveys the semantic structure of the academic search results as a whole, similarly to a human-written survey?*

Additional desired property of the sought representation is that it should allow interaction with the search results, namely:

- viewing the structure of the results at different levels of granularity,
- focusing on various parts of the search results (filtering),
- refining the search request through interaction with the results (navigating to the results for related queries).

The formulated problem of generating a "survey" of the search results is more of a research direction than something we hope to achieve in the nearest future. In this thesis we took the first steps—some of the many possible—in this direction.

1.3 Contributions and structure of the thesis

Contribution 1 (Sections 3.1–3.2). *The idea of topic maps as a way to summarize academic search results in terms of the most important topics arranged into a taxonomy.*

This is the main contribution of this thesis. The proposed way of representing the academic search results is novel with respect to both the current academic search engines and the methods described in the literature. Satisfying many of the requirements listed in Section 1.1 and having advantages over other state-of-the-art methods, topic maps are a significant step towards building the structured survey-like representations of the academic search results. This contribution is described in Chapter 3. The topic maps are presented in Section 3.1, while in Section 3.2 we show their usage in a prototype academic search tool. Other contributions are related to our implementation of the topic maps and lie in slightly different dimensions.

Contribution 2 (Section 3.3). *A method for building topic maps from the categories and articles of Wikipedia.*

The method relies on *wikification*—automatic annotation of texts with links to Wikipedia—in order to map the search results to Wikipedia articles. The relations between articles and categories are retrieved from Wikipedia to form a topic graph, which, after a number of processing steps, becomes a topic map. Few methods have been proposed in the literature to represent documents collections with Wikipedia articles. Ours is unique in that it builds a hierarchy (directed acyclic graph) of topics and relies on the Wikipedia category network.

Contribution 3 (Section 3.4). *A structured prediction–based method for summarizing topic maps in the most informative way.*

A comprehensible topic map should contain a reasonable number of nodes (far less than a hundred). The question of how to summarize topic maps most informatively led us to a supervised machine learning approach. We creatively applied a state-of-the-art structured prediction technique to this novel problem.

Contribution 4 (Chapter 4). *A supervised learning–based method for bootstrapping domain ontologies from the category network of Wikipedia.*

The structure of the topic maps described in Chapter 3 is limited to the untyped hierarchical relations between the topics. The motivation to enrich the structure of the topic maps with ontological information led us to the idea of the automatic construction of domain-specific ontologies of scientific disciplines. For a specified domain our method bootstraps a large-scale ontology by extracting the relevant categories from Wikipedia, classifying them into classes and individuals, and classifying relations into a few specific types and the generic “related-to”. The main novelty of the method is that i) it relies exclusively on the category network of Wikipedia (rather than the rich textual and semi-structured information therein) ii) it avoids complex heuristics and rule-based pipelines, employing simple binary classification in all the three steps.

Contribution 5 (Chapter 5)**.** *Two methods that reduce the effort of providing training data for sequential structured prediction tasks.*

In building informative summaries of topic maps and extracting domain ontologies from Wikipedia we relied on supervised machine learning for constructing large structured objects. In both of these problems we ran into an interesting setting, in which providing the complete training examples of the predicted structures was time-consuming or even entirely impractical. We developed alternative methods of structured prediction for our tasks, which reduce the labeling effort by either i) interactively suggesting labels to the annotator and learning from the provided feedback, or ii) requiring only partial supervision, which is sparingly asked of the annotator in the active manner. We describe these methods and the initial theoretical analysis thereof in Chapter 5.

Publications

Research presented in this dissertation resulted in the following publications:

- [2] Marcos Baez, Daniil Mirylenka, and Cristhian Parra. *Understanding and supporting search for scholarly knowledge.* Proceeding of the 7th European Computer Science Summit (2011): 1-8 (ECSS’2011).
- [69] Daniil Mirylenka and Andrea Passerini. *Navigating the topical structure of academic search results via the Wikipedia category network.* Proceedings of the 22nd ACM international conference on Conference on Information & Knowledge Management. ACM, 2013 (CIKM’2013).
- [70] Daniil Mirylenka and Andrea Passerini. *Scienscan—an efficient visualization and browsing tool for academic search.* Machine Learning and Knowledge Discovery in Databases. Springer Berlin Heidelberg, 2013. 667-671 (ECML-PKDD’2013).
- [68] Daniil Mirylenka and Andrea Passerini. *Learning to grow structured visual summaries for document collections.* ICML Workshop on Structured Learning: Inferring Graphs from Structured and Unstructured Inputs. 2013 (SLG@ICML’2013).
- [71] Daniil Mirylenka and Andrea Passerini. *Supervised graph summarization for structuring academic search results.* NIPS Workshop on Constructive Machine Learning. 2013 (CML@NIPS’2013).
- Daniil Mirylenka, Andrea Passerini, and Luciano Serafini. *Bootstrapping domain ontologies from Wikipedia: a uniform approach.* Under submission, 2015.

Chapter 2

Background

In this thesis we address the problem of building structured representations of the academic search results, and propose a possible way of building such representations from the categories and articles of Wikipedia with the help of machine learning techniques. In this chapter we provide the necessary background for the formulated problem and the proposed methods. In Sections 2.1 and 2.2 we describe the current academic digital libraries and search engines, as well as the state-of-the-art methods for representing the academic search results, Web search results, and documents in general. In Sections 2.3–2.6 we give the necessary background on the machine learning methods used in this work.

2.1 Academic search services

Online publication repositories have existed for as long as the Web itself, enabling search over ever increasing collections of published material. Similarly to the general Web search, the presentation of the academic search results has remained quite limited. One piece of functionality that has traditionally been provided to the users in addition to displaying the "ten blue links" is filtering and browsing of the result set or the whole indexed collection according to some attribute, such as the publication year or the authors. In contrast to the available services, the literature contains substantial research on representation (typically clustering) of the search results, publication collections, and document collections in general. In this section we will describe the current academic services, and survey some of the methods most relevant to representing the academic search results. Our survey will be biased towards the services providing access to computing-related literature.

2.1.1 The model of the academic search results

First, we would like to list the types of information about publications that can be found online. A typical academic search service will list the following attributes for most publications in the search results:

- a **title**,
- an **abstract**,
- one or more **authors** (possibly with own attributes, such as affiliation, etc.),
- a publication **type** (such as article, chapter, book, Ph.D. thesis, etc.),
- a publication **language**,
- a publication **year**,
- a publication **venue**, such as a conference or a journal,
- a publishing organization (**publisher**),
- other publication-related information, such as volume, pages, ISBN, etc.,
- a list of **references**—papers that *are cited* in the publication,
- a list of **citations**—papers that *cite* the publication,
- possibly, a list of **keywords**.

In social applications (discussed further) additional attributes may include:

- a list of **readers**—users, who bookmarked the publication,
- a list of user **tags**,
- a number of user **reviews**.

We have not listed the full text of a publication among the typical attributes. The publication contents is usually subject to copyrights and is thus not generally available online. Therefore, we assume that the available textual information about a publication is limited to the title and the abstract (possibly, also to the keywords).

2.1.2 Service types

With scholarly services expanding functionality, the boundaries between different types of services have become blurred. In the following we will loosely differentiate between digital libraries, search engines and social networking sites mainly for historical reasons.

Digital libraries were among the first academic services to appear. Initially, publishing organizations have put the catalogs of their publications online, allowing keyword-based search or browsing according to the various attributes (see Section (2.1.1)). Over time, digital libraries have started to aggregate other publishers' content, providing access to the extended collections. Some of the most famous publishers' digital libraries related to computing include `ACM Digital Library`, `IEEEXplore` and `SpringerLink`.

One of the most widely used bibliographic databases in computer science—Digital Bibliography & Library Project [51] (`DBLP`)—is not associated with any publisher. `DBLP` collects and indexes publications from a predefined number of sources, relying largely on human effort [52, 53]. Another remarkable digital library, `arXiv` [61], indexes publication preprints in a number of fields, including computer science. The texts of the articles are submitted to `arXiv` by the authors, and are made openly available.

Academic search engines crawl the Web, aggregating the metadata about publications form a variety of sources, such as digital libraries, institutions' publication archives, and authors' personal pages. Some of the academic search engines covering the area of computing include `Google Scholar` (henceforth, `Scholar`), `Microsoft Academic Search` (henceforth `MSAcademic`), CiteSeer^X and `Arnetminer` (henceforth `AMiner`).

In addition to publication metadata, some of the search engines maintain detailed information about the authors, institutions, and venues, and display profile pages for these entities including various statistics, such as the number of citations and h-index. Another type of functionality, useful for exploratory search, is providing citation links. As of now[1], `MSAcademic` provides citation links in both directions, allowing the exploration not only of *citations* (as most other services), but also *references*. Other functionality differs by the search engine, and may include suggestion of related publications, viewing co-citations and co-authors, and displaying citation graphs.

Specialized **social networking services** have appeared relatively recently, providing the familiar social networking and social bookmarking/tagging functionality for scientists. In these services users can create and manage their own profiles, upload publications, add publications to their libraries (bookmark them), assign tags, arrange publications into folders, "connect" with other users or "follow" them, and, of course, search for publications. Some of the sites currently active include `Mendeley`, `CiteULike`, `Academia.edu`, `Researchr`, `ResearchGate` and `Zotero`. With full access to user profiles and their activity, these sites typically implement recommendation of publications to users, as well as suggestion of related publications.

Reference managers / reading applications is another relevant type of application, which partly intersects with specialized social networking applications. Among other functionality, applications like `Docear`, `readcube` and `Papers` allow organizing the references on a user's desktop, searching for publications in multiple online sources, annotating the texts of the PDF documents, exporting citations, and many more. `Mendeley`, a social networking service mentioned earlier, provides a desktop application as well.

2.2 Structured representation of document collections

Academic search engines return metadata about scientific publications as query results. These pieces of metadata can be viewed as small documents that consist of text (title and abstract) and some other attributes (see Section 2.1.1). Therefore, methods for visualization or clustering of (short) documents can be applied to academic search results. Scientific publications have, however, some specific attributes that can be exploited by more specific methods. In particular, scientific publications are authored, dated (due

[1]In the text "now" refers to the submission data of this thesis, February 2015.

to the publication year) and linked (due to citations). The latter property makes the academic search results similar to Web documents, which are connected with hyperlinks. In the following we will review some of the methods for representing scientific publications, Web search results, and documents collections in general.

The methods we review in this section will roughly fall into two main categories, which we call *predefined groupings* (Section 2.2.2) and *unsupervised methods* (Section 2.2.3). The two categories of methods will have different pros and cons with respect to the informativeness of the grouping, the required effort, and the performance. In Section (2.2.4) we describe arranging the documents according to Wikipedia-based topics, which combines the best of both worlds, and which is the approach we take in this thesis.

2.2.1 Dimensions of the structured representation

The methods we will review in the following section can be used for representing the collections of documents in a structured way. In order to describe and compare these methods we will first give a common definition of what is structured representation and list some dimensions for comparison.

Definition 1. *A* ***structured representation*** *of a collection of documents is an arrangement of the documents into groups, with optional relations between the groups.*

We do not formalize Definition 1 further, as the specific details will vary in different methods. The differences will span the nature and the crispness of the groups, as well as the presence, direction, type and nature of the relations.

The nature of the groups. The documents can be grouped according to various criteria. Examples include grouping according to some attribute (e.g. publication venue), unsupervised clustering (e.g. according to text similarity), grouping into pre-defined topics, latent topics, and so on.

The labels of the groups. The meaningful labeling of groups is one of the most important properties for informative structured representation of documents. High-quality labels should be short, unambiguous, accurate and grammatically sound. Ideally, they should correspond to semantic concepts that are meaningful to the user, such as, for example, the names of some pre-defined topics. Other types of labels include representative phrases and keyword sets. Many *unsupervised* methods, such as clustering and self-organizing maps do not provide any labels for the constructed groups. Other methods provide only rudimentary labels, such as the most probable words returned by the probabilistic topic models, which are far from ideal. Generating the labels for an existing grouping is a research problem in its own right [81, 95, 99].

Static vs. dynamic grouping. We introduce this dimension to distinguish between the groupings that are built a priory for all indexed documents, and the groupings that are build specifically for the displayed documents (e.g the search results). In other words, the groups are *static* if they exist independently of the result sets. Grouping according to an attribute (e.g. publication venue) or a pre-defined set of topics is clearly *static*. Clustering methods can be either *static* or *dynamic* depending on what document set they are applied to. Some methods will fall in between these categories, for instance, if some static a priory grouping is refined for each result set.

Automatic vs. hand-coded vs. crowd-sourced grouping. The distinction between automatic and hand-coded groups is self-explanatory. We call a grouping *crowd-sourced* if it has been created by a large *unrestricted* set of authors in a distributed, decentralized and voluntary manner. Such are the grouping of the papers according to author-defined keywords (when the keywords are unrestricted), or grouping of items according to user tags in a social tagging system, or grouping of articles into categories in Wikipedia. In contrast, the `ACM CCS` taxonomy, reportedly[2] created by 120 computing specialists, will be classified as *hand-coded*, as the set of authors, though large, was purposefully selected.

Unsupervised vs. supervised grouping. This classification applies to automatic grouping methods. Most of the state-of-the-art methods are *unsupervised* in that they do not *learn* from example groupings. Exceptions include supervised clustering [17, 30, 40, 107], supervised topic models [60], and our method presented in section 3.4. This distinction is not clear-cut, as various methods can use *supervision* of different degrees, and at different stages.

Granularity of the grouping. We will refer to the groupings with a small number of large groups as *coarse*. Conversely, the grouping with large number of small groups is *fine*. Another important property related to the granularity of groups is whether the coarseness of the grouping can be dynamically changed. We will distinguish between the groupings of *fixed* and *flexible* granularity. Flexibility is a favorable property of a grouping, as it gives the user more control over the representation of the documents. Most *static* methods, such as plain clustering, have *fixed* granularity. Exceptions include *hierarchical* groupings, such as hierarchical clustering [110] and hierarchical topic models [4, 5, 54], in which documents are grouped at multiple levels of granularity simultaneously. Another way of obtaining a *flexible* grouping is dynamically summarizing (aggregating) a *fine* grouping into *coarse*r groupings, which is the approach we took in our method (Section 3.4).

Soft vs. hard grouping. In *hard* (*crisp*) groupings every document is assigned to at most one group. Such are the traditional *hard* clustering methods. In contrast, in *soft* (*fuzzy*) clustering, documents can belong to multiple clusters simultaneously, and even have degrees of membership in the clusters. In probabilistic topic models, such as LDA [8], each document has a probability distribution over the latent topics, which can be

[2] http://www.acm.org/news/featured/2012-acm-ccs

thought of as *soft* groups.

Relations between the groups. Relations between the groups is the third important element of the structured representation of documents, the first two being document-to-group assignments and the labeling of the groups. The first distinction should be made between the groupings that include relations and the groupings that do not. In plain clustering methods, for instance, the groups are not related in any way. The second distinction is between *directed* and *undirected* relations. Any hierarchy defined on the groups is naturally translated into a directed relation. In hierarchical clustering, for instance, the relation is between a bigger cluster and every smaller cluster it contains. Another example is when the documents are grouped according to a taxonomy of concepts: the taxonomical relation translates into the relation between the corresponding groups. In `SKOS` taxonomies [64], such as `ACM CCS`, this hierarchical relation is called "broaderGeneric" (with the inverse being "narrowerGeneric"). An example of an *undirected* relation is generic "related-to" relation, which can be defined between the topics (e.g. as in [89]). The last distinction is between the *typed* and *untyped* relations. If topic taxonomies define one type of relation (e.g. "broaderGeneric"), ontologies can contain multiple relation types. For example, in [76] the authors define semantic relations `relatedEquivalent`, `skos:broaderGeneric`, and `contributesTo` between research topics.

2.2.2 Predefined groupings

Grouping according to an attribute. Most academic search services provide some way of interacting with the search results or the whole indexed collection based on the attributes of the publications (see Section 2.1.1). For instance, `IEEExplore` allows filtering the search results according to the publication type, year, publisher, authors, authors' affiliation, country and city of the conference. Two of the most valuable attributes—citations and references—allow viewing the papers cited in a given paper, and papers that cite a given paper. This form of grouping is usually not used for representing the results visually, but rather for filtering or refining them. According to our classification, grouping by attribute is *static*, *automatic*, *unsupervised*, with no relations between the groups. The granularity of the grouping is *fixed*, and can, depending on the attribute, range from very *coarse* (e.g grouping by publication type) to very *fine* (e.g. grouping by author). Many-to-one attributes (e.g. publication year) and many-to-many attributes (e.g. the author) result in *hard* and *soft* groupings respectively.

Pre-defined topics are used by most digital libraries and some academic search engines. This kind of grouping is *static* and usually *hand-coded*. For example, `MSAcademic` arranges the publications into (as of now fifteen) non-overlapping "Fields of Study", a very *coarse* grouping. The groups used by `Mendeley` form a two-level hierarchy of "Disciplines" and "Sub-disciplines", all of them coarse as well. An example of a rather *fine*-grained grouping is the `ACM Computing Classification System` (`ACM CCS`), which is used as one of the ways to browse the publications in `ACM Digital library`. `ACM CCS` contains about two thousand topics arranged into a multi-level taxonomy. The papers published by `ACM` are explicitly assigned the topics from `ACM CCS` by their authors.

The main benefit of the pre-defined groups is that they have well-defined semantics and meaningful labels. Additional benefit of taxonomies is that they provide hierarchical grouping of documents, which can be viewed at a desired level of granularity. Hierarchical relations between the subtopics provide additional useful information about the structure of the field in a visual way.

The main drawback of the predefined topics is that they have to be manually created and maintained up to date, which requires significant effort. As already mentioned, the new version of `ACM CCS` of 2012 have reportedly taken 120 computing specialists to be constructed. Additional effort is required to assign new documents to the pre-defined topics, by hand or automatically [10, 62]. Another shortcoming of the predefined taxonomies is that, for the most part, they represent a rather *coarse*-grained structure of the field. For example, imagine that we want to group the search results on the topic of `clustering`. The topic "Clustering" is a leaf topic, having no further sub-topics even in `ACM CCS`, one of the most *fine*-grained categorizations in the field.

2.2.3 Unsupervised grouping methods

Despite limited use in the commercial search engines, various unsupervised grouping methods have been proposed in the literature. The main drawback of these automatic methods is that the results of the grouping are **not easily interpretable**. Firstly, some of the discovered groups may simply not correspond to distinct topics relevant to a human user. Secondly, most of the unsupervised learning techniques provide no meaningful labels for the groups. Although there have been research efforts towards generating labels for document collections [81, 95, 99], the results are not yet as expressive and meaningful as manually created topic names. Another shortcoming of most of the unsupervised grouping methods is the **fixed granularity**, with hierarchical methods being an exception. In order to discover meaningful topics in the result set, unsupervised (e.g. latent topic) models have to be previously built on a large corpus of data, a "universal dataset" [75], which is usually a computationally expensive task that cannot be performed on the fly. For most of the methods this implies choosing all the parameters, such as the number of topics, beforehand. The chosen topic granularity has to remain fixed, even though it may not be optimal for specific users and result sets.

Clustering. Cutting et al. [21] first introduced a clustering method, Scatter/Gather, as a metaphor for exploring document collections, while Hearst and Pedersen evaluated this method in the context of Web search [37]. Scatter/Gather produces a *hierarchical*, *dynamic* grouping. A variety of other clustering algorithms have since been applied to Web search results. We refer the reader to [13] for an extensive survey of the subject. We should note that research on search result clustering has largely focused on handling ambiguous queries ("Jaguar", "apple", etc.) rather than on the informative visualization.

Selecting the labels after and independently of the clustering phase is a difficult problem. Most of the approaches produce cluster labels as sets of frequent words that are not grammatically or otherwise connected [13]. Zamir and Etzioni [109] suggested using a suffix tree for discovering phrases to form initial cluster seeds and serve as cluster labels. Other class of methods has been specifically developed with the primary objective of providing meaningful cluster labels. A famous example of this class is the Lingo algorithm by Osiriski et al. [78]. Lingo first selects the frequent phrases that could serve as cluster labels, and then assigns documents to the clusters represented by these labels. This work is also an early example of using a dimensionality reduction technique (Singular Value Decomposition [26]) for discovering topics in search results. Other data mining techniques that have been applied to search result grouping include agglomerative clustering [58], k-means clustering [104], concept lattices [14], and probabilistic topic models [75].

Probabilistic topic models represent a class of methods that have been applied extensively in the context of scientific papers. In probabilistic models the documents are associated with probability distributions over some latent topics. The latent topics in turn are associated with distributions over words. For a given document, the probabilities of various topics can be viewed as degrees of membership in the corresponding clusters.

Griffith and Steyvers [33] applied Latent Dirichlet Allocation (LDA) [8]—a general-purpose topic model—to a collection of abstracts from PNAS. A correlated topic model [7] developed by Blei and Lafferty improved upon LDA by introducing pairwise topic correlations. Pachinko allocation [54] allowed more complex and sparse topic correlations by modeling topic mixtures through directed acyclic graphs. It is also an example of a topic model with hierarchical relations between topics, another example being hLDA (Hierarchical Latent Dirichlet Allocation) [5]. Specific topic models have been developed to account for various aspects of scientific literature, such as explicit document authorship [83, 98, 103, 108], author interests [47], publication venues [98, 103, 108], temporal ordering of the documents [9], topic evolution [6, 36] and citations [25, 36, 73, 102, 108].

The Author-Conference-Topic (ACT) model is known [98] to power the ranking of publications, authors, and conferences in `AMiner`. The topics discovered by the ACT model are also used for filtering authors' publication lists and browsing the entire collection of publications. The use of the ACT model for interacting with publication lists in `AMiner` highlights some of the general drawbacks of the latent topic models for representing the document collections. First, the number of topics is fixed: two hundred topics are currently displayed to the user. Second, the labels of the topics have apparently been assigned manually, or at least with some human control. Most of the topic labels consist of two distinct phrases, for example “Database Systems / Programming Languages” or “Information Systems Development / Knowledge Management”: this suggests that most of the discovered topics, though probably coherent internally, do not correspond to single human-recognizable concepts.

2.2.4 Wikipedia topics

In this work we propose structuring the academic search results based on the topics derived from Wikipedia. Wikipedia is a large online encyclopedia that is collaboratively edited by Web users. Containing over 4 million articles in English alone, Wikipedia covers a broad range of subjects with considerable level of detail. Wikipedia articles can be viewed as fine-grained topics. Recently developed techniques [19, 29, 63, 66] allow annotating texts with links to Wikipedia articles, providing the basis for grouping texts according to the mentioned article-topics. The articles in Wikipedia are arranged into categories, which can be viewed as higher-level topics. The titles of both articles and categories are short, meaningful, and self-contained, being ideally suited for topic labeling. Furthermore, the categories are arranged into higher-level categories, forming a subsumption hierarchy.

Overall, the network of categories and articles of Wikipedia can be viewed as extremely large, fine-grained, constantly collaboratively updated taxonomy of topics. The hierarchical relations between the topics allow viewing the network at the desired level of granularity. Using Wikipedia topics for arranging the documents therefore combines most of the advantages and avoids most of the drawbacks of both predefined taxonomies and unsupervised learning methods. The Wikipedia-based structured representation proposed in this thesis (Chapter 3) is *automatic*, *dynamic*, *supervised*, meaningfully-*labeled*, *soft*, and with *flexible* granularity.

Wikipedia has been used in a number of works on search result clustering, though in ways different from our methods. Few works proposed improving the results of various clustering algorithm with the help of Wikipedia. Săcărea et al. [88] exploited redirects and disambiguation pages of Wikipedia to improve the clustering algorithm based on the formal concept analysis. In the work of Calli et al. [12] semantic relations derived from Wikipedia were used to improve the performance of the Suffix Tree Clustering algorithm.

A few works proposed organizing the search results based on the topics derived from Wikipedia articles. In the method of Han and Zhao [34] the relevant articles are arranged into a graph, with links representing the pairwise semantic relatedness, and the topics are defined as communities in this graph. Similarly, Scaiella et al. [89] consider a graph, which contains documents (search result snippets) in addition to articles, and discovers topics through spectral clustering on that graph. We give a more detailed description of [89] in Section 3.5.2. In both methods topics are represented by sets of Wikipedia articles, and an additional phase is needed to select the best article from each topic to provide the topic label. In contrast to these works, we use both articles and categories of Wikipedia to represent topics. The hierarchical relations between the topics provide for dynamically choosing the desired level of granularity. Additionally, in contrast to these works, we rely on structured prediction to summarize the document-topic graph, which allows us to jointly learn the useful combination of parameters, rather then tuning them by hand.

2.3 Supervised learning

The notation used in this and the following sections is summarized in Appendix A.

The goal of the supervised learning is to infer a mapping

$$\pi : \mathcal{X} \to \mathcal{Y},$$

given a sample of input-output pairs

$$D = \{(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{n},$$

where $\mathcal{X}$ and $\mathcal{Y}$ denote the respective spaces of inputs and outputs. In different contexts we will refer to mapping π as the *prediction function*, *classifier* or *policy*.

Depending on the nature of the outputs $\mathcal{Y}$ we can distinguish special cases of the learning problem. In **binary classification**, $\mathcal{Y}$ contains only two elements $\{Y_+, Y_-\}$, corresponding to the *positive* and the *negative* class. While the specific values of $\{Y_+, Y_-\}$ do not matter, a common choice is to encode $Y_+ = 1, Y_- = -1$. In **multiclass classification** $\mathcal{Y}$ contains K classes, which can be (arbitrarily) encoded as $1 : K$. Another common supervised learning task is **sequence labeling**, in which an input is a sequence $X = x_{1:T}$. The output assigns a label $y_i \in 1 : K$ out of K possible labels to every element x_i in the sequence. Every output is thus a sequence of labels $Y = y_{1:T}$, and the space $\mathcal{Y}$ contains $|\mathcal{Y}| = K^T$ different outputs. **Learning to rank** is another interesting problem. In this problem there exist k distinct elements $V = v_{1:K}$, and the outputs correspond to different ways to order these elements. Each output can be seen as a permutation $Y = \sigma(v_{1:K}), \sigma \in \mathfrak{S}_K$, with the number of distinct outputs being $|\mathcal{Y}| = K!$. An example of the ranking problem is predicting the order of the search results for a given query. In general the outputs $\mathcal{Y}$ can be arbitrary structures, such as vectors, sets, or graphs. In Section 3.4.1 we describe a problem, in which both inputs X and outputs Y are graphs.

2.3.1 Inference problem

One question in supervised learning is how to define a function π that can return complex structured outputs. This is usually done in the following way:

Definition 2 (inference)**.**

$$\pi(X) := \operatorname*{argmax}_{Y \in \mathcal{Y}} F(X, Y). \tag{2.1}$$

Computing the right part in (2.1) is called *inference* or *prediction*. The expression $F(X, Y)$ can be seen as a score that measures the compatibility between input X and output Y. The mapping π in the definition (2.1) returns the best output, which is most compatible with the given input.

In problems like binary and multiclass classification (with small number of classes), one can compute $\pi(X)$ by simply evaluating F on all possible outputs. With structured outputs, such as sequences and graphs, the possible outputs are too many to enumerate. The efficient computation of argmax is possible when the scoring function F *decomposes* over the structure of the outputs. An example of such decomposition in the sequence labeling task is the *Markov assumption*: when the score F of the complete output $Y = y_{1:T}$ equals the sum of partial scores F_t, each of which involves only d adjacent elements:

$$F(X, y_{1:T}) \equiv \sum_{t=0}^{T-d} F_t(X, y_{t+1:t+d}). \tag{2.2}$$

With Markov assumption (2.2) the argmax in 2.1 can be computed via dynamic programming in $O(T \cdot K^d)$, where K is the number of possible labels for each element. In the simplest case when $d = 1$ the predictions of individual labels y_t can be made independently. In the worst case the complexity of computing argmax is exponential in the size of the output, rendering the inference problem (2.1) infeasible.

2.3.2 Learning linear models

Most widely used is the class of *linear* functions F parametrized by weight vectors $\mathbf{w}$:

$$F[\mathbf{w}](X, Y) := \langle \mathbf{w}, \Psi(X, Y) \rangle. \tag{2.3}$$

Here $\mathbf{w}$ and $\Psi(X, Y)$ are vectors in $\mathbb{R}^M$, and $\langle \cdot \rangle$ denote the scalar product. The vector-function $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^M$ is called the *feature map*. The components of the vector $\Psi(X, Y)$ (*features*) represent different measures of compatibility between X and Y, and the function $F[\mathbf{w}]$ linearly combines these measures. With linear scoring function $F[\mathbf{w}]$, the linear prediction function π (2.1) is uniquely defined by the weight vector $\mathbf{w}$:

$$\pi[\mathbf{w}](X) := \operatorname*{argmax}_{Y} \langle \mathbf{w}, \Psi(X, Y) \rangle \tag{2.4}$$

In our work we only use linear prediction functions $\pi[\mathbf{w}]$. Learning a linear function reduces to learning the weight vector $\mathbf{w}$.

Empirical risk minimization. Given a sample D of input-output pairs, we want to learn a function $\pi[\mathbf{w}]$ that predicts outputs given inputs. Defining this formally requires a way to measure the quality of the prediction. Let $L(X, \mathbf{w}, Y)$ be the *loss* of the policy $\pi[\mathbf{w}]$ on the example (X, Y), measuring how bad it is to predict $\pi[\mathbf{w}](X)$ instead of Y. Consider the weights $\mathbf{w}$ that deliver minimum to the aggregated loss over the dataset D:

Definition 3 (Empirical Risk Minimization).

$$\mathbf{w}[L, D] := \operatorname*{argmin}_{\mathbf{w}} \sum_{(X,Y)\in D} L(X, \mathbf{w}, Y). \tag{2.5}$$

The rule (2.5) for finding **w** is called *empirical risk minimization.*

We want the classifier $\pi[\mathbf{w}]$ to *generalize* well—that is, be able to make good prediction on unseen inputs. The necessary condition for that is that the learning rule for the vector **w** be robust to small changes in the dataset D. In order to achieve this, a *regularization term* $R(\mathbf{w})$ is added to the minimization objective (2.5), resulting in the following rule of

Definition 4 (Regularized Empirical Risk Minimization).

$$\mathbf{w}[L, R, D] := \operatorname*{argmin}_{\mathbf{w}} \left(\frac{1}{|D|} \sum_{(X,Y)\in D} L(X, \mathbf{w}, Y) + \lambda R(\mathbf{w}) \right). \tag{2.6}$$

The term $R(\mathbf{w})$ prevents the norm of the vector **w** from growing indefinitely, and the parameter λ trades off the loss on the training set with the size of **w**. One typical specific choice for the regularization function is the squared L_2-norm:

$$R(\mathbf{w}) = \frac{1}{2}\|w\|_2^2. \tag{2.7}$$

The loss functions. The most natural choice of the loss function L is the **zero-one loss**, which measures if the prediction output matches the true output exactly:

$$L_{0/1}(X, \mathbf{w}, Y) := \Delta_{0/1}(Y, \pi[\mathbf{w}](X)) := \mathbb{1}(\pi[\mathbf{w}](X) \neq Y). \tag{2.8}$$

(We use symbol Δ to define distances, which are functions of pairs of outputs. In contrast, we use L for losses, which are functions of an input, a weight vector, and a correct output.) The zero-one loss is, unfortunately, non-convex in **w**, rendering the learning problem (2.6) NP-hard [28]. In addition, being discrete, $L_{0/1}$ is not continuous. In practice various convex upper-bounds to the zero-one loss are used. One of such functions is the **hinge loss**. Let Δ be some dissimilarity function defined on the pairs of outputs:

$$\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+. \tag{2.9}$$

It is not required that Δ be a metric, only that it is bounded and $\Delta(Y, Y) \equiv 0$. The hinge loss $L_h[\Delta]$ is defined in the following way:

Definition 5 (Hinge loss).

$$L_h[\Delta](X, \mathbf{w}, Y) := \max_{Y'\in\mathcal{Y}} (\Delta(Y, Y') + \langle \mathbf{w}, \Psi(X, Y') - \Psi(X, Y)\rangle) \tag{2.10}$$

$L_h[\Delta]$ encodes the soft requirement that the true output Y should score higher than any other output Y' by at least $\Delta(Y, Y')$. This also means that the difference in scores should be higher the more dissimilar Y' is from Y. One can show that $L_h[\Delta](X, \mathbf{w}, Y)$ is an upper bound for $\Delta(Y, \pi[\mathbf{w}](X))$:

$$\begin{aligned}\Delta(Y, \pi[\mathbf{w}](X)) &\le \Delta(Y, \pi[\mathbf{w}](X)) + \max_{Y' \in \mathcal{Y}} \langle \mathbf{w}, \Psi(X, Y')\rangle - \langle \mathbf{w}, \Psi(X, Y)\rangle \\ &= \Delta(Y, \pi[\mathbf{w}](X)) + \langle \mathbf{w}, \Psi(X, \pi[\mathbf{w}](X))\rangle - \langle \mathbf{w}, \Psi(X, Y)\rangle \\ &\le \max_{Y' \in \mathcal{Y}} \left(\Delta(Y, Y') + \langle \mathbf{w}, \Psi(X, Y') - \Psi(X, Y)\rangle\right) \\ &= L_h[\Delta](X, \mathbf{w}, Y).\end{aligned}$$

Definition 5 (2.10) is sometimes called the hinge loss with *margin rescaling* [100]. There is an alternative definition of the hinge loss with *slack rescaling* [100]:

$$L'_h[\Delta](X, \mathbf{w}, Y') := \max_{Y' \in \mathcal{Y}} \left(\Delta(Y, Y')(1 + \langle \mathbf{w}, \Psi(X, Y') - \Psi(X, Y)\rangle)\right). \tag{2.11}$$

The slack-rescaling version of the hinge loss also enjoys the properties of convexity and being an upper bound for Δ.

Support Vector Machine (SVM). The standard SVM approach [100, 101] minimizes the regularized empirical risk with hinge loss and squared L_2 regularization[3]. Variations of the SVM algorithms exist for different types of outputs, including SVM for binary classification, ranking SVM, and SVM for structured outputs. The learning problem (2.6) for SVM can be re-formulated as quadratic minimization problem with $n \cdot |\mathcal{Y}|$ constraints:

$$\begin{aligned}&\mathbf{w}[L, R, D] := \operatorname*{argmin}_{\mathbf{w}, \xi_i \ge 0} \left(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i\right), \\ &s.t. \quad \Delta(Y_i, Y') + \langle \mathbf{w}, \Psi(X_i, Y') - \Psi(X_i, Y_i)\rangle \le \xi_i, \quad \forall i \in 1{:}n,\ \forall Y' \ne Y_i\end{aligned} \tag{2.12}$$

For complex output spaces the number of constraints is prohibitively large[4]. State of the art *cutting plane* algorithms [45, 46] reduce the number of constraints that have to be dealt with, but require the “loss-augmented search” to be efficiently solvable:

$$\operatorname*{argmax}_{Y' \in \mathcal{Y}} \left(\Delta(Y, Y') + \langle \mathbf{w}, \Psi(X, Y')\rangle\right). \tag{2.13}$$

When Δ “decomposes” over the structure of the outputs Y (see [22] for the formal definition), the problem (2.13) is as “easy” as the inference problem (2.1). The loss-augmented search has to be solved about $O(n)$ times during learning, where n is the size of the training set. This illustrates that the learning problem is typically much harder than the prediction problem, and is thus not feasible for general outputs and loss functions Δ. We will next describe two cases important for our work, in which learning *is* tractable.

[3]There exist variations of the SVM that use *squared* hinge loss, and/or L_1 regularization.

[4]The actual SVM algorithms do not solve the exact problem (2.12), but re-formulate it in various ways that admit more efficient solutions. We use the number of constraints in (2.12) only as an illustration for the complexity of the problem.

2.3.3 Specific cases of supervised learning

Binary classification Binary classification is perhaps the simplest learning problem. There exist only two classes (outputs), which can be arbitrarily coded as:

$$\mathcal{Y} = \{-1, 1\}.$$

Inference (2.1) reduces to evaluating a single dot product:

$$\pi[\mathbf{w}](X) := sign(\langle \mathbf{w}, \Phi(X) \rangle).$$

The simplified feature map $\Phi(X)$ depends only on the input X[5] . With the two possible outputs, the zero-one loss becomes the only sensible loss function.
The hinge approximation to the zero-one loss in the binary case can be rewritten as:

$$L_h[\Delta_{0/1}](X, \mathbf{w}, Y) = \max(0, 1 - Y\langle \mathbf{w}, \Phi(X) \rangle) \tag{2.14}$$

The SVM learning problem thus simplifies to:

$$\begin{aligned} \mathbf{w}[L, D] &:= \operatorname*{argmin}_{w, \xi_i \geq 0} \left(\frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right), \\ s.t. \quad & Y_i \cdot \langle \mathbf{w}, \Phi(X_i) \rangle \geq 1 - \xi_i, \quad \forall i \in 1\!:\!n, \end{aligned} \tag{2.15}$$

where C is a regularization constant inversely proportional to λ in (2.6). This is an easy problem with only n constraints, which is linear in the size of the training set.

Learning to rank. A ranking $Y = y_{1:K}$ is a permutation of some elements $V = v_{1:K}$:

$$y_{1:K} = \sigma(v_{1:K}), \; \sigma \in \mathfrak{S}_K. \tag{2.16}$$

Alternatively, a ranking Y can be viewed as an order relation $\succ_Y$ on the set of elements V, such that $v \succ_Y v'$ whenever v is higher than v' in the ranking Y. A ranking function π takes X as an input and produces the ranking Y as an output.

Consider the following family of ranking functions $\pi[\mathbf{w}]$ parametrized by the weight vectors $\mathbf{w}$, such that

$$v \succ_{\pi[\mathbf{w}]} v' \iff \langle \mathbf{w}, \phi(X, v) - \phi(X, v') \rangle > 0, \tag{2.17}$$

where ϕ is a feature map defined jointly on the inputs X and the elements y_i of the output. Condition (2.17) specifies that in the ranking $\pi[\mathbf{w}](X)$ the elements $v \in V$ are ordered by the decreasing value of $\phi(X, v)$. This way of predicting the ranking Y corresponds to a specially designed $\Psi(X, Y)$ in the inference problem 2.1. Prediction for a given X is easy, as it only requires evaluating ϕ for K different values.

[5] The inference rule for binary classification can be derived from the general (2.1) by setting $\Phi(X) := \Psi(X, 1) - \Psi(X, -1)$ and simplifying the argmax over the two possible outputs.

The learning problem 2.12 can be re-formulated in terms of the elements v (rather than outputs Y) as well. A ranking Y can be thought of as a function that predicts the relative ordering of the pairs of elements $\{v, v'\}$:

$$Y(v, v') := \begin{cases} 1 & \text{if } v \succ_Y v', \\ -1 & \text{if } v' \succ_Y v. \end{cases}$$

Consider the loss function $L_{inv}(X, \mathbf{w}, Y)$ that measures the number of pairs $\{v, v'\}$ that are ordered differently by the rankings Y and $\pi[\mathbf{w}](X)$:

$$L_{inv}(X, \mathbf{w}, Y) := \sum_{\{v,v' \in V \,|\, v' \neq v\}} L_{0/1}((X, v, v'), \mathbf{w}, Y(v, v')). \tag{2.18}$$

The loss L_{inv} decomposes into the zero-one losses of predicting the relative ordering of individual pairs $\{v, v'\}$. Similarly to the binary SVM classification, the ranking SVM replaces the zero-one loss $L_{0/1}$ with its hinge approximation $L_h[\Delta_{0/1}]$. The learning problem becomes as follows:

$$\begin{aligned} \mathbf{w}[L, D] &:= \operatorname*{argmin}_{w, \xi_{ijk} \geq 0} \left(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{2C}{n \cdot K(K-1)} \sum_{i=1}^{n} \sum_{\{j,k \in 1:K, \,|\, j \neq k\}} \xi_{ijk} \right), \\ s.t. \quad & \langle \mathbf{w}, \phi(X_i, v_j) - \phi(X_i, v_k) \rangle \geq 1 - \xi_{ijk}, \quad \forall i \in 1\!:\!n,\ v_j \succ_{Y_i} v_k \end{aligned} \tag{2.19}$$

The formulation (2.19) is the same as that for binary classification (2.15), except the inputs are triples (X_i, v_j, v_k), with the feature map $\Phi(X_i, v_j, v_k) := \phi(X_i, v_j) - \phi(X_i, v_k)$. This is a relatively simple quadratic optimization problem with $O(n \cdot K^2)$ constraints.

Predicting the top-ranked item. The problem (2.19) is formulated in terms of the pairwise preferences between the elements $v \in V$, rather than complete rankings Y. This allows training the ranking SVM with only partial information about preferences. In an important special case, the ranking SVM can be trained to predict the "optimal" (highest-ranked) element, without further distinction between the rest of the elements. In this case, the problem (2.19) will contain only $n \cdot K$ constraints of the form

$$\langle \mathbf{w}, \phi(X_i, y_{i1}) - \phi(X_i, y_{ij}) \rangle \geq 1 - \xi_{i1j},$$

where $(y_{i1}, y_{i2}, \ldots, y_{iK})$ is the order of the elements according to the ranking Y_i, with (X_i, Y_i) being a single training example. This setting arises naturally when the training data contain only the best element for each input rather than the complete ranking. Similar to this case is the formulation of the multiclass SVM with K classes [18].

General structured outputs. As we mentioned earlier, the exact inference (2.1) is only feasible when the features $\Psi(X, Y)$ decompose over the structure of the output Y. In particular, the exact inference is *not feasible* for the problem of summarizing the topic maps, which we formulate in Section 3.4.1. A number of recently developed algorithms

[23, 24, 35, 85, 86, 87] define a novel approach of *learning to search*, in which the structured outputs are predicted sequentially. These algorithms avoid the exact computation of argmax, while providing good theoretical guarantees for the prediction quality. In the work we describe in Chapters 3 and 5 we extensively rely on the DAGGER algorithm [87], which views structured prediction as an *imitation learning* problem. DAGGER reduces *imitation learning* and structured prediction to *no-regret online learning.* In the following we describe the algorithm of DAGGER and its guarantees, after giving some background on the *online learning* and *imitation learning.* The material in Sections 2.4–2.6 will be particularly useful for understanding the contribution of Chapter 5.

2.4 Online learning

In **online learning** the learner interacts with the possibly adversarial environment. At each iteration i the learner generates a hypothesis, in our case a policy π^i, and the environment responds with a loss function L^i_{onl}. The learner suffers the loss $L^i_{onl}(\pi^i)$ and proceeds to the next iteration. The goal of the learner is to minimize the regret

$$Regret(\pi^{1:N}) := \sum_{i=1}^{N} L^i_{onl}(\pi^i) - \min_{\pi} \sum_{i=1}^{N} L^i_{onl}(\pi). \tag{2.20}$$

The first term in the definition of regret (2.20) is the total loss suffered by the learner over N iterations, while the second term is the minimum attainable loss of any fixed hypothesis π with respect to the loss functions $L^{1:N}_{onl}$.

Definition 6 (No-regret online learning algorithm)**.** *The learning algorithm is* no-regret *if the average regret tends to zero for any sequence of loss functions $L^{1:N}_{onl}$:*

$$\lim_{N\to+\infty} \frac{1}{N} Regret(\pi^{1:N}) \leq 0. \tag{2.21}$$

With certain assumptions on $L^{1:N}_{onl}$ various algorithms are known to be no-regret.

Follow-The-Leader (FTL) algorithm chooses the next hypothesis by minimizing the loss over all previous iterations:

$$\pi^{i+1}_{FTL} := \operatorname*{argmin}_{\pi} \sum_{j=1}^{i} L^j_{onl}(\pi). \tag{2.22}$$

Assuming the functions $L^{1:N}_{onl}$ are *strongly convex*, the average regret of Follow-the-Leader is known [91] to be

$$\frac{1}{N} Regret(\pi^{1:N}_{FTL}) = O\left(\frac{log\,(N)}{N}\right). \tag{2.23}$$

Follow-the-Regularized-Leader (FTRL) is a modification of FTL that minimizes the loss over the previous iterations *plus a regularization term*:

$$\pi^{i+1}_{FTRL} := \operatorname*{argmin}_{\pi} \left(\sum_{j=1}^{i} L^{j}_{onl}(\pi) + \lambda_i R(\pi) \right). \tag{2.24}$$

FTRL is known to be no-regret [91] under a milder assumption, but with a slightly worse guaranteed convergence speed. Specifically, assuming the regularization function R is *strongly convex*, the loss functions $L^{1:N}_{onl}$ are just *convex* and the regularization constants λ_n are $\Theta(N)$, the average regret of FTRL is:

$$\frac{1}{N} Regret(\pi^{1:N}_{FTRL}) = O\left(\frac{1}{\sqrt{N}}\right). \tag{2.25}$$

Implementation of FTRL with data aggregation. An example of the FTRL implementation important for our work is when the hypotheses π^i represent linear classifiers, and the loss functions $L^i_{onl}(\pi)$ represent the loss of the classifier π on some dataset D^i. In particular, let $\pi[\mathbf{w}]$ be a linear classifiers parametrized by the weight vectors $\mathbf{w}$. Let $l(X, \mathbf{w}, y)$ be some loss function computed for the input-output pair (X, y). We will the refer to l as the *local loss*. Consider the online learning scenario, in which every loss function L^i_{onl} is defined as the aggregation of the loss l over some dataset D^i:

$$L^i_{onl}(\pi[\mathbf{w}]) := \sum_{(X,y)\in D^i} l(X, \mathbf{w}, y). \tag{2.26}$$

Proposition 1 (FTLR with data aggregation)**.** *Implementing the iteration $i+1$ of FTRL with loss functions (2.26) amounts to training a classifier $\pi[\mathbf{w}]$ that minimizes the regularized empirical risk with respect to the loss l on the aggregated dataset $\bigcup_{j=1}^{i} D^j$.*

In order for the data-aggregating implementation of the FTRL to enjoy the convergence guarantees (2.25), it is sufficient to pick the convex local loss function $l(X, \mathbf{w}, y)$ and strongly convex regularizer R. In particular, the guarantee holds for the specific case when we use the standard L_2-regularized SVM to train the policies $\pi^{1:N}$ at each iteration. The latter case corresponds to using $L_h[\Delta](X, \mathbf{w}, y)$ as the local loss function.

2.5 Imitation learning

In **imitation learning**, we observe the expert interacting with the environment, and aim to mimic the expert's behavior. Formally, let $\mathcal{S}$ be the space of *states*, and $\mathcal{A}$—the state of *actions*. For every state $S \in \mathcal{S}$, the set $actions(S) \subseteq \mathcal{A}$ defines the actions possible in this state. Each possible action $a \in actions(S)$ transforms the state S into a new state $a(S) \in \mathcal{S}$. The expert interacts with the environment in rounds. Each time the interaction starts from some initial state S_0. The expert takes the first action $a_1 \in \mathit{actions}(S_0)$, which produces the new state $S_1 = a_1(S_0)$, then the expert takes the next action $a_2 \in \mathit{actions}(S_1)$, and so on. We observe the resulting *trajectory*, a sequence of state-action pairs:

$$(S_0, a_1), (S_1, a_2), \dots, (S_{T-1}, a_T). \tag{2.27}$$

We view the expert as a *policy* π^* that defines the mapping from states to actions:

$$\pi^* : \mathcal{S} \to \mathcal{A}.$$

The goal of imitation learning is to find a policy π that mimics the expert policy π^*. As usual, we are considering only linear classifiers $\pi[\mathbf{w}]$ parameterized by weight vectors $\mathbf{w}$. Let $l(S, \mathbf{w}, a^*)$ denote a local loss associated with taking the action $\pi[\mathbf{w}](S)$ instead of the optimal a^* in state S.

Definition 7. *The **imitation loss** is the aggregate loss suffered by the policy π with respect to the optimal policy π^* over the trajectory generated by $\pi[\mathbf{w}]$:*

$$L_{imit}(\pi[\mathbf{w}]) := \sum l(S_t, \mathbf{w}, \pi^*(S_t)), \quad S_t \sim \pi[\mathbf{w}]. \tag{2.28}$$

Here and below we use the notation $S_t \sim \pi$ (or $a_t \sim \pi$) to denote that the corresponding states (or actions) have been produced by the policy π.
We seek the policy π that minimizes the imitation loss:

$$\hat{\pi} := \underset{\pi}{\operatorname{argmin}}\, L_{imit}(\pi). \tag{2.29}$$

It is important to note that $\hat{\pi}$ in equation (2.29) minimizes the aggregate loss over its own-produced sequence of states, rather than over the states produced by the expert π^*.

Why is imitation learning difficult? The trickiness of the imitation learning problem (2.29) is that decisions a_t are taken sequentially, with earlier decisions affecting the state of the system at further steps. In particular, this means that the state-action pairs (2.27) in the training set cannot be treated as independent. Consider a naïve *supervised learning* approach that constructs a policy π by training some standard supervised learning method on all the state-action pairs in the training set D:

$$\pi_{sup} := \underset{\pi[\mathbf{w}]}{\operatorname{argmin}} \sum_{(S,a) \in D} l(S, \pi[\mathbf{w}], \pi^*(S)). \tag{2.30}$$

At test time, when executing π_{sup} on the new examples, any mistakes made by π_{sup} at step t may bring it into the states it has not observed during training. As most supervised learning methods are guaranteed to work only under the same distribution of training and test data, π_{sup} may perform arbitrarily badly in its own-induced states. Another way to put this is that the policy π_{sup} will not be able to recover from its own mistakes. It has been shown in [85] that the loss $L_{imit}(\pi_{sup})$ may be as bad as $O(T^2)$, where T is the length of the trajectory. This motivates the more advanced approaches to imitation learning.

2.6 DAgger: imitation learning as no-regret online learning

The DAgger (Dataset Aggregation) framework [87] reduces the problem of imitation learning to *no-regret online learning.* The policy is built iteratively by training every next policy π^i on the states produced by all previous policies $\pi^{0:i-1}$. In the limit, the policy $\pi^\infty := \lim_{N\to\infty} \pi^N$ has been trained on its own-produced states and is thus robust to its own mistakes. Under certain assumption on the local loss l, the algorithm of DAgger provides a policy whose imitation loss scales linearly in the number of steps T (rather than quadratically like the naïve supervised learning method (2.30)).

2.6.1 The algorithm of DAgger and its guarantees

We will describe a version of the DAgger algorithm and sketch a proof of its guarantees (taken from [84][6]). DAgger is an iterative algorithm that produces a new policy π^i at each iteration. DAgger maintains a dataset D of state-action pairs, in which the states have been generated by all the previous policies $\pi^{0:i-1}$, and the actions are generated by the expert π^*. At i_{th} iteration, a new policy π^i is trained on the current version of the dataset D. The policy π^i is executed, producing the trajectory

$$S_0, S^i_1, S^i_2, \ldots, S^i_{T_i} = S^i_{1:T_i} \sim \pi^i.$$

For each state S^i_t the *expert action* $\pi^*(S^i_t)$ is generated, and the state-action pair $(S^i_t, \pi^*(S^i_t))$ is added to the dataset D. The listing of Algorithm 1 describes the algorithm formally.

Algorithm 1: DAgger algorithm

```
1  $D^0 \leftarrow \varnothing$ // empty set
2  $\pi^0 \leftarrow \pi^*$ // expert policy
3  for $i \in 0, 1 \ldots, N-1$ do
4      Generate the sequence $(S^i_0, S^i_1, \ldots, S^i_{T_i}) \sim \pi^i$
5      Generate expert actions $\{\pi^*(S^i_t)\}_{t=0}^{T_i-1}$
6      Build the dataset $D^{i+1} \leftarrow \{(S^i_t, \pi^*(S^i_t))\}_{t=0}^{T_i-1}$
7      Aggregate the datasets: $D \leftarrow D \cup D^{i+1}$
8      Train the policy $\pi^{i+1}$ on the dataset $D$
9  end
10 return $\pi_N$
```

Let us analyze the properties of the DAgger algorithm in the case when the policies $\pi^{1:N}$ are trained by minimizing the regularized empirical risk with a convex loss function l and a strongly convex regularizer (e.g. L_2-regularized ranking SVM). For simplicity,

[6]The framework of DAgger, as described in [84], is quite general, and its theoretical guarantees come in a number of related theorems that make different specific assumptions. We will present a specific formulation of the algorithm and the theorems that are most suitable for our case.

assume $T^i = T,\ \forall i \in 1 : N$. Let $L^i_{class}(\pi[\mathbf{w}])$ denote the per-state loss of the classifier $\pi[\mathbf{w}]$ on the states produced by the policy π^i:

$$L^i_{class}(\pi[\mathbf{w}]) := \frac{1}{T}\sum_{t=0}^{T-1} l(S^i_t, \mathbf{w}, \pi^*(S^i_t)). \tag{2.31}$$

Note that for the DAgger policies the classifier loss equals the normalized imitation loss:

$$L^i_{class}(\pi^i) \equiv \frac{1}{T} L_{imit}(\pi^i).$$

We can also note that, with the described classifier (minimizing the regularized empirical risk), Algorithm 1 can be viewed as training an *online learner*, namely Follow-the-Regularized-Leader, in which $L^{1:N}_{class}$ are the loss functions generated by the environment:

$$L^i_{onl}(\pi) \equiv L^i_{class}(\pi).$$

The last expression holds for any policy π.

Consider the average imitation loss suffered by the policies $\pi^{1:N}$. Following the argument in [87], any upper bound on the average loss will also hold for the loss of the best policy in $\pi^{1:N}$. Up to a factor $\frac{1}{T}$, the average imitation loss of the policies $\pi^{1:N}$ is equal to the total loss of the online learner:

$$\frac{1}{T}\cdot\frac{1}{N}\sum_{i=1}^{N} L_{imit}(\pi^i) = \frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi^i). \tag{2.32}$$

The latter decomposes as follows:

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi^i) &= \min_\pi \frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi) + (\frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi^i) - \min_\pi \frac{1}{N}\sum_{i=1}^{N} L^i_{class})\\ &= \min_\pi \frac{1}{N}\sum_{i=i}^{N} L^i_{class}(\pi) + \frac{1}{N} Regret(\pi^{1:N})\\ &= \epsilon_{class} + \epsilon_{regret},\end{aligned} \tag{2.33}$$

where

$$\epsilon_{class} := \min_\pi \frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi)$$

is the minimum attainable classifier loss on the states produced by the policies $\pi^{1:N}$, and

$$\epsilon_{regret} := \frac{1}{N}\sum_{i=1}^{N} L^i_{class}(\pi^i) - \epsilon_{class}$$

is the average regret of the online learner. Combining (2.32) with (2.33) we get

$$\frac{1}{N}\sum_{i=1}^{N} L_{imit}(\pi^i) = T\cdot(\epsilon_{class} + \epsilon_{regret}). \tag{2.34}$$

Theorem 1 (Convergence of DAgger (loosely, from [84]))**.** *For a convex loss function l, a strongly convex regularizer R, and any $\epsilon > 0$, after $N = O(T^2/\epsilon^2)$ iterations of DAgger, there exists a policy $\pi^j \in \pi^{1:N}$, such that*

$$L_{imit}(\pi^j) = T \cdot \epsilon_{class} + O(\epsilon).$$

Proof. The proof follows from the decomposition (2.34) of the average imitation loss, and the no-regret properties of Follow-the-Regularized-Leader (2.25). □

It follows from Theorem 1 that the best policy across all iterations of DAgger has the imitation loss that tends to the minimum attainable classifier loss on the states visited at all iterations times T. In cases when the expert policy π^* is *realizable*, that is, when there exists a weight vector $\mathbf{w}^*$ such that $\pi^* = \pi[\mathbf{w}]$, the loss ϵ_{class} may equal zero. The guarantees similar to those of Theorem 1 can be obtained for the general case when any no-regret online learning algorithm is used for training the policies $\pi^{1:N}$.

2.6.2 Sequential prediction of structured outputs

Structured prediction can be viewed as imitation learning, if the outputs are constructed *sequentially.* First, we need to represent a structured output Y as a sequence:

$$Y \iff y_{1:T}.$$

In the straightforward case, the elements y_t of the sequence may represent parts of the structured output Y. For instance, if Y is a list, y_t may be its t_{th} element. More generally, we may define a search procedure that moves in the space of partial outputs and returns Y upon completion, with y_t representing the moves in this space, or even the steps of the search procedure itself (e.g scheduling a node for traversal in beam search). This general view explains why sequential prediction is sometimes called *learning to search.*

Learning to predict Y given X can be reduced to learning to predict the "action" y_{t+1} given the "state" $(X, y_{1:t})$:

$$y_{t+1} := \pi\left(X, y_{1:t}\right).$$

For a new input X the output Y can be built sequentially, by first applying π to $(X, ())$ to obtain y_1, then applying π to $(X, (y_1))$, and so on until we get the full sequence $y_{1:T}$.

The algorithm of DAgger described in Section 2.6 can be used to train the policy π so that it behaves well in its own-produces states. A few ingredients are needed in order to apply DAgger to structured prediction.

Training the classifier $\pi[\mathbf{w}]$. First, we need a way to train the classifier π on the aggregated datasets of state-action pairs (line 8 of Algorithm 1). With regularized empirical risk minimization, it is equivalent to choosing the local loss l and the regularization function R. The value $l((X, y_{1:t}), \mathbf{w}, \hat{y}^*_{t+1})$ corresponds to the cost of predicting the action

$y_{t+1} = \pi[\mathbf{w}](X, y_{1:t})$ in state $(X, y_{1:t})$ instead of the optimal, expert's action $\hat{y}^*_{t+1}$. Suppose that for some input X the true output Y^* decomposes into $y^*_{1:T}$, and the predicted Y—into $y_{1:T}$. The local loss l should be such that low total imitation loss

$$L_{imit}(\pi[\mathbf{w}]) = \sum_{t=0}^{T-1} l((X, y_{1:t}), \mathbf{w}, \hat{y}^*_{t+1})$$

would result in low target loss $L(Y, Y^*)$. In some cases, it is possible to construct l so that $L_{imit}(\pi[\mathbf{w}])$ exactly equals $L(Y, Y^*)$. For instance, when outputs are sequences, the zero-one local loss $L_{0/1}$ summed over the steps equals the Hamming loss between outputs.

In our work we use the L_2-regularized ranking SVM, whose loss $L_h[\Delta_{0/1}]$ is the hinge approximation to the zero-one loss. The loss function $L_h[\Delta_{0/1}]$ encodes the information that the optimal (expert's) action should be ranked higher than the non-expert's actions, while the relative ordering of the non-optimal actions does not matter.

Specifying the expert policy π^* is the second ingredient. For any given input X *in the training set*, and any partial, possibly non-optimal, sequence $y_{1:t}$ corresponding to this input, the expert policy has to produce the optimal action $\hat{y}^*_{t+1}$:

$$\hat{y}^*_{t+1} := \pi^*(X, y_{1:t}).$$

In pure imitation learning it is often assumed that the true expert (e.g. human) is available to provide the actions. For example, [87] reported on experiments in which DAgger was applied to learning to drive a racing car in a computer game. In those experiments a human first played the game to provide the initial trajectories, and then observed the driving produced by the policies of DAgger while continuing to "steer". This steering provided the actions the expert would have taken in the non-optimal states to which the DAgger policies brought the car.

When applying DAgger to structured prediction it is possible to implement the expert policy based on the training dataset alone, without the need of the human expert. Informally, the idea is that the expert actions should bring the current trajectory closer to the ground truth trajectory corresponding to the same input. This idea can formally encoded by specifying a distance function Δ on the partial trajectories of the same length:

$$\Delta(y_{1:t}, y'_{1:t}) \geq 0. \tag{2.35}$$

Δ does not have to be a metric. Consider an input X from the training set, and some partial non-optimal output $y_{1:t} = (y_1, y_2, \ldots, y_t)$. Consider also the ground truth sequence $y^*_{1:T} = (y^*_1, y^*_2, \ldots, y^*_T)$ corresponding to the input X.
The expert's action is formally defined in the following way.

Definition 8 (Expert action in sequential structured prediction)**.**

$$\pi^*(X, y_{1:t}) := \operatorname*{argmin}_{\hat{y}_{t+1}} \Delta((y_1, y_2, \ldots, y_t, \hat{y}_{t+1}), y^*_{1:t+1}). \tag{2.36}$$

When Δ is the Hamming distance, computing argmin in equation (2.36) is easy: $\pi^*(X, y_{1:t})$ is just equal to y^*_{t+1}. In more complex cases, such as the one we describe in Section 3.4.4, providing the expert actions requires nontrivial computations.

Chapter 3

Building structured topical summaries

This chapter introduces *topic maps*—a form of informative visual representation of document collections. In Section 3.2 we show how topic maps can be used for efficient visualization and browsing of the academic search results. In Section 3.3 we describe a method for building topic maps from the network of categories and articles of Wikipedia. We present a supervised learning method for producing informative maps with a given number of topics in Section 3.4, and report on the evaluation experiments in Section 3.5.

3.1 Topic maps

Topic map represents a collection of documents. It is a graph in which the nodes are topics, and the edges—relations between them (see Figure 3.1 for an example). In case of hierarchical relations, such as the sub-topic relation we use in this work, the graph is directed and acyclic. We will refer to the sub-topics of a topic as *children* and to the super-topics as *parents*. Each topic in the graph is relevant to a subset of the documents in the collection. We say that a topic *covers* the documents to which it is relevant. We assume that if a topic is relevant to a document, its parents are also relevant. Thus, a parent topic covers all the documents covered by its children and, possibly, some additional documents. An important property of the topics in the map is that they correspond to meaningful concepts and are represented by short meaningful titles.

Hierarchical topic maps are naturally visualized with a layered layout [31], as implemented, for instance, in the `graphviz` package [32]. The topic titles label the nodes in the displayed graph. To emphasize the connection between the topics and the documents, we display the number of covered documents in the parenthesis after the title. We use the font size proportional to the number of covered documents as an additional visual cue.

A topic map represents the collection of documents with respect to the underlying

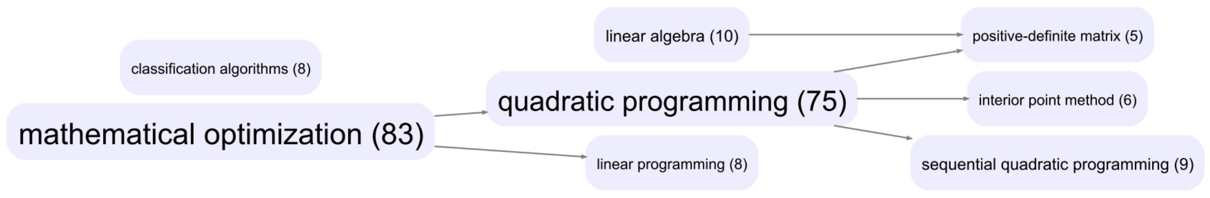

linear algebra (10)
positive-definite matrix (5)
classification algorithms (8)
quadratic programming (75)
interior point method (6)
mathematical optimization (83)
linear programming (8)
sequential quadratic programming (9)

Figure 3.1: A topic map summary the top 100 search results for the query "quadratic programming".

topics. It summarizes the collection visually, so that one can understand the high-level structure of the collection without having to examine every individual document. Additionally, a topic map can serve as a browsing or filtering tool: clicking on a certain topic in the map can retrieve the documents of the collection covered by this topic.

3.1.1 Quality of the topic maps

The quality of a topic map depends on several factors and is difficult to define precisely. On one hand, a good topic map should be *accurate*, displaying most topics occurring in the documents according to their importance, interestingness, and rate of occurrence. On the other hand, the topic map should minimize the cognitive load on the user, which requires it being *simple*, easily understandable, and visually appealing. Finally, the topic map should be *informative*, conveying maximum useful information about the structure of topics per unit of displayed data. We do not attempt to formalize these principles for building the topic maps. Instead, we formulate a number of properties that we think should correlate with the topic map quality, and use supervised machine learning in order to find the useful balance between these properties (see Section 3.4.1). Here we informally list some of these properties (many of them are related and overlap in meaning):

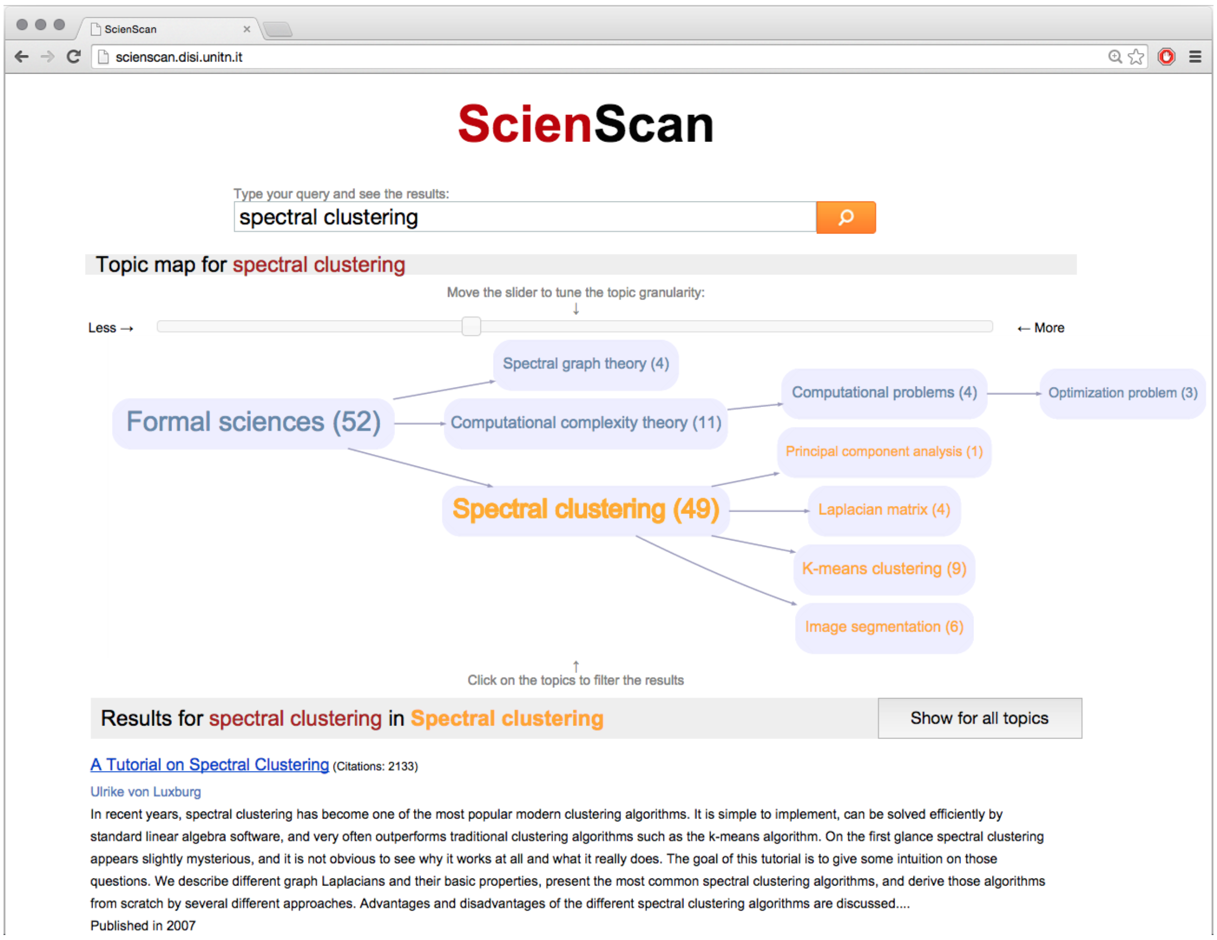
- *coverage*:
  - the displayed topics should be relevant to many documents in the collection,
  - most documents should be covered by one or more topics;
- *diversity*:
  - the topics should partition the documents into (soft) clusters with low overlap,
  - the topics should not be too similar to each other (by meaning or by title);
- *structure of the graph*:
  - the topics should be connected to show relations between them,
  - there should not be many isolated nodes in the graph,
  - there should not be many isolated components in the graph,
  - the topic graph should have a balanced tree–like structure:

    * the graph should not be similar to a chain (list-like) graph,
    * topics should have few parents (preferably one),
    * non-leaf topics should have several children;
  - the graph should have sufficient depth:
    * e.g., it should not be a bushy 1-level tree;
  - there topic graph should not be cluttered with too many links;

- the topics should specific, low-level (rather than general, high-level).

3.2 ScienScan, an academic search tool

Here we present ScienScan [70]—a prototype browsing and visualization tool for academic search that relies on topic maps for representing the search results. The tool operates in real time by post-processing the query results of a third-party academic search engine.


ScienScan
scienscan.disi.unitn.it
ScienScan
Type your query and see the results:
spectral clustering
Topic map for spectral clustering
Move the slider to tune the topic granularity:
Less →
← More
Spectral graph theory (4)
Formal sciences (52)
Computational complexity theory (11)
Computational problems (4)
Optimization problem (3)
Principal component analysis (1)
Spectral clustering (49)
Laplacian matrix (4)
K-means clustering (9)
Image segmentation (6)
Click on the topics to filter the results
Results for spectral clustering in Spectral clustering
Show for all topics

A Tutorial on Spectral Clustering (Citations: 2133)

Ulrike von Luxburg

In recent years, spectral clustering has become one of the most popular modern clustering algorithms. It is simple to implement, can be solved efficiently by standard linear algebra software, and very often outperforms traditional clustering algorithms such as the k-means algorithm. On the first glance spectral clustering appears slightly mysterious, and it is not obvious to see why it works at all and what it really does. The goal of this tutorial is to give some intuition on those questions. We describe different graph Laplacians and their basic properties, present the most common spectral clustering algorithms, and derive those algorithms from scratch by several different approaches. Advantages and disadvantages of the different spectral clustering algorithms are discussed....

Published in 2007

Figure 3.2: The interface of ScienScan.

ScienScan has an interface of a typical search engine (see Figure 3.2). Users type queries into the search box and obtain a list of the search results. In addition to the standard controls, ScienScan displays the *topic map* of the retrieved results. The topic map is built based on the titles and the abstracts of the retrieved papers in order to summarize the results in the most informative way (see Sections 3.3–3.4). When the user clicks on a topic in the hierarchy, the topic and its sub-topics are highlighted, and the displayed results get restricted to those covered by the selected topic. The user can control the number of nodes in the topic map with a slider. ScienScan builds multiple instances of the topic maps of various sizes, and moving the slider switches between these instances, making the displayed topic map grow or shrink visually. The algorithms behind ScienScan are implemented in such a way that bigger topic maps are built incrementally from smaller ones. This makes the computations efficient, and ensures that topics do not disappear from the map when the map size is increased.

3.2.1 Implementation

When the user submits the search query, ScienScan performs the following steps:

1. forward the query to an existing search engine and collect the results,
2. build the topic map of the retrieved results,
3. summarize the topic map to the size specified by the slider,
4. visualize the summarized map using a graph-drawing tool.

ScienScan can be deployed on top of any academic search engine with an API. The current version relies on the search API of `Arnetminer`, while the previous version used the API of `Microsoft Academic Search`. The algorithms for building and summarizing the topic map are described in Sections 3.3 and 3.4 respectively. After the summarized topic map is produced, we submit it to the `graphviz` package [32] for visualization.

3.3 Building topic maps with the help of Wikipedia

In Section 2.2.4 we discussed how the network of articles and categories of Wikipedia can be viewed as a large-scale collaboratively edited taxonomy of topics. This section presents our method for building the topic maps for collections of documents, primarily academic search results, using the Wikipedia-based topics. In the next Section 3.4 we describe the method for summarizing topic maps to arbitrary sizes.

At the high level, the procedure for building a topic map consists of the following steps:

1. mapping documents to Wikipedia articles,
2. retrieving the articles' parent categories,
3. establishing the relations between the categories,

4. merging duplicate topics, and
5. breaking the cycles in the topic graph.

In the following sections we will describe these steps in detail.

3.3.1 Notation

Let us introduce some notation for the problem of building topic maps. An input to the procedure is a collection of documents

$$D = \{d_1, d_2, ..., d_N\}.$$

In the academic search scenario, the documents correspond to titles and abstracts of the papers retrieved by the search engine. The result of the procedure is a topic graph

$$G[V, E],$$

in which the links $(v, v') \in E$ represent parent-child relation between topics, and a relation

$$R \subseteq V \times D$$

defining which documents are relevant to which topics. In order to form a valid topic hierarchy, the graph G must be acyclic.

3.3.2 Mapping documents to Wikipedia articles

In this step we identify which Wikipedia articles are relevant to the documents in D, using *wikification* procedure [66]. *Wikification* takes arbitrary text fragment as an input, and annotates the phrases in this fragment with hypertext links to Wikipedia articles, similarly to the way the texts of the Wikipedia articles are linked to each other. For a piece of text like the following:

> . . . a method of summarizing collections of documents with concise topic hierarchies, and show how it can be applied to visualization and browsing of academic search results.

wikification may return something like this:

> . . . a method summarizing collections of documents with concise [[Topic (linguistics) |topic]] [[Hierarchy |hierarchies]], and show how it can be applied to [[Visualization (computer graphics) |visualization]] and [[Web browser |browsing]] of [[List of academic databases and search engines |academic search]] results.

Some of the tools that currently provide wikification services are TAGME [29], Machine Linking [11], DBPediaSpotlight [63], Wikipedia Miner [67], and Dexter [15]. We *wikify* the documents (e.g. the abstracts), and then associate each document with the set of Wikipedia articles to which its text has been linked:

$$\begin{aligned} R &:= \{(v,d) \mid \text{ the wikified text of } d \text{ contains a link to } v\}, \\ V &:= \{v \mid \exists d, (v,d) \in R\}. \end{aligned} \tag{3.1}$$

The collected Wikipedia articles form the initial set of topics in the topic map. Consider a toy example, in which the set of collected articles is shown in Figure 3.3. We will use this topic map as a running example in following steps of the algorithm.

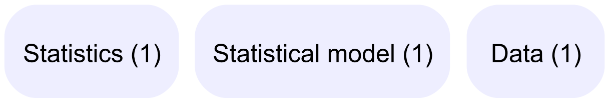


Figure 3.3: A small topic map consisting of isolated topics, corresponding to Wikipedia articles.

3.3.3 Retrieving the parent categories.

For every article v obtained at the previous step we add all its parent categories to the set of topics V:

$$V := V \cup parent\_categories(v),$$

Relations between an article and its categories are added as links to the topic map:

$$E := \{(c,v) \mid c \in categories(v)\}.$$

This step transforms the graph consisting of isolated article-topics (Figure 3.3) into a bipartite graph of articles and categories (Figure 3.4).

3.3.4 Establishing links between categories

At this step we query Wikipedia to discover sub-category relations between the categories introduced at the previous step. The discovered relations are added as links to the map:

$$E := E \cup \{(v_2, v_1) \mid v_1,\, v_2 \in V,\ v_1 \in sub\_categories(v_2)\}.$$

This transforms a bipartite graph (Figure 3.4) into a general directed graph (Figure 3.5).

3.3.5 Merging duplicate topics

Some of the topics have both an associated article and a category in Wikipedia. After the previous step such topics will be repeated twice in the topic map, like topics `Data` and `Statistics` in Figure 3.5. In order to eliminate the redundancy we merge the

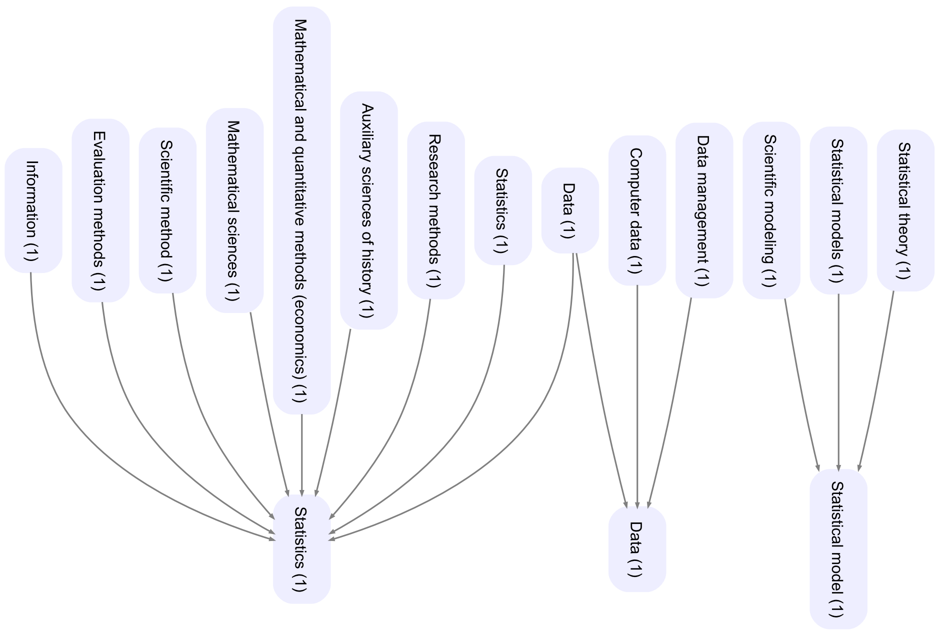


Figure 3.4: A small topic map consisting of Wikipedia articles and their categories.

duplicate topics into one (see Figure 3.6). In addition, we merge near-duplicate topics whose titles coincide up to *lemmatization*, such as singular `Statistical model` and plural `Statistical models`. From this point on we start treating articles and categories homogeneously as topics, without any distinction.

3.3.6 Breaking the cycles

Due to occasional cycles the category graph of Wikipedia does not form a valid hierarchy. The previous steps, in particular merging similar topics, may introduce additional cycles into the topic map. For example, topics `Data management`, `Data` and `Computer data` form a cycle in Figure 3.6. We detect and break the cycles in the topic graph using the depth-first search algorithm. The search is started from the "root" topics—those having no parents except other topics in the cycle. The result of performing this step on our running example is shown in Figure 3.7.

3.3.7 The size of the graph

The described procedure for building the topic maps returns a large directed acyclic graph of topics. For the collection of one hundred abstracts, a typical size of the produced topic map is around three hundred nodes. An example of the topic map build from the search results for the query "quadratic programming" can be found in Appendix B.

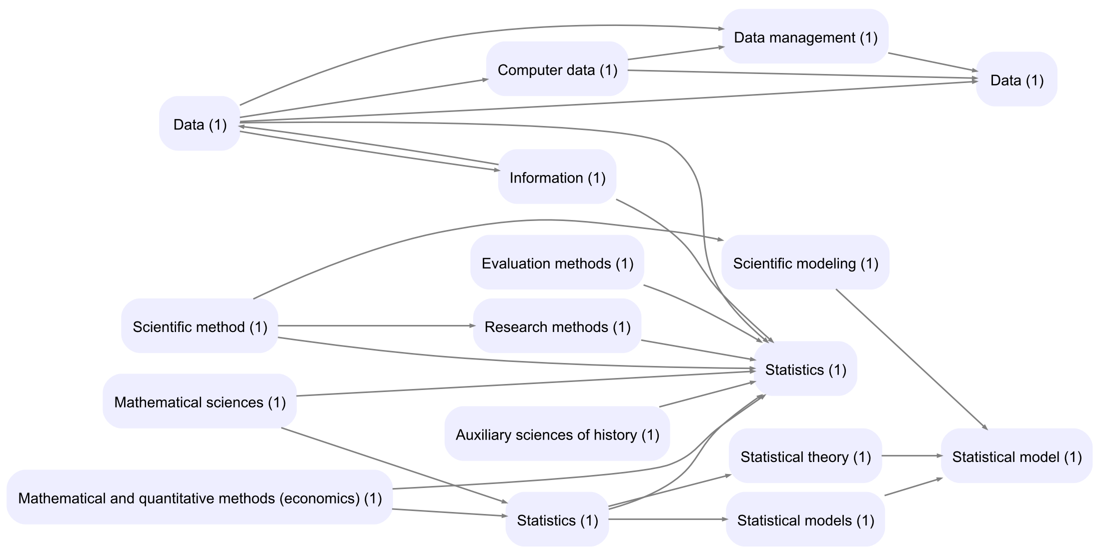


Figure 3.5: A small topic map with established relations between the categories.

3.4 Supervised summarization of the topic maps

In the previous section we described a method for building the topic maps that visually represent a collection of documents (academic search results). However, the topic maps produced at that stage contain hundreds of nodes, being too large to be visually comprehensible. In order for the topic map to be useful it has to be reduced to a reasonable size. In this section we address the problem of summarizing the topic maps. For a given large topic map we aim to produce a smaller map of the given size that would represent the original map—and the underling search results—in the *most informative way*.

The notion of *informativeness* incorporates the quality criteria for the resulting topic maps that we listed in Section 3.1.1. This notion can hardly be defined formally, for one, because it is subjective to a certain extent. It is thus difficult to perceive that one could encode this notion into a deterministic summarization algorithm. On the other hand, we argue that a human expert could provide examples of what is an informative summary for a given topic map. At the very least, the expert could judge whether a certain summary is informative, and make qualitative comparisons between alternative summaries. This observation lead us to taking a *machine learning approach* to this summarization problem. We developed a supervised learning algorithm for summarizing the topic maps based on DAgger—a state-of-the-art method for imitation learning and structured prediction. Provided with a number of examples, the algorithm is able to learn the required combination of features that make for high-quality topic summaries. We describe the summarization algorithm in detail in the next sections.

Here we would like to present some intuitions as to why such summarization is possible.

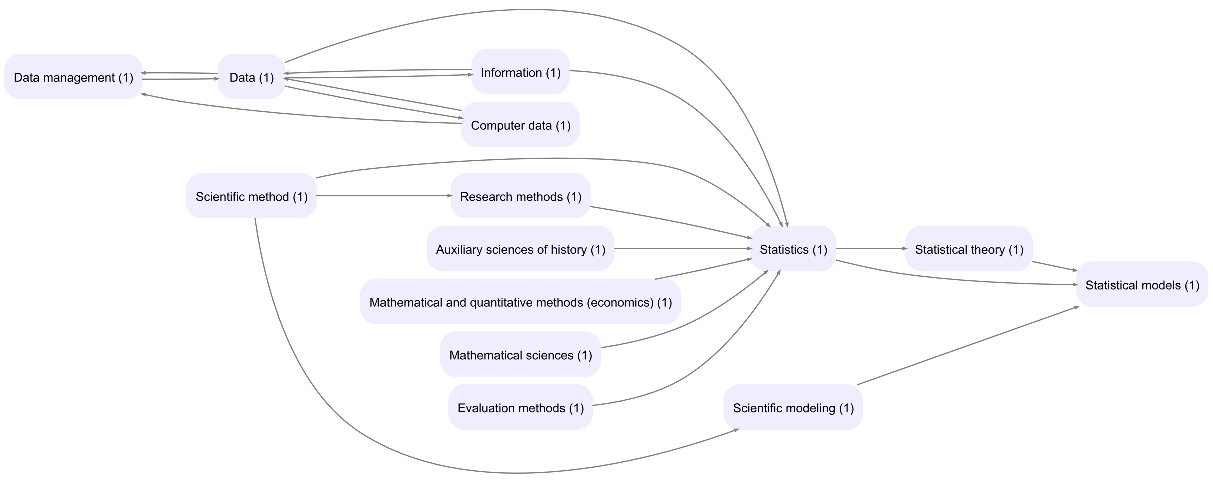


Figure 3.6: A small topic map with similar topics merged into one.

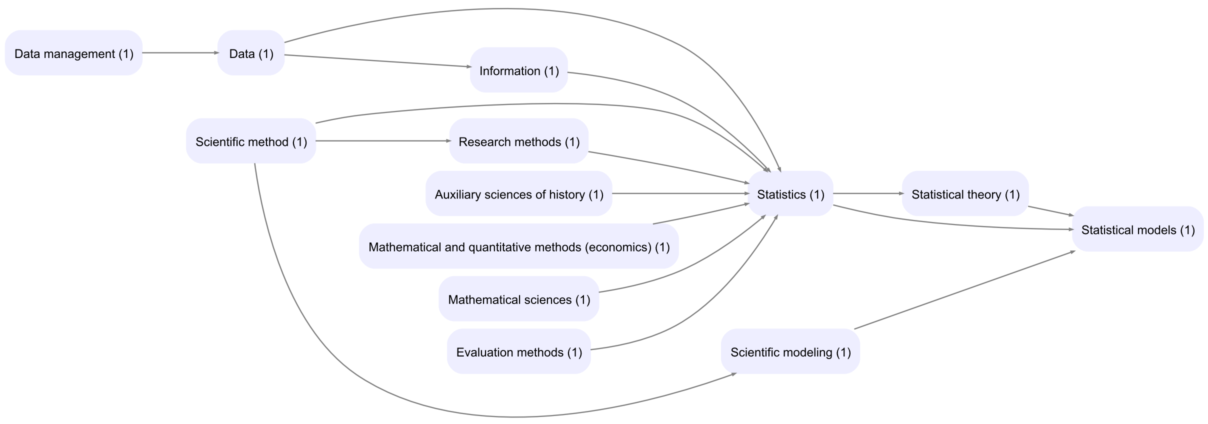


Figure 3.7: A small topic map with removed cycles.

The first intuition is that the topics that *cover* only few documents in the collection can be omitted without loosing much information. Alternatively, such “small” specific topics can be merged into their higher-level parent topics (which, by definition cover all their children’s documents anyway). On the other hand, arguably, the topics that are too general (or cover most of the documents) can also be removed from the map without much loss in information. In general, the map could be summarized by removing topics that are for any reason and to any extent redundant. For instance, of two topics that cover (almost) the same documents, one could be omitted from the summary.

3.4.1 Problem formulation

Inference. Consider a topic map $G[V,E]$ along with the topic-document relation R, as defined in Section 3.3.1. Let $\mathcal{S}_T(G[V,E])$ be the set of possible summaries of G of size T:

$$\mathcal{S}_T(G[V,E]) := \{G_T[V_T, E_T] \mid V_T \subseteq V,\ |V_T| = T\}.$$

For a given size $T \in 1 : (|V|-1)$ our goal is to produce the topic map $G_T \in \mathcal{S}_T$ that summarizes the original topic map G in the most informative way.

We view our summarization problem as a structured prediction problem by defining a linear *scoring function* over the possible summaries:

$$F[\mathbf{w}]((G,R),G_T) := \langle \mathbf{w}, \Psi((G,R),G_T)\rangle. \tag{3.2}$$

The vector-function Ψ is joint feature map that describes the compatibility between the inputs (topic maps) and outputs (summaries). The vector $\mathbf{w}$ defines the relative importance of the features in the resulting score. In the following we will always assume the dependency of $F[\mathbf{w}]$ and Ψ on the topic-document relation R, and omit this relation from notation for simplicity. The solution to the summarization problem is a summary G_T that delivers maximum to the scoring function $F[\mathbf{w}]$:

$$G_T := \operatorname*{argmax}_{G'_T \in \mathcal{S}_T(G)} F[\mathbf{w}](G, G'_T). \tag{3.3}$$

Learning. In order to learn the weights $\mathbf{w}$ we need a "ground truth" training set D of examples of the form (G, G_T), where G is a topic map, and G_T—its most informative summary. The goal of *learning* is to find $\mathbf{w}$ that make the ground truth summaries in the training set score higher than all other possible summaries with respect to $F[\mathbf{w}]$.

Complexity considerations As we discussed in Section 2.3.1, the exact inference, such as predicting G_T according to 3.3, is infeasible in general, due to the large number of candidate outputs to be evaluated. As an illustration, consider the number of possible summaries of size 10 of a topic map of size 300. If we ignore the edges for simplicity, and count only the sets of topics, we will arrive at $\binom{300}{10} = 1\,398\,320\,233\,241\,701\,770$ summaries—clearly to many to evaluate. In the following Sections 3.4.2–3.4.3 we introduce two assumptions that 1) make the evaluation of argmax feasible and 2) make the learning problem amenable to sequential prediction as described in Section (2.6.2).

3.4.2 Simplification: Predicting only the nodes (topics)

A topic map summary consists of the nodes (topics) and the links (relations). Thus, in principle, predicting a summary requires predicting both the nodes and the links. However, we note that, once the nodes V_T are predicted, the links E_T can be automatically derived based on the links E of the original map.

Assumption 1. *Predicting a summary $G_T[V_T, E_T]$ reduces to predicting the topics V_T.*

In Section 3.4.6 we describe an procedure for deriving the links E_T based on the nodes V_T and the original map G. For simplicity, in the following we will use the notation for graphs G, G_T interchangeably with the notation for nodes V, V_T.

3.4.3 Simplification: Nested summaries

As we mentioned in Section 3.4.1, enumerating all possible topic map summaries of size T in the computation of argmax (3.3) is prohibitively expensive. We alleviate this problem by imposing an additional constraint on the summaries that is natural in our settings. Specifically, we require that for a given input graph G the optimal topic summaries of different sizes should be nested:

$$G_1 \subset G_2 \subset ... \subset G_T.$$

In other words:

Assumption 2. *Bigger summaries are made from smaller ones only by adding new topics:*

$$G_{t+1} = G_t \cup \{v_{t+1}\}.$$

Considering the browsing interface of ScienScan presented in Section 3.2, Assumption 2 can be justified by the principle of least surprise: when increasing the granularity of the topic map summary, it is natural that additional topics are included, while none of the previously displayed topics disappear. With Assumption 2, predicting a summary G_T reduces to predicting a sequence of topics $v_{1:T}$. The prefixes of the topic sequence constitute the nodes of the intermediate summary graphs of sizes from 1 to T:

$$G_t = v_{1:t}, \quad t \in 1 : T \tag{3.4}$$

The assumption of nested summaries imposes an additional requirement on the training dataset: for a given input graph G in the training set we require that we know the "ground truth" summaries of all sizes from 1 to T, rather than only the summary G_T.

The main advantage of assumption 2 is that the summaries can be built sequentially, by predicting the topics $v_1, v_2, \dots, v_T$ one by one. The decomposition of a summary into a sequence of topics lends itself naturally to the framework of sequential prediction of structured outputs, in particular the DAgger algorithm we described in Section 2.6.2. In the following sections we will detail the application of DAgger to the specific problem of predicting the summaries of the topic maps.

3.4.4 Applying DAgger to summarizing the topic maps

In the previous sections we discussed how predicting the summaries $G_{1:T}$ of a topic map G can be reduced to predicting the sequence of topics $v_{1:T}$. Learning to build the sequences of topics from examples can be viewed as an imitation learning problem, in which we aim to mimic the expert's actions. The *states* in this case are intermediate summaries $G_t = v_{1:t}$, while *actions* are the topics v_{t+1} that are being added to the summaries. In Section 2.6.1 we described the DAGGER algorithm for imitation learning, and detailed its application to sequential structured output prediction in Section 2.6.2. For the sake of clarity we repeat the algorithm of DAGGER applied specifically to problem of learning to predict summaries of the topic maps.

Premises.
As an input we have a **training set** of examples of the form $(G, v_{1:T})$, where G is a topic map, and $v_{1:T} = (v_1, v_2, \dots, v_T)$ is the sequence of topics, such that its first t elements $v_{1:t}$ constitute the intermediate summary of size t. We also have the **expert policy** π^*. For a given training example $(G, v_{1:T})$, and a non-optimal partial summary ("state") $\hat{G}_t = \hat{v}_{1:t}$, the expert policy should return the best next topic ("action") $\hat{v}^*_{t+1}$ that can be added to this summary. Lastly, we have a way to **train a classifier** $\pi[\mathbf{w}]$ on the dataset D of the state-action pairs. With these premises, the algorithm of DAGGER works as follows.

The algorithm.

```
• Initialize the current policy $\pi^0 := \pi^*$ with the expert policy.
• Initialize the dataset $D \leftarrow \varnothing$ of state-action pairs.
• Repeat for $N$ iterations $i \in 1 : N$:
    – For each training example $(G, v^*_{1:T})$
        ∗ Predict the summary $G^i_T = v^i_{1:T}$ using the current policy $\pi^i$:
              $v^i_1 := \pi^i(G),$
              $v^i_2 := \pi^i(G, v^i_1),$
              $\dots$
              $v^i_T := \pi^i(G, v^i_1, v^i_2, \dots, v^i_{T-1}).$
        ∗ For each visited "state" (partial summary) $G^i_t = v^i_{1:t},\ t \in 0 : (T-1)$:
            · Generate the expert "action" (best next topic to be added):
                  $\hat{v}^{*i}_{t+1} := \pi^*(G, v^i_{1:t})$
            · Add the state-action pair $(G^i_t, \hat{v}^{*i}_{t+1})$ to the dataset $D$:
                  $D \leftarrow D \cup \{(G^i_t, \hat{v}^{*i}_{t+1})\}$
    – Train the next policy $\pi^{i+1}$ on the dataset $D$
• Return $\pi^N$, or best $\pi^j$, $j \in 1 : N$ across iterations.
```

Training the policy π. In order to train the policy $\pi[\mathbf{w}]$ on the state-action pairs, we use the L_2-regularized ranking SVM, as implemented in SVM^{rank} [45]. When predicting the next topic v_{t+1} for a sequence $v_{1:t}$, we rank all possible candidates v' according to the score $\langle \mathbf{w}, \Psi(G, v_1, v_2, \ldots, v_t, v') \rangle$, and select the highest-ranked topic. As we are always interested in a single highest-scoring topic, we train the ranking SVM to distinguish only between the "ground truth" action and the rest, as described in "Predicting the top-ranked item", Section 2.3.3. This results in a number of constraints in SVM^{rank} that is *linear* in the number of possible outputs v_{t+1}, which in our case equals $|G| - t$.

Specifying the expert policy π^*. As described in Section 2.6.2, the expert policy π^* returns the "action" that moves the current trajectory closest to the ground truth trajectory. In terms of our problem, given a topic map G, and the ground truth sequence $v^*_{1:T}$, for a given non-optimal partial sequence $v_{1:t}$ the expert policy returns the topic $\hat{v}^*_{t+1}$ that makes the sequence $v_1, v_2, \ldots, v_t, \hat{v}^*_{t+1}$ most similar to $v^*_{1:T}$:

$$\pi^*(v_{1:t}) := \operatorname*{argmin}_{\hat{v}_{t+1}} \Delta((v_1, v_2, \ldots, v_t, \hat{v}_{t+1}), v^*_{1:t+1}). \tag{3.5}$$

To completely define the expert policy, we need to specify the distance function Δ on the partial topic sequences. For this purpose, using the zero-one distance $\Delta_{0/1}$ would be inappropriate, as it would score all non-optimal sequences equally, rendering the minimization problem (3.5) meaningless in most cases. An obvious candidate for Δ is the Jaccard distance [42] between the sequences of topics viewed as sets:

$$\Delta_{Jaccard}(v_{1:t}, v'_{1:t}) := \frac{v_{1:t} \bigtriangleup v'_{1:t}}{v_{1:t} \cup v'_{1:t}}.$$

However, the Jaccard distance does not take into account the similarities between the topics, which may result in redundant topic maps. As an illustration, suppose that the ground-truth sequence $v^*_{1:t}$ contains just two topics (A, B), while the partial non-optimal sequence is (A'), where A' is similar to A (in terms of title and the covered documents). Both sequences (A', B) and (A', A) look equally good with respect to Jaccard distance to (A, B), while (A', B) is clearly a better match.

We designed a matching-based distance function $\Delta = \Delta_{match}$ that takes into account the similarities between the topics. Let $d(v, v')$ be some measure of dissimilarity between topics. Let $\Delta_{pairwise}(v_{1:t}, v'_{1:t})$ be the dissimilarity between v_k and v'_k, averaged across k:

$$\Delta_{pairwise}(v_{1:t}, v'_{1:t}) = \frac{1}{\sup_{v,v'}(d(v, v'))} \frac{1}{t} \sum_{k=1}^{t} d(v_k, v'_k). \tag{3.6}$$

In our work we define $d(v, v')$ as the Jaccard distance between the sets of documents covered by v and v' plus the constant α if $v \equiv v'$ (in this case $\sup\limits_{v,v'}(d(v, v')) = 1 + \alpha$). We define $\Delta_{match}(v_{1:t}, v'_{1:t})$ to be the minimum $\Delta_{pairwise}$ across all permutations of $v'_{1:t}$:

$$\Delta_{match}(v_{1:t}, v'_{1:t}) := \min_{\sigma \in \mathfrak{S}_t} \{\Delta_{pairwise}(v_{1:t}, \sigma(v'_{1:t})). \tag{3.7}$$

As the exact computation of $\Delta_{match}(v_{1:t}, v'_{1:t})$ is overly expensive, we use a greedy approximation to Δ_{match}. The approximate algorithm finds the permutation $\sigma(v'_{1:t})$ by greedily minimizing $\Delta_{pairwise}(v_{1:t}, \sigma(v'_{1:t}))$ in equation (3.7). First, we find the best-matching topic for v_1 among $v'_{1:t}$, then—for v_2 among the remaining $t-1$ topics, and so on. The greedy algorithm requires $O(t^2)$ operations for the approximate computation of Δ_{match}.

3.4.5 Features

A crucial part of the described learning algorithm is the joint feature representation $\Psi(G, v_{1:t+1})$ that is involved in training the policies of DAgger. Based on the features, the policy π should be able to learn how to add topics v_{t+1} to the intermediate summary graphs $G_t = v_{1:t}$. The features we use measure various properties of the topic map $G_{t+1} = v_{1:t+1}$ that results from adding the topic v_{t+1} to the summary G_t. The computed properties describe the structure and the look of the resulting summary, as well as the topic-document relations it induces.

The first set of features is related to the coverage and diversity of the topics $v_{1:t+1}$ in the summary. Recall the definition of the topic-document relation R in (3.3.1), and the parent-child topic relation E. R describes which documents are *directly* covered by which topics. By R^+ we denote the relation that results from propagating the topic-document relation to the parent topics. R^+ describes which documents are *transitively* covered by which topics, and is defined technically as the composition of E^+ and R:

$$R^+ := E^+ \circ R,$$

where E^+ is the *transitive closure* of E. Let $docs(v)$ and $docs^+(v)$ denote the documents covered by the topic v directly and transitively respectively:

$$docs(v) := \{d \in D \mid (v, d) \in R\},$$
$$docs^+(v) := \{d \in D \mid (v, d) \in R^+\}.$$

Let *topics* and *topics*$^+$ denote the respective inverse relations of *docs* and *docs*$^+$. With these definitions, the features related to coverage and diversity include:

1. *direct document coverage:* $\left| \bigcup_{v \in v_{1:t+1}} docs(v) \right|$;
2. *transitive document coverage:* $\left| \bigcup_{v \in v_{1:t+1}} docs^+(v) \right|$;
3. average and minimum *topic frequency*, where

$$topic\_freq(v) := |docs(v)|;$$

4. average and minimum *transitive frequency*, where

$$trans\_topic\_freq(v) := \big|docs^+(v)\big|;$$

5. average and maximum topic *overlap*, where

$$overlap(v_i, v_j) := 1 - \Delta_{Jaccard}(docs^+(v_i), docs^+(v_j));$$

6. average and maximum parent-child *overlap*, with *overlap* defined above;
7. average pairwise *distance* between the topics, where *distance* is the length of the shortest path through a common ancestor in the original graph;
8. *partition coefficient* (measures the crispness of topics as fuzzy document clusters):

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{|topics^+(d_i)|}.$$

Another set of features describes the properties of the topic summary G_{t+1} as a graph. These features include *a*) the number of connected components, *b*) the number of links, *c*) the height of the graph, *d*) the average number of children for topics having children, and *e*) the average and the maximum number of parents for topics having parents.

Some features communicate certain properties that do not fall into the described categories. Thus, we have a feature that measures the *unevenness* of the sizes (transitive frequencies) of pairs of sibling topics. Another feature is used to measure the "subtopic coverage", that is the average ratio between the number of subtopics in the original graph and the number of subtopics in the summary (for topics that have subtopics in the summary graph).

3.4.6 Connecting topics and documents.

In the previous sections we described a procedure that allows sequential selection of topics $V_T = v_{1:T}$ of the topic map summary G_T. In order to completely define the summary, we need to decide on the links between the topics, and the topic-document relations. On one hand, the links in the summary should reflect the hierarchical relations between the topics in the original topic map. This property can be defined as follows:

Definition 9. *$G_T[V_T, E_T]$ maintains the hierarchical structure of $G[V, E]$ if and only if*

$$\forall (v, v') \in G_T \; ((v, v') \in E_T^+ \Leftrightarrow (v, v') \in E^+).$$

In other words, for any pair of topics in the summary, one is an ancestor of the other either in both the summary and the original graph or in none of them.

On the other hand, the summary should not be cluttered with unnecessary and redundant links. A natural balance between these two objectives is finding the minimum number of links that still maintain the hierarchical structure of the original graph. Let r^- denote the *transitive reduction* [1] of the relation r (transitive reduction is the relation inverse to the transitive closure).

Proposition 2. *Among all the graphs on the nodes V_T satisfying the definition (9), the graph that has the links*

$$E_T := (E^+ \cap (V_T)^2)^-$$

has the minimum number of links.

Completing the graph G_T with edges E_T given the topics V_T amounts to:

- computing the transitive closure G^+ of the original graph $G[V, E]$,
- selecting the subgraph G_T^+ of G^+ containing only the nodes V_T,
- computing the transitive reduction of the selected subgraph G_T^+.

As for the topic-document relation, to any topic v in the topic summary we assign all the documents that were *transitively covered* by v in the original graph.

3.5 Evaluation of the topic map summarization

We carried out the evaluation of the proposed method for summarizing the topic maps on the search results obtained from Microsoft Academic Search[1] for 10 distinct queries. For each query we collected one hundred top results, discovered the topics in their titles and abstracts, and built the topic maps as described in Section 3.3. The topic maps were then annotated with "ground truth" topic sequences of length 8, corresponding to topics in the summaries of sizes 1 to 8. The summaries were selected so as to represent the search results and the discovered topics in the most informative way according to our judgment.

The summarization method (presented in Section (3.4)) was evaluated on the task of predicting the topic sequences using leave-one-out cross-validation on the described dataset. Two different performance metrics were used: `precision@n` and `match@n`. `precision@n` measures the percentage of the correctly predicted topics in the subsequence of length n, taking into account only exact matches. Similarly, `match@n` measures the similarity based on Δ_{match} (see definition (3.7)) between the subsequences of length n, thus allowing for partial matches between similar topics. For the sake of comparison we implemented one baseline algorithm that greedily maximizes the coverage of the topics, and adapted the spectral clustering–based method of Scaiella et al. [89].

[1]http://academic.research.microsoft.com/

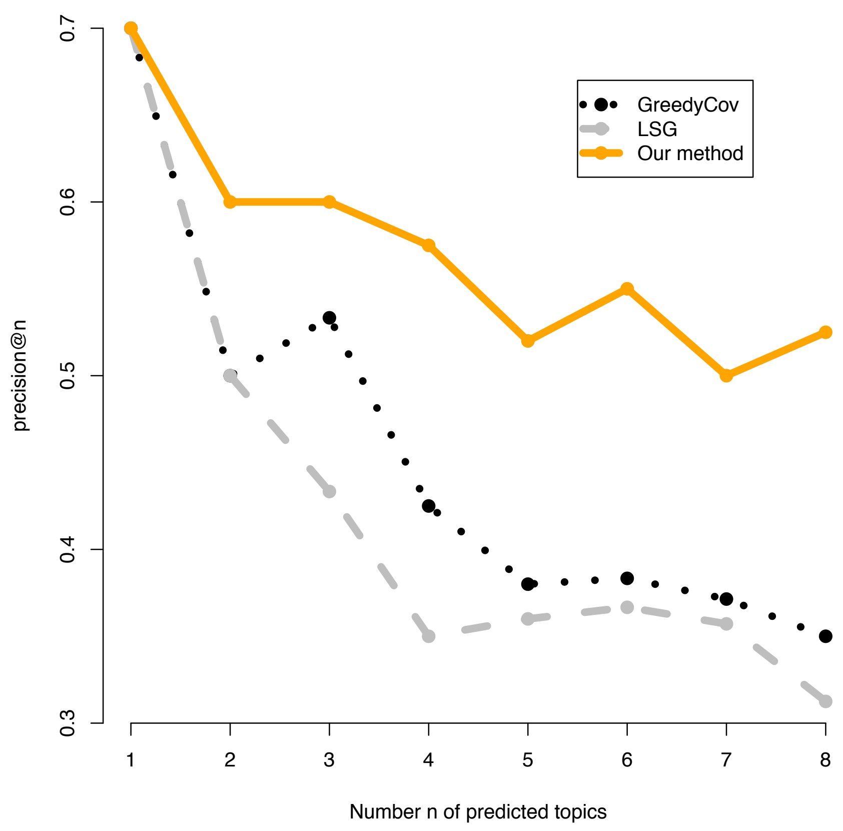


Figure 3.8: Precision@n of predicted topics. Dotted black line corresponds to the baseline `GreedyCov` method, dashed grey line—to `LSC`, and solid orange line—to our method at the 10th iteration of DAgger.

3.5.1 Baseline GreedyCov algorithm.

The implemented baseline algorithm `GreedyCov` selects topics by greedily maximizing the document coverage. At step $t+1$ `GreedyCov` chooses the topic v_{t+1} that covers the most of the documents that have not been covered by the previous topics $v_{1:t}$:

$$v_{t+1} := \underset{v}{\operatorname{argmax}} \left| \bigcup_{v' \in v_{1:t} \cup \{v\}} docs(v') \right|$$

This is a reasonable baseline, as it optimizes both the document coverage and the diversity of the selected topics. As we pointed out in Section (3.1.1), both of these properties are important for a good topic map. The supervised method we proposed in Section (3.4) can also be seen as greedily optimizing a linear combination of features, with the difference being that the feature weights are learned from the training data.

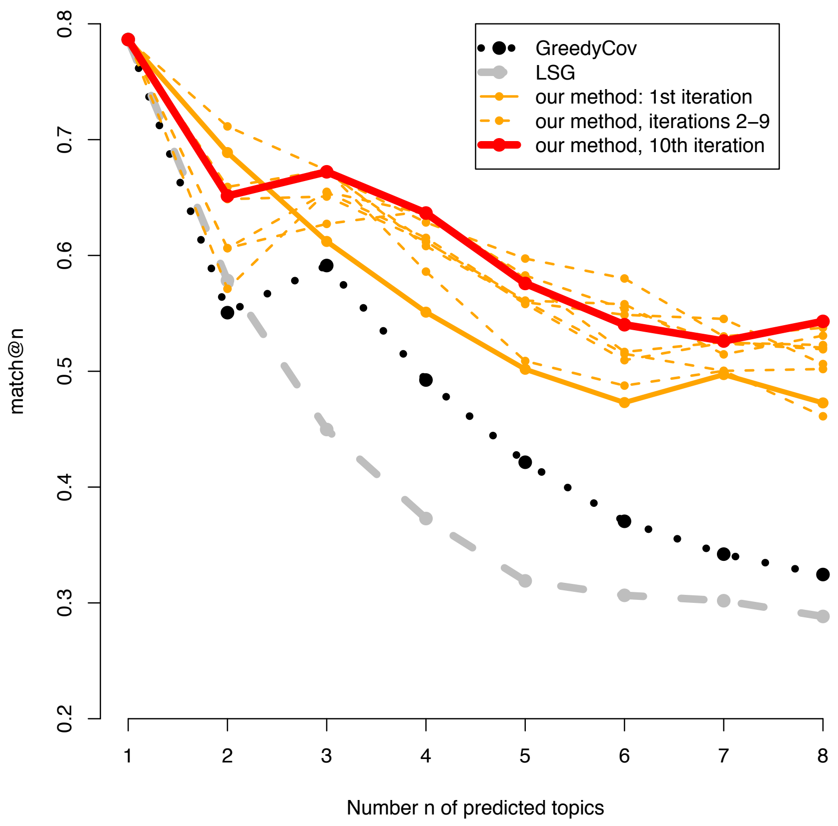


Figure 3.9: match@n of predicted topics. Dotted black line: baseline `GreedyCov` method; dashed grey line: `LSC`; solid orange and red lines: our method at the 1st and the 10th iteration respectively; thin dashed orange lines: intermediate iterations.

3.5.2 Labeled spectral clustering

Scaiella et al. [89] recently proposed a novel method for clustering of the Web search results based on Wikipedia. The method performs a particular form of spectral clustering on the graph of documents and topics, with subsequent selection of cluster labels. For convenience we will refer to this method as `LSC` (labeled spectral clustering).

In the first step of `LSC` the documents (search result snippets) are annotated with links to Wikipedia articles using the TAGME topic annotator [29]. For each topic-document link (v, d) TAGME provides an importance score $\rho(v, d)$, which is used as link weight in the resulting weighted bipartite document-topic graph. The graph is then augmented with *non-hierarchical* (undirected) between-topic links (v, v') that are weighted according to topic relatedness. The relatedness between topics $rel(v, v')$ is also computed by TAGME and is based on the incoming citations of the corresponding Wikipedia articles. The subsequent graph pre-processing step selects the most significant topics by greedily solving a variation of a set cover problem.

In the original method of Scaiella et al. topics that cover more then fifty percent of the documents are removed from the graph prior to selecting significant topics. This step is justified in plain clustering considered in [89], as overly frequent topics do not help discriminating between documents. In our approach the most frequent topic often corresponds to the main topic of the search query, and is arguably useful for our hierarchical topic representation. Therefore we omit this step in our implementation of `LSC`.

The topics are then iteratively clustered based on the spectral properties of the graph of topics and their relations. At each iteration the algorithm selects one of the big clusters—those covering more than δ_{max} documents—and splits it in two. Informally, the algorithm ensures that the sparsest of the big clusters is selected, and that the split goes through the sparse region of the corresponding topic subgraph. As recommended in [89], in order to obtain T final clusters we build $T + m - 1$ clusters with the described algorithm and then merge m smallest clusters into one.

Finally, each cluster is labeled with the topic most strongly associated with the documents in the cluster, as measured by document-topic weights $\rho(d, v)$. We treat the produced cluster labels as the representation of the search results according to `LSC`.

Overall there are the following main differences between `LSC` and our algorithm: *a*) `LSC` uses only Wikipedia articles to represent topics, while we use both articles and categories; *b*) `LSC` relies on similarity-based relatedness between topics, while we rely on hierarchical relations in the article-category network; *c*) in `LSC` topic aggregation is performed on the basis of clustering, while in our method on the basis of topic *generalization* (based on the hierarchy); *d*) `LSC` selects topics through unsupervised procedure of labeled clustering, while we rely on supervised structured output prediction.

Details of the evaluation setup. For the ease of comparison, we used TAGME as a topic annotator for all the three algorithms[2]. When evaluating the results produced by `LSC` we first embedded them into the topic graph, built as described in Section 3.3, in order to correctly match the results to the "ground truth" topics, in particular to capture *similar topic* matches.

The metrics `precision@n` and `match@n` are designed to evaluate the sequences of summary graphs of increasing sizes $t \in 1 : T$, as they are produced by our method. In order to evaluate the method of Scaiella et al. on the same basis, we ran `LSC` with different values of $t \in 1 : T$. For each t we executed the method with different parameter values, and selected the best cross-validated result according to the metric in question. For simplicity, δ_{max} was fixed at the value of 3 which is arguably the smallest number of documents we would like to see in a cluster. We should note that changing the value of δ_{max} did not notably affect the results, which confirms the robustness of the method reported in [89]. The value of m—the number of the smallest clusters to be merged—was ranged from 1 to 5, which corresponded to producing from 8 to 12 clusters prior to merging.

The performance of our method was measured at the first ten iterations of DAGGER.

[2]Due to this setting, `LSC` was executed with the topic annotator it was originally used with

3.5.3 Evaluation results

Figures 3.8 and 3.9 show the performance of the three evaluated methods. The first iteration of our method is equivalent to not using DAgger, and just training the policy on the states encountered in the ground truth labeling. As we can see from the figures, at $T = 1$ the curves coincide, as for each of the 10 queries the methods happen to agree on the first predicted topic. As the number of topics increases, the curves begin to diverge, with the growing advantage of our method over the other two. The advantage over `LSC`, measured as the difference between the scores, becomes statistically significant with p-value of 0.05 starting from $n = 3$, and over `GreedyCov`—starting from $n = 6$. The difference between the performance scores of `GreedyCov` and `LSC` is not significant for any n. We should note that the performance increase of our method after the first iteration justifies the procedure of dataset aggregation.

We can see that `GreedyCov` performs reasonably well for small n, which indicates that document coverage is an important characteristic for summary topic graphs of small sizes. As more topics are added to the graph, the performance of `GreedyCov` notably deteriorates. At the point when most of the documents are covered by previously selected topics, the greedy coverage strategy becomes suboptimal, as it starts to prefer "outlying" topic nodes.

The spectral clustering–based `LSC` method encourages regular topic sizes both in terms of contained Wikipedia articles and documents. The drawback of `LSC` in the context of our task is that it is designed for plain rather than hierarchical clustering. In general we can conclude that, being unsupervised methods, `LSC` and `GreedyCov` encode some of the important properties of good topic summaries. However, as they are not specifically tailored for producing hierarchical summaries of various sizes, the captured properties are not sufficient for building the sequences of informative summary graphs. In these settings our supervised method has an advantage, as it is able to learn how to combine multiple properties in order to build high-quality summary sequences.

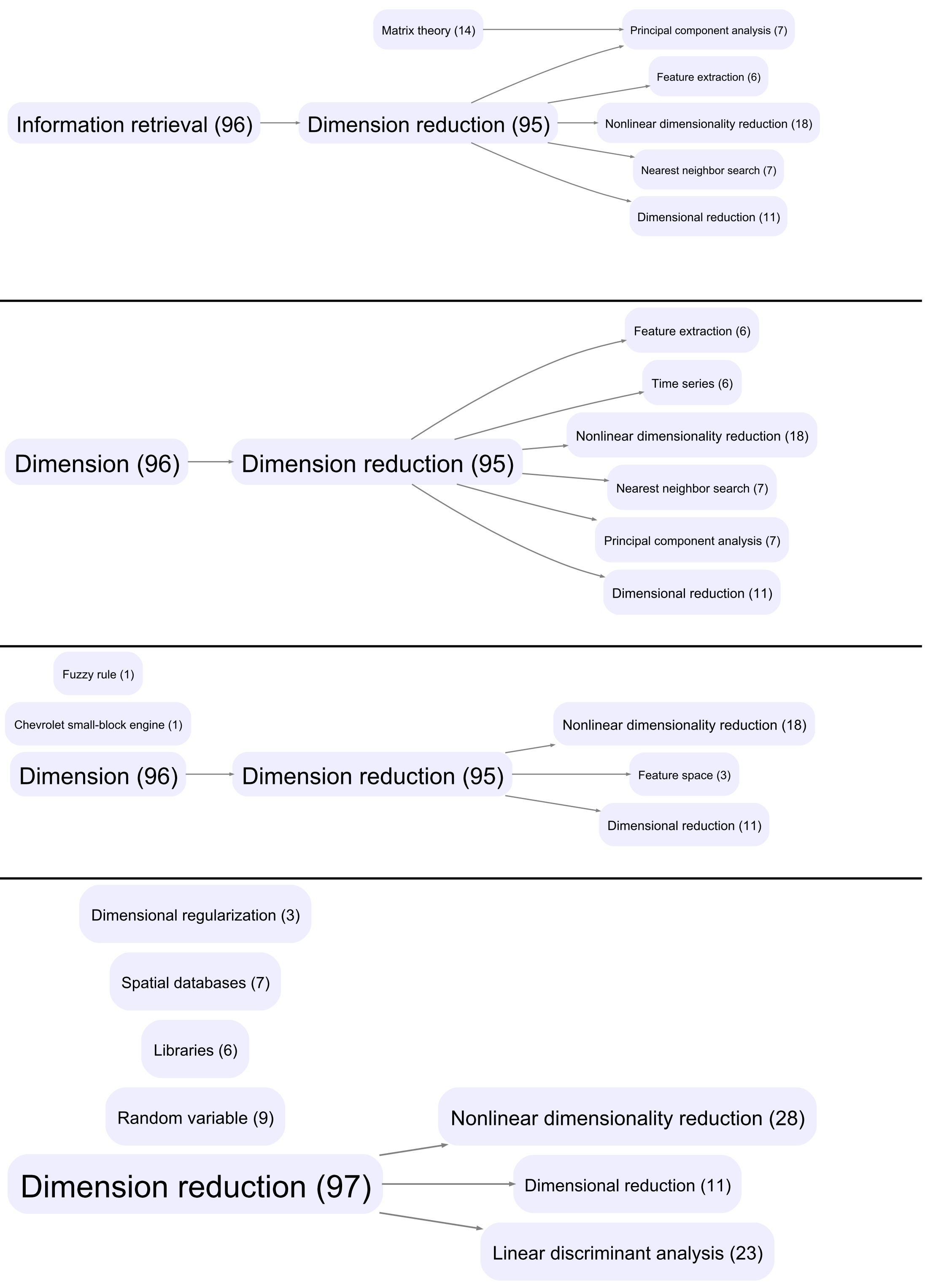

Matrix theory (14)
Principal component analysis (7)
Feature extraction (6)
Information retrieval (96)
Dimension reduction (95)
Nonlinear dimensionality reduction (18)
Nearest neighbor search (7)
Dimensional reduction (11)
Feature extraction (6)
Time series (6)
Dimension (96)
Dimension reduction (95)
Nonlinear dimensionality reduction (18)
Nearest neighbor search (7)
Principal component analysis (7)
Dimensional reduction (11)
Fuzzy rule (1)
Chevrolet small-block engine (1)
Nonlinear dimensionality reduction (18)
Dimension (96)
Dimension reduction (95)
Feature space (3)
Dimensional reduction (11)
Dimensional regularization (3)
Spatial databases (7)
Libraries (6)
Random variable (9)
Nonlinear dimensionality reduction (28)
Dimension reduction (97)
Dimensional reduction (11)
Linear discriminant analysis (23)

Figure 3.10: Summarized topic maps of the search results for the query "dimensionality reduction" produced by (from the top): manual labeling, our method, `GreedyCov` and `LSC`.

Chapter 4

Deriving ontologies from Wikipedia categories

In the previous chapter (3) we described the idea, the methods and an implementation of topics maps—a possible step towards representing the academic search results in a structured and informative way. The useful information conveyed by a topic map included the names of the topics (concepts) that are relevant to the search results, the grouping of the search results according to these topics, and the subsumption relations between the topics. The work presented in this chapter is motivated by the idea of enhancing the structure of the topic maps with additional knowledge. From the point of view of knowledge representation, the topic maps described in Chapter 3 are *taxonomies*—classification systems with a subsumption relation of generic (unspecified) nature. A richer kind of knowledge structure, *ontology*, permits various specific relation and node types as well as non-hierarchical relations.

Ontologies have been useful for organizing knowledge in a variety of domain applications [94]. The topic maps we presented in the previous chapter are powered by the category structure of Wikipedia. With the motivation of building ontologically rich topic maps, we have targeted a broader task of extracting domain-specific ontologies from Wikipedia. In the following sections we describe our approach to this problem, with early implementation and experimental results. The method first selects a subset of the Wikipedia category network relevant to the domain, then determines the types of nodes and relations in the extracted subset, relying on supervised learning in all the subtasks.

Facilitating the creation of large-scale domain ontologies is an important problem in its own right, with any advances in this direction having an impact in the general field of knowledge engineering and in various domain applications. We would like to highlight a specific contribution that this problem makes to the main theme of this thesis. A way to extract domain ontologies from the categories of Wikipedia enables applications like ScienScan (Section 3.2) to run on top of the ontologically rich network of domain-specific topics. We envision further improvement of the knowledge structures supporting ScienScan through integration of the extracted Wikipedia-based ontologies with existing domain vocabularies and classification systems like `ACM CCS` and MESH [56].

4.1 Introduction

Building ontologies is a difficult task that requires expertise in the domain that is being modeled, as well as in logic and ontological analysis. Domain knowledge is necessary to decide the scope and the boundaries of the ontology, that is, to separate the entities relevant to the modeled domain from the accessory elements that should not be included into the ontology. This activity is identified by most of the ontology engineering methodologies as "definition of scope and boundaries" (see [41] for a comparative analysis). In this phase a typically large set of terms is collected, from which it is necessary to identify the relevant subset. In the subsequent phase, domain expertise is needed to express the relations between the selected entities. The most important kind of relations are hierarchical relations, such as meronomy and partonomy. Another essential relation is the one between an entity and its type. A clear distinction between specific relation types requires additional competences in logics and ontological analysis. Domain experts tend to merge these relations into a generic `broader/narrower` relation, which results in the partially formalized knowledge resources, such as classification schemes, lexicons, and thesauri.

These types of partially structured descriptions of the domain are widely used in the Semantic Web. They span from global categorizations, such as that of Wikipedia, to domain-specific schemes, such as the ACM Computing Classification System, and provide great support for structural access to web resources that go beyond the keyword search method. In Section 3.2 we presented ScienScan, which provides structured access to academic literature by navigating the Wikipedia category network. A fully developed formal representation of this structure would enhance these types of applications with the possibility of a more flexible search based on semantic query answering [77]. Therefore, we argue, it is worth investigating into the effective methods that could support transforming the informally structured knowledge representations systems to fully formalized ontologies. The formalization of these, typically large, structures cannot be accomplished manually by the knowledge engineers, and should be performed in an automatic manner.

In the following sections, we propose an automatic method based on the supervised machine learning techniques for transforming the Wikipedia category hierarchy, into *ontology skeletons*. We use the term *ontology skeleton* to indicate a basic version of an ontology that contains all the primitive concepts, the essential hierarchical relations between them, and some track of the remaining generic relations of unspecified type. An ontology skeleton is meant to be further refined in two ways: first, by providing feedback and corrections to the solutions proposed by the automatic algorithm, and second, by refining the generic relations into more specific ones. The method first selects a subset of the categories that are relevant for the given domain. The relevant categories are then split into classes and individuals, and, finally, the relations between the categories are classified as either `subclass_of`, `instance_of`, `part_of`, or generic `related_to`.

We evaluate our method by generating ontology skeletons for the domains of Computing and Music from the categories of Wikipedia. The quality of the generated ontologies has been measured against manually built ground truth datasets. The evaluation results suggest high quality in the part of selecting the relevant nodes, discriminating classes from individuals, and identifying the `subclass_of` relation. The accuracy of identifying the `instance_of` relation is on par with the state of the art. The more difficult `part_of` relation between Wikipedia categories was never addressed in the literature, and our initial results need further improvement.

4.2 Background and problem statement

The described problem fits into the scenario of reusing non-ontological resources for building ontologies, according to [96]. We start from a non-ontological resource that is the Wikipedia category network [1].

The Wikipedia category system provides navigational links to all Wikipedia pages in a mostly hierarchical organization of categories. Each Wikipedia page can be placed under one or more categories. With some exceptions, the categories follow the established naming conventions[2]. Examples of conventions relevant for our task include the following:

- the names of the *topic categories* should be singular, normally corresponding to the name of a Wikipedia article;
- the names of the *set categories* should be plural;
- category names should not contain structure;
- the meaning of a name should be independent of the way the category is connected to other categories;
- the words and phrases in the category names should exist in reliable sources.

As suggested above, there are two main kinds of categories: *topic categories*, named after a topic (and usually sharing a name with the Wikipedia article on that topic), and *set categories*, which are named after a class (usually in the plural). An example of a topic category is France, which contains articles speaking about the whole France; an example of a set category is Cities_in_France, which contains articles about cities in France.

Wikipedia categories are organized hierarchically into a lattice[3]. There is a top-level

[1]`http://en.wikipedia.org/wiki/Wikipedia:Topic_category`

[2]`http://en.wikipedia.org/wiki/Wikipedia:Category_names`

[3] It is not a pure lattice, as it contains occasional cycles (for instance, Data $\leftarrow$ Computer_data $\leftarrow$ Data_processing $\leftarrow$ Data), which will be ignored in our approach.

category, and all other categories have at least one parent. The semantics of the hierarchical relation is not specified, and individual relations may be of different ontological nature. The main types of relations are the following:

1. subset relation between the set categories, e.g.

   $\mathsf{Countries\_in\_Europe} \leftarrow \mathsf{Baltic\_countries}$,

2. membership relation between a set category and a topic category, e.g.

   $\mathsf{Countries\_in\_Europe} \leftarrow \mathsf{Albania}$,

3. part-of relation, usually, between two topic categories, e.g.

   $\mathsf{Scandinavia} \leftarrow \mathsf{Sweden}$,

4. sub-topic relation between two topic categories, e.g.

   $\mathsf{European\_culture} \leftarrow \mathsf{European\_folklore}$,

5. other relations, whose nature may be specified the sub-category label, e.g.

   $\mathsf{Europe} \leftarrow \mathsf{Languages\_of\_Europe}$

   (here the label "languages of" implies a relation between a geographic area, namely Europe, and the individual languages, such as Italian and German, that are members of Languages_of_Europe).

Formalizing these relations in description logics leads to the following activities:

Definition of the signature: For each set category one should introduce a class. For example, Country_in_Europe, Baltic_countries, and Language_of_Europe should be declared as classes. For each topic category one should introduce an individual. Thus, Albania, Sweden, Scandinavia, European_folklore, and European_culture should be declared as individuals. Then, the relations between the entities should be declared, such as part_of and subtopic_of. Similarly, the relation between a geographic area and a language is the fact that the language is spoken in that area, and one should declare the relational symbol spoken_in to formalize it.

Definition of the axioms: Finally, the actual relations contained in the Wikipedia category system should be transformed into the axioms of description logic as follows:

- $\mathsf{Baltic\_countries} \sqsubseteq \mathsf{Country\_in\_Europe}$,
- $\mathsf{Country\_in\_Europe(Albania)}$,
- $\mathsf{part\_of(Sweden,\ Scandinavia)}$,
- $\mathtt{subtopic\_of}(\mathsf{European\_folklore,\ European\_culture})$,
- $\mathsf{Language\_of\_Europe} \sqsubseteq \mathsf{Language} \sqcap \exists\ \mathtt{spoken\_in}.\{\mathsf{Europe}\}$.

In the rest of the chapter we propose an automatic method for extracting domain ontology skeletons from the category network of Wikipedia. At the high level, the method consists of the following three steps:

1. selecting the relevant subset of categories;
2. transforming each category into either a class or an individual;
3. establishing the semantic relations between the categories.

Our solution relies on the following simplifying assumptions:

1. relations `subtopic_of` and `part_of` are merged into a unique relation `part_of`,
2. relations other than `part_of`, `instance_of`, and `subclass_of` are codified as a single generic relation `related_to`[4].

4.3 Solution

Each of the three steps of our method—selecting the relevant categories, splitting them into classes and individuals, and classifying the relations—is cast into a binary classification problem. More precisely, the problem at step 1 reduces to discriminating between relevant and irrelevant nodes, the one at step 2—to discriminating between classes and individuals, and that at step 3—to discriminating between a specific relation (such as `subclass_of`) and the generic `related_to`. Solving the problem at each step requires providing manually annotated examples. Step 1, for instance, requires examples of relevant and irrelevant categories, similarly the other two steps. The rest of the work is done by a machine learning algorithm. In the following sections we provide the detailed description of the three steps of the method.

4.3.1 Selecting the subgraph of relevant topics

In order to identify the set of relevant topics, we first select the most general root category representing the domain of interest (henceforth referred to as "the root"). For the computer science domain, for instance, we choose the category Computing[5]. One commonsense observation is that all the relevant categories should be among the sub-categories of the root. Therefore, we should be able to find all relevant topics by recursively following the sub-category links, starting from the root (e.g via breadth-first traversal). Another observation is that a direct sub-category of a relevant category should too be relevant. If this observation were perfectly accurate, by transitivity, all the descendants of the root would be relevant. This latter, however, is not true: it is quite easy to arrive at an irrelevant category by following the sub-category links long enough. Consider, for instance, how Computing is connected to Buddhism_in_China via a chain of sub-category links:

[4]Discovering specific relations of other types is out of scope of this work, and can be a matter of future studies.
[5]`http://en.wikipedia.org/wiki/Category:Computing`

> Computing ← Computer_science ← Areas_of_computer_science ← Artificial_intelligence ← Problem_solving ← Abstraction ← Philosophy ← Philosophy_by_region ← Chinese_philosophy ← Buddhism_in_China.

The algorithm should, therefore, identify the irrelevant categories and exclude them from the recursive traversal. As the likeliness of a category being relevant generally decreases with the distance from the root, the breath-first traversal tends to visit most relevant categories first. In order to ensure the termination of the algorithm, we set a limit `max_depth` on the maximum allowed traversal depth—that is the maximum number of links between any examined category and the root. Based on our experience with the Wikipedia category network, we claim that with `max_depth` set to a reasonably high value, e.g. `max_depth`=20, any category that is further than `max_depth` from the root can safely be assumed irrelevant. The given estimate for `max_depth` is a conservative one, and in practice one can choose a much lower value depending on the domain. For the domain of `computing` we empirically discovered that `max_depth`=7 with high confidence. A final observation is that, assuming the algorithm is perfectly accurate at identifying the irrelevant nodes, and with `max_depth` being reasonably high, the condition on the depth will be redundant, as the algorithm will terminate before reaching the specified depth.

Algorithm 2 formally defines the selection procedure.

Algorithm 2: Selection of the relevant categories.

```
input : root: the root category; max_depth: maximum traversal depth
output: relevant: the set of relevant categories
1  queue ← empty FIFO queue                              // categories to be visited
2  visited ← {root}                                      // categories already visited
3  relevant ← ∅                                          // empty set
4  queue.push(root)                                      // add root to the queue
5  while queue is not empty :
6      category ← queue.pop()
7      if isRelevant(category) :
8          relevant ← relevant ∪ {category }
9          if getDepth(category) < max_depth :
10             for subcategory in getSubcategories(category) :
11                 if subcategory not in visited :
12                     queue.push(subcategory)
13                     visited ← visited ∪ {subcategory }
```

Classifying nodes into relevant and irrelevant. An important step of the algorithm is deciding whether a certain category is relevant (call to `isRelevant` in line 7 of Algorithm 2). There are a number of properties that are correlated with relevance, such as the depth of the category (its distance from the root), the number of the parent categories it has and their distances from the root, and so on. As it is difficult to combine these properties into an explicit hand-coded rule, we train a binary classifier to predict if a category is relevant based on these properties. We used the standard L_2-regularized linear SVM [27] with hinge loss as the classifier.

Training the classifier requires providing examples of both relevant and irrelevant categories. In order to collect the training examples, we first run the simple breath-first traversal of the category graph starting from the root, and limiting the depth to `max_depth`. This results in the initial set of categories that are under the root category and within the distance of `max_depth` from it. From this initial set we randomly sample paths going down from the root and add the categories along these paths to the training dataset. The rationale behind this sampling procedure is the following. Most of the categories at the distance `max_depth` from the root are irrelevant. The root being obviously relevant, transitions from relevant to irrelevant will often happen somewhere along the paths going down from the root. Including the entire paths into the training set ensures that a fraction of the training examples comes from the boundary between relevant and irrelevant categories, which is what we need for learning to discriminate between these two cases.

Computing the features. An important detail is the definition of features used by the classifier. Intuitively, the features should encode the information that helps discriminating relevant from irrelevant categories. The features we implemented describe the position of the category in the graph. The main feature is the category depth, that is, its distance to the root. Generally speaking, the relevance of the categories decreases with depth.

Another observation is that irrelevant categories tend to have irrelevant parents. For example, the category Museums is linked to Computing with the following chain:

$$\text{Computing} \leftarrow \text{Information\_technology} \leftarrow \text{Information\_science} \leftarrow$$
$$\leftarrow \text{Information\_storage} \leftarrow \text{Museums}.$$

The parent categories of “Museums” are “Buildings and structures by type”, “Heritage organizations”, “Educational buildings”, “Educational institutions”, “Museology”, “Visitor attractions”, and “Information storage”, only the last of which is arguably relevant. The relevance status of the parents categories is not available during classification, therefore we cannot use it directly as a feature. As a proxy for this information, we compute the fraction of the parent categories that have been visited by the selection algorithm (and have thus already been identified as descendants of the root category), as well as their minimum, maximum and average depth.

A single text-based feature we use is the maximum similarity of the category title to its parents' titles, measured as the Jaccard index between the sets of stemmed words.

Implications of the sequential selection of categories. An unusual property of the described category selection procedure is that the decisions for different categories are not taken independently. A category is processed by the classifier only if at least one of its parent categories has previously been classified as relevant. The cost of misclassification is thus higher for the higher-level categories (those closer to the root). Furthermore, misclassifying a relevant category has higher impact than misclassifying an irrelevant one: by discarding a relevant category we implicitly discard its sub-categories as well (although they may still be visited from other parents); accepting an irrelevant category, we may still classify its sub-categories as irrelevant. We convey this knowledge to the classifier during training by introducing misclassification costs which depend on the example being misclassified: the cost of predicting a relevant category as irrelevant is set as twice the cost of the opposite error. Furthermore, we use instance-based factors in misclassification costs that decrease with the distance of the category being classified from the root.

4.3.2 Discriminating between classes and individuals

Having selected the set of relevant categories, we need to split them into classes and individuals. Following our general approach, we view this task as a classification problem. We collect a sample of relevant categories, annotate them manually as either classes or individuals, and use the sample to train a binary classifier. The categories can be classified independently from each other, which makes this task easier than the one described in the previous chapter. The features used by the classifier encode various properties of the category that are indicative of its type. We tried to minimize the effort and resources required for computing the features in order to make our approach lightweight and easily reusable. In particular, for this task we limited the set of features to those describing the title of the category. One of the most indicative properties is the grammatical number (singular or plural) of the head word of the title. To utilize this property, we encoded the suffixes of length 1, 2, and 3 of the head word of the title. This set of features is simpler than deriving the grammatical number with a part-of-speech tagger: it is not prone to tagging errors, and can capture more subtle cases, such as learning that suffix *-are* in words like “software” and “hardware” is indicative of class. This limited set of features proves sufficient for achieving decent performance in this task.

4.3.3 Classifying the relations between the nodes

In the previous steps we selected a set of relevant categories from Wikipedia and transformed them into ontology classes and individuals. At this step we need to establish the semantic relations between the entities in the ontology. We introduce a relation between the two nodes of the ontology whenever there is a sub-category relation between the corresponding Wikipedia categories. By default, the introduced relation has a generic type, which we denote as `related_to`. More specific relation types can be defined for certain combinations of node types. Specifically, two classes may be linked with `subclass_of` relation, two individuals with `part_of`, and an individual and a class with the `instance_of` relation. Table 4.1 summarizes the specific relations depending on the node types. In these settings, predicting the type of relation between the given two nodes is equivalent to predicting whether the relation is specific or generic, which can be viewed as a binary classification problem. In case of specific relation, the exact relation type is determined by the node types according to Table 4.1.

Table 4.1: Specific relation types depending on the types of the nodes

parent type	child type	specific relation type
class	class	`subclass_of`
class	individual	`instance_of`
individual	class	—
individual	individual	`part_of`

For each of the specific relation types we train a separate classifier, using a dedicated training set. The training sets are collected by sampling pairs of relevant categories linked with sub-category relations. The node types predicted at the previous step are used to filter out the category pairs that are inconsistent with the specific relation type (according to Table 4.1).

As previously, we only use features that describe the titles, or, more precisely, the relation between the titles of the parent and the child categories, namely:

- whether there is a verbatim match between:
  - the stems of the head words,
  - the stems of the first words,
- the Jaccard similarity between the set of stems,
- similarity between the *head words* (for a number of similarity measures),
- *average pairwise* similarity between the words,
- whether there is a hyponym relation between the words in the titles:
  - whether the relation is between the head words,
  - whether the relation is between a head and a non-head word,
  - whether the relation is between the non-head words,
- whether the two titles are related with a certain pattern, e.g.:
  - one title is included in the other,
  - the titles end with the same phrase, etc..

For similarity between the words we took the maximum similarity between the possible senses of the words, according to WordNet. We used a number of similarity measures based on WordNet, as implemented in the NLTK[3] package. These included the similarities measures due to Leacock and Chodorow [49], Wu and Palmer [106], Resnik [82], Lin [55], and Jiang and Conrath [43].

4.4 Evaluation

First, we applied our approach to building an ontology skeleton for the domain of computing. We executed and evaluated the 3 steps of our method on the categories of Wikipedia:

1. selecting the subgraph of relevant categories,
2. classifying categories into classes and individuals,
3. classifying relations (
   (a) `subclass_of` versus `related_to`,
   (b) `instance_of` versus `related_to`,
   (c) `part_of` versus `related_to`).

For each of the tasks 1–3, we manually annotated a sample of "ground truth" data for both training and evaluation. The performance of the tasks was evaluated in cross-validation, and the parameters of the classifiers were tuned with a nested cross-validation on the training parts of each fold. We will now describe the experiments in more detail.

4.4.1 Evaluation of the relevant subgraph selection

We started from the Wikipedia category Computing[6] as the root, and executed the breadth-first selection procedure described in Section 4.3.1. The 7-level deep selection produced 28 264 categories, from which a random sample of 1 000 categories was selected for labeling. Two experts independently annotated the categories in the set as "relevant" or "irrelevant", omitting the dubious cases. One annotator provided the labels for 994, and another—for 721 categories of the 1000. The 642 categories for which the two annotators gave the same label were selected as the ground truth, and contained 264 relevant and 378 irrelevant categories according to the labeling.

We evaluated the category selection using 5-fold cross-validation. For each fold, the category classifier was trained on the training part of the fold and plugged into Algorithm 2. The output of the algorithm was then evaluated on the test part of the fold. As performance measures we used the accuracy of the prediction, the F1 scores with respect to both relevant categories (positive class) and irrelevant categories (negative class), and the sum of the F1 scores weighted according to the class sizes. Table 4.2 summarizes the mean and the standard deviation of the performance scores across the 5 folds.

To put these results into perspective, we implemented a number of baseline algorithms. The most effective, **depth-based baseline** marks all nodes within a certain distance from the root as relevant, leaving out all the rest. Figure 4.1 shows the accuracy of this baseline depending on the selection depth. Table 4.2 includes the performance scores for depth 3, 4 and 5, which are the best. As seen from the figure and the table, the best performance of the depth-based selection (with depth equal 4) almost reaches that of our method.

[6]`http://en.wikipedia.org/wiki/Category:Computing`

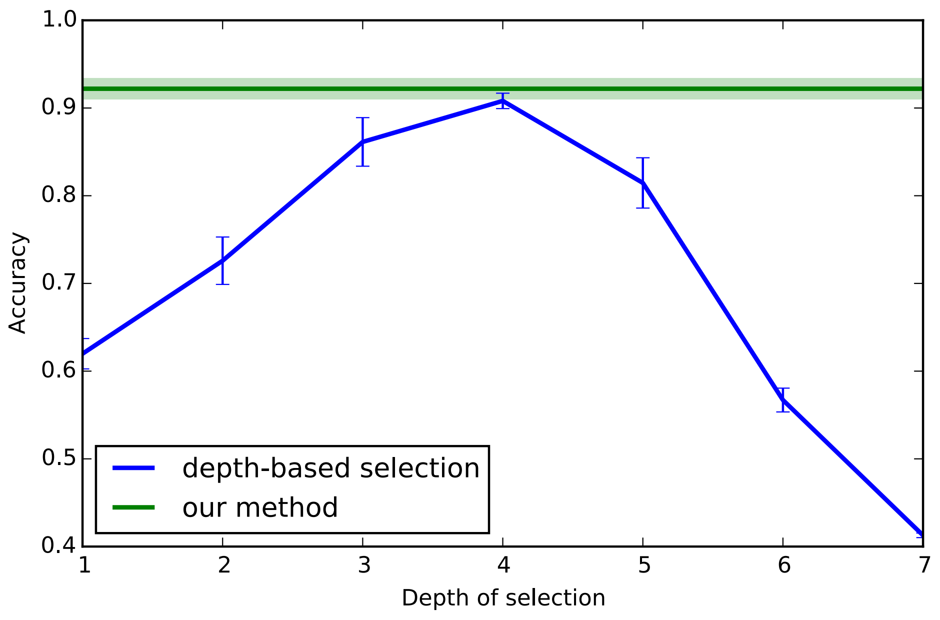


Figure 4.1: Accuracy of the depth-based baseline depending on the depth. The green horizontal line represents the performance of our method. The green band and the vertical blue bars represent the standard deviation of the performance score as per cross-validation.

Table 4.2 also contains the performance scores of two “dummy” baselines that we implemented for additional comparison. The **majority rule** always predicts the class that was more frequent in the training set. This baseline always has zero recall, and therefore zero F1 measure for the smaller (minority) class. The second, **stratified random** baseline predicts the class randomly, but respecting the distribution of the classes in the training set. These two baselines are most useful for comparison when the classes are imbalanced, as their performance tends to the perfect one as imbalance gets stronger.

Table 4.2: Performance of the category selection for the domain of Computing. The scores are in the form “mean (standard deviation)” and are given in per cents.

method	accuracy	F1 pos.	F1 neg.	weighted F1
Ours	92 (0.6)	91 (0.8)	94 (0.5)	92 (0.6)
Depth(3)	86 (2.7)	80 (4.7)	89 (1.9)	86 (3.1)
Depth(4)	91 (0.8)	89 (1.3)	92 (0.6)	91 (0.9)
Depth(5)	81 (2.8)	81 (2.4)	81 (3.3)	81 (2.9)
Majority	59 (0.4)	0 (0)	74 (0.3)	44 (0.5)
Random	53 (7.6)	38 (8.4)	60 (4.8)	55 (6.0)

Figure 4.2 presents the learning curve for the two performance scores (accuracy and weighted F1), showing that as few as 30 categories suffice to train the selection procedure.

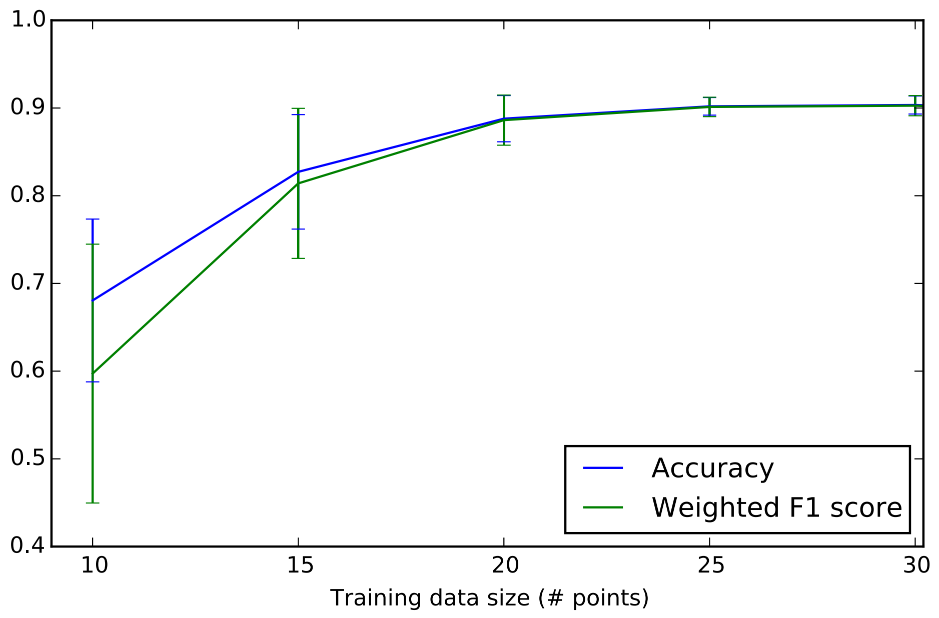


Figure 4.2: Performance scores of the category selection procedure depending on the size of the training data set (accuracy and the weighted F1 measure). The vertical bars represent the standard deviation of the performance scores as per cross-validation.

After evaluating the category selection using cross-validation, we trained the classifier on the whole ground truth dataset, and executed Algorithm 2 from scratch with the re-trained classifier. This final run of the algorithm produced a set of 7159 categories, which we then used in the subsequent stages of the overall procedure.

Evaluating the coverage of the selected categories on ACM CCS. ACM Computing Classification System[7] (ACM CCS) is a standard and widely-used categorization of computer science. Its current version (as of February 2015) contains over two thousand concepts, and can be seen as a gold standard for topics relevant to computing.

We evaluated the results of our selection procedure by estimating how well they cover the concepts of ACM. For the purpose of this experiment, we established a partial mapping between the concepts of ACM and the categories of Wikipedia. The mapping was performed using an automatic ontology matching technique [44], and produced 398 pairs of the corresponding topics (the mapping technique is approximate and only produces a subset of the pairs that truly match). Assuming ACM CCS only contains relevant topics, all mapped Wikipedia categories were considered relevant in this experiment. Of the 398 categories relevant according to the mapping, 327 were also marked as relevant by our algorithm, corresponding to the coverage (or recall) of **0.82**.

[7] `https://www.acm.org/about/class/2012`

We should note that the question of which topics are relevant to a given domain is subjective to a considerable extent. Consequently, there was some disagreement between ACM CCS and the ground truth annotations on which our algorithm was trained. For instance, ACM CCS contained the following concepts irrelevant according to our manual labeling: Reference, Measurement, Cartography, Reference_works, Documents. Accordingly, even our ground truth labeling had a non-perfect recall, estimated at **0.90**. This suggests that the computed recall potentially underestimated the performance of our algorithm.

We should also note that we could not estimate the precision of the selected categories in the same way, firstly because ACM CCS does not contain the corresponding topics for all relevant Wikipedia categories, and secondly because our mapping between ACM CCS and Wikipedia was only partial, and only contained a subset of all matching topic.

There were a total of 71 mistakes—ACM concepts marked irrelevant by our algorithm. The minimal depth of the mistaken categories was 4. Table 4.3 shows a subset of mistakes—the 35 mistaken categories that had depth up to 5. While some of the categories in the table are clearly relevant to computer science (genuine mistakes), some other categories, such as Trademarks and Temperature_control, are arguably truly irrelevant.

Table 4.3: Topics of ACM labeled as irrelevant by our algorithm (depths 4 and 5 only).

depth	title	depth	title
4	Ergonomics	5	Trademarks
4	Graph theory	5	Support vector machines
4	Information theory	5	Neural networks
4	Computer-aided design	5	Network flow
4	Reference	5	Stochastic control
4	Temperature control	5	MP3
4	Mobile phones	5	Coding theory
4	Cartography	5	Robust regression
5	Graph coloring	5	Survival analysis
5	Latent variable models	5	Measurement
5	Random graphs	5	Generating functions
5	Matching	5	Bayesian networks
5	Tracking	5	Go software
5	Heuristics	5	Mixed reality
5	Quantum information theory	5	Extremal graph theory
5	Graph enumeration	5	Years in robotics
5	Typing	5	Virtual On
5	Stochastic differential equations		

4.4.2 Evaluation of the node type classification

Having selected the subgraph of categories relevant to Computing, the next step was to group the nodes into classes and individuals. From the 7 159 categories obtained in the previous step, we randomly selected a sample of 270 categories for manual annotation, and used the sample for training and evaluation. Similarly to the previous step, we used 10-fold cross-validation to evaluate the performance of this step. Table 4.4 summarizes the performance scores of our method, as well as the two "dummy" classifiers.

Table 4.4: Performance of the category type classification for the domain Computing. The scores are in the form "mean (standard deviation)" and are given in per cents.

method	accuracy	F1 pos.	F1 neg.	weighted F1
Ours	95 (4.1)	96 (3.3)	92 (5.4)	95 (4.0)
Majority	66 (1.0)	80 (0.7)	0 (0)	52 (1.2)
Random	54 (8.9)	65 (6.1)	40 (13.4)	61 (6.6)

Comparison with WikiTaxonomy. WikiTaxonomy[80] is a large-scale taxonomy derived from the Wikipedia category network, and containing over a hundred thousand categories. Similarly to our work, WikiTaxonomy discriminates between classes and individuals, and establishes `subclass_of` and `instance_of` relations between them. We looked at the classes and individuals of WikiTaxonomy as an alternative classification of categories, and measured its performance on our ground truth dataset. As our dataset was collected from a recent version of Wikipedia, it included some new categories that were not present in WikiTaxonomy. For the purpose of comparison, we limited the test set to the 75 categories that WikiTaxonomy did contain. The performance scores of our method and WikiTaxonomy, as measured on this dataset, were **0.92** and **0.54** respectively.

We should point out that the proportions of classes in our manual labeling and in WikiTaxonomy are dramatically different. In WikiTaxonomy only about 8 per cent of the categories are classified as individuals. In contrast, according to our labeling, individuals account for as much as a third of all categories. This suggests that our notions of classes and individuals and the criteria for assigning categories to these groups are likely different from those used in WikiTaxonomy.

Most of the errors of our method in this task were related to difficulties in distinguishing between the plural and the singular forms in the category titles. For instance, Robotics, Bioinformatics, and Computer graphics were mistakenly labeled as classes. With few exception, the "errors" of WikiTaxonomy on this task were in predicting too many classes. Examples of such mistakes (according to our ground truth labeling) include Game theory, Data management, World Wide Web, Cryptography, and Graph coloring.

4.4.3 Evaluation of the relation type classification

We evaluated the task of classifying the relation types in a way similar to that described in the previous sections. For each of the three specific relation types (`subclass_of`, `instance_of`, and `part_of`) we evaluated the corresponding binary classification task of discriminating between that specific relation and the generic `related_to`. For each of the three subtasks we collected and manually labeled a "ground truth" dataset of parent-child category pairs. Similarly to the previous steps, we evaluated the performance in a 10-fold cross-validation. We tuned the parameters with a nested 10-fold cross-validation on the training part, and evaluated the performance on the test part of each fold.

Table 4.5: Performance of the relation type classification for the domain Computing. The scores are in the form "mean (standard deviation)" and are given in per cents.

method	accuracy	F1 pos.	F1 neg.	weighted F1
		`subclass_of`		
Ours	80 (9.0)	87 (6.0)	48 (29.4)	83 (7.6)
WikiTx	66 (10.8)	74 (10.9)	39 (24.5)	71 (9.4)
Majority	85 (9.8)	92 (6.0)	0 (0)	79 (13.6)
Random	79 (8.6)	87 (5.8)	15 (24.0)	78 (12.2)
		`instance_of`		
Ours	67 (11.9)	71 (16.8)	55 (14.5)	67 (10.9)
WikiTx	68 (13.8)	66 (13.6)	66 (15.4)	68 (14.4)
Majority	59 (15.5)	73 (15.2)	0 (0)	45 (16.0)
Random	53 (13.2)	61 (15.4)	37 (14.6)	52 (13.2)
		`part_of`		
Ours	59 (13.4)	63 (23.0)	32 (32.9)	62 (15.4)
Majority	69 (26.2)	79 (19.5)	0 (0)	59 (33.1)
Random	62 (17.0)	70 (12.9)	33 (34.8)	66 (17.0)

For the subtasks of discovering `subclass_of` and `instance_of` relations, we used WikiTaxonomy as a natural alternative method to compare with (similarly to what we described in the previous section). For the sake of comparison, we measured the performance of our method on the fraction of pairs from the test data, for which both parent and child categories were contained in WikiTaxonomy. In order to have enough such pairs for the comparison, we specifically sampled a number of them and included into the dataset for manual annotation. The performance of WikiTaxonomy on the `subclass_of` relation was measured in the following way: if for a certain parent-child pair of categories from the test set either `subclass_of` or `instance_of` relation was found in WikiTaxonomy, we considered that WikiTaxonomy predicted the `subclass_of` relation between these categories; if no relation was found, we considered that WikiTaxonomy predicted the generic `related_to` relation. The performance of WikiTaxonomy on the `instance_of` task was measured in the same way. Table 4.5 summarizes various statistics about the training and test sets, and shows the performance scores of both WikiTaxonomy and our method, as well as the "dummy" baselines. As we can see from the table, our method performs well on the `subclass_of` relation and comparable to WikiTaxonomy on the `instance_of` relation. On the `part_of` relation, due to the difficulty of the task and the class imbalance, our method performs comparably to the random baseline.

4.4.4 Evaluation on a different domain

In order to evaluate the generalization of the proposed method, we executed it on the domain of `Music`. The only detail we had to adapt was the `max_depth` parameter of Algorithm 2. For this domain we empirically chose the value `max_depth`=9. We sampled the nodes at different depths and performed a statistical test, according to which, with 95% confidence, no more than 6% of nodes at distance 9 from category `Music` are relevant to music. Table 4.6 summarizes the results of the experiments. The best performance was achieved when the classifiers were trained on the annotated data from the new domain (`Music`). However, it is interesting to see that the method performed reasonably well on the first two tasks even with the classifiers trained on the `Computing` domain.

As we can see from the table, our methods performed well on the tasks of selecting the relevant categories, classifying the category types and classifying the `subclass_of` relation, and poorly on classifying the other two relation types. As in the case of other domain, the baseline depth-based selection performed well, when the depth was selected appropriately. In case of `Music` the optimal depth turned out to be equal 6. Note that our machine learning–based method was not informed about this optimal depth and selected the nodes through the automatic procedure of Algorithm 2.

We can also see from the table that in case of the `Music` domain, the performance of our method on the `instance_of` relation is significantly worse, and also worse than that of the baseline "dummy" methods. The main reason for this is that in this domain we could hardly find negative examples for this type of relation (4 negative examples in a sample of 54). It is clear that in this case it is "safe" to always predict the positive class (the majority rule). The results on this relation could be improved by selecting a larger sample containing more negative examples. The `part_of` relation, on the other hand, is intrinsically more difficult, and likely requires more sophisticated features based on semantics and background knowledge.

4.5 Related Work

In this chapter we describe a method for bootstrapping domain ontologies from the category hierarchy of Wikipedia. While there is a large number of works that derive semantic

Table 4.6: Performance of the various tasks on the domain Music. The scores are in the form "mean (standard deviation)" and are given in per cents.

method	accuracy	F1 pos.	F1 neg.	weighted F1
category selection				
Ours	89 (0.9)	92 (0.7)	79 (1.4)	88 (0.9)
Ours*	86	90	78	89
Depth(6)	89 (2.9)	92 (2.0)	79 (5.8)	88 (3.1)
category type classification				
Ours	98 (2.0)	99 (1.2)	92 (6.6)	98 (2.0)
Ours*	98 (2.0)	99 (1.2)	92 (6.6)	98 (2.0)
Majority	85 (0.6)	92 (0.3)	0 (0)	79 (0.8)
Random	76 (4.7)	83 (0.3)	17 (8.3)	73 (2.9)
relation `subclass_of`				
Ours	95 (5.5)	97 (3.3)	71 (83.0)	96 (4.6)
WikiTx	60 (10.8)	71 (11.0)	31 (14.5)	68 (10.8)
Majority	90 (5.5)	95 (3.1)	0 (0)	86 (8.0)
Random	84 (6.3)	91 (3.9)	4 (12.0)	82 (7.9)
relation `instance_of`				
Ours	71 (29.7)	73 (37.1)	12 (21.3)	72 (35.1)
WikiTx	41 (23.2)	50 (29.0)	13 (20.4)	49 (28.1)
Majority	93 (8.6)	96 (4.7)	0 (0)	90 (12.5)
Random	84 (15.8)	91 (10.2)	0 (0)	85 (14.0)
relation `part_of`				
Ours	45 (21.0)	47 (33.9)	17 (22.3)	44 (25.9)
Majority	80 (12.3)	88 (7.9)	0 (0)	71 (16.7)
Random	67 (16.1)	77 (15.1)	12 (18.4)	66 (15.8)

* Trained on `Computing`

knowledge from Wikipedia, very few of them attempt to refine the information available in its category structure.

4.5.1 WikiTaxonomy

WikiTaxonomy represents the most relevant line of work in this direction. The first work in this line [80] was concerned with identifying the `is_a` relation between categories (corresponding to the union of our `subclass_of` and `instance_of` relations). The method consists of several steps, at which a number of heuristic rules are applied. Each step classifies a fraction of relations as either `is_a` or `not_is_a`, while the remainder is passed to the next step. Syntax-based rules compare the titles of the categories, looking, in particular, at their lexical heads. A matching between the two heads indicates an `is_a` relation, while the head matching the non-head part indicates a `not_is_a` relation. Connectivity-based rules examine the Wikipedia articles belonging to the categories in question. For instance, a relation between two categories is classified as `is_a` whenever there is an article belonging to both categories. Lexico-syntactic rules, applied next, mine the evidence for `is_a` and `not_is_a` relations from large text corpora using a number of predefined patterns. The rules are combined via majority voting to classify the relations. The results of the majority voting are also used to correct the output of the previous step. Finally, a number of additional `is_a` relations are discovered by propagation.

In a successive paper [111] the authors proposed a method for discriminating between the individuals and classes in WikiTaxonomy. Similarly to the previous work, the method applies a number of heuristic rules that examine the title of the category, as well as its connections to other categories and Wikipedia articles. The authors evaluate three deterministic schemes for combining the rules, based on their cross-validation performance. Title-based rules look at the capitalization, the grammatical number, and the presence of named entities. Other rules look at the `is_a` relations of the category, and whether it contains an article with the same title.

Similarly to these two works, our method uses the information in the category titles to classify both categories and relations between them. The syntactic features of the titles implemented in our method are largely inspired by the rules used in WikiTaxonomy. An important distinction of our work is the use of machine learning. In each of the tasks, various pieces of evidence are taken as features, re-weighted and combined by the supervised classifier rather then being encoded as hard rules. This allows us to avoid the multi-step refinement process and manually encoded rule combination schemes. Our approach is, therefore, much more flexible: incorporating a new heuristic or an external knowledge source amounts to simply adding a new feature to the classifier.

An additional distinction is that in the tasks of classifying the categories and relations, we have relied only on the information in the category titles. In particular, we have not used the information about Wikipedia articles, the connections between the categories[8], and external text corpora. We have, however, used WordNet to improve the performance of relation classification. Relying only on the title-based features makes our approach potentially applicable in non-Wikipedia scenarios, which has yet to be verified. In case of Wikipedia, although we could have used a much richer set of features, the title-based information was sufficient to achieve a reasonable performance on our ground truth data. Lastly, WikiTaxonomy is a general-purpose taxonomy that covers the whole Wikipedia, and is, thus, not concerned with domain specificity. Intended for building domain ontologies, our method also addresses the problem of selecting the relevant subset of categories.

[8]We used the connectivity information in the task of selecting the relevant categories.

4.5.2 Extracting rich semantic knowledge from Wikipedia

Wikipedia is an extraordinarily rich source of knowledge, and many works attempt to extract and re-integrate the information contained in its various parts.

DBpedia [50] extracts knowledge available in Wikipedia in semi-structured format, such as infoboxes, between-language links and redirects, and makes it available in RDF format. The central entities in DBpedia correspond to Wikipedia articles. The properties in the infobox of an article instantiate specific relations linking the article to the value of the property. For instance, the infobox in the article about Trento city, specifies (among other) that the patron saint of Trento is St.Virgilius. This will be transformed into a relation equivalent to `patron_saint_of`(St._Virgilius, Trento). The Wikipedia categories are not used as classes to group the entities. Instead, DBpedia relies on a manually curated ontology that was created from the most commonly used infoboxes, and contains (as of February 2015) several hundred classes and over two thousand properties. The category network is exported in SKOS format, preserving the hierarchical structure, without being semantically enriched in any way.

YAGO [39, 97] similarly extracts structured information from Wikipedia in the form of facts about entities that represent Wikipedia articles. In contrast to DBpedia, it integrates WordNet [65], and relies on its taxonomical structure to form the class subsumption hierarchy. From Wikipedia categories, only the leaf ones are taken into consideration, and a simple heuristic is used to identify the leaf categories that represent classes. In addition to the relations extracted from infoboxes, some relations are derived from the so-called relational categories. For example, an article belonging to the category `1879_births` will generate an instance of the `born_in_year` relation. The set of relations extracted from the relational categories is predefined.

In [79] Ponzetto and Navigli used the taxonomical structure of WordNet to improve the category structure of WikiTaxonomy. The idea of exploiting the relational Wikipedia categories is taken even further by WikiNet [74]. Careful analysis of the Wikipedia category names, combined with the infobox information, allowed the authors to automatically extract 454 relation types with over 49 million individuals. Notably, WikiNet uses Wikipedia as its only data source.

The Catriple [57] system also uses Wikipedia category names to generate facts about the articles, with new property types being extracted automatically. In contrast to other systems, Catriple observed the patterns in the titles of the parent-child category pairs. For instance, the two categories, Songs_by_artist and The_Beatles_songs, suggest that "artist" is a property, and "The Beatles" is a value. The articles under The_Beatles_songs can, thus, be annotated with this newly generated property. Another distinction of Catriple is that it does not rely on the infobox templates.

The main distinction of the described methods is that they focus on Wikipedia articles as entities, and mainly intend to derive new knowledge, such as facts and relations describing the articles. The purpose of our work is to extract ontological information about classes and individuals represented by Wikipedia categories. Most of the described methods also rely on various aspects of Wikipedia, such as article pages and infobox templates. In contrast, our method works with the categories alone, and can potentially be applied to other concept hierarchies.

Chapter 5

Reducing the labeling effort in search-based structured prediction

In this thesis we suggest facilitating academic search by representing the query results in a concise, structured, and visual way. Our approach to this problem, presented in the previous chapters, relies on the supervised machine learning techniques to build large structured objects. The two specific tasks in the previous chapters that we reduced to supervised learning were:

1. building informative summaries of topics maps, and
2. extracting domain ontologies form Wikipedia.

Providing the *supervision* (the training set) for structured prediction tasks is usually nontrivial. For the two particular tasks we dealt with, providing the complete structured annotations was especially tedious, or even completely impractical, due to a combination of the following factors:

(a) complexity of the structured examples (both inputs and outputs),
(b) the overly large size of the structured output,
(c) the overly large number of alternative labels to select from.

In task (2), for instance, most contributing was the factor (b): a single structured output corresponds to a complete extracted ontology, which, of course, cannot be provided as an annotation due to its size. In task (1), in contrast, the main problems were (a) and (c). Additional effort is required in order to collect the data for annotation.

In this chapter we report on our initial efforts in simplifying the provision of annotations for such structured prediction tasks. Along the lines of active learning, we sought to develop interactive interfaces and algorithms for data labeling, which could:

- minimize the amount and the complexity of information presented to the annotator,
- interactively learn from the annotator's feedback in the online manner (with every piece of provided feedback),

- minimize the number of queries to the annotator, by selecting the most useful queries.

One of the initial results presented in this chapter is the interactive labeling interface for the task of summarizing the topic (Section 3.4). When collecting the next topic in a ground truth summary, the interface presents a small number of suggested candidates, reducing the cognitive load on the annotator. The suggestions are displayed *in the context* of the summary built so far, giving enough information for the informed decision. Accepting or discarding any of the presented topics takes a single click. We describe a learning algorithm that can support the presented labeling interface, which is an *online* modification of the DAgger algorithm that takes one example at a time without re-training from scratch. The modified algorithm has the same regret guarantees as the original DAgger, meaning that the suggested topics improve over time.

The second initial result is the active learning-to-search procedure for selecting the topics from Wikipedia. Supporting this procedure is a modification of the DAgger algorithm for approximate/partial feedback, with provable asymptotic regret guarantees. Unfortunately, as with many active learning algorithms, we do not provide guarantees on the number of queries being less than $O(T)$.

5.1 Interactive labeling of topic map summaries

Consider the problem of learning to summarize topic maps that we introduced in Chapter 3, Section 3.4.1. In this prediction problem, the input is represented by 1) the set of documents (e.g. search results), 2) the topic graph $G[V, E]$ built for these documents, and 3) the topic-document relation R. The output is a summary graph $G_T[V_T, E_T]$. According to the sequential prediction approach described in Section 3.4.1, a ground truth "label" for a given input consists of the topics $v_1, v_2, \ldots, v_T \in V$, such that $G_1 := \{v_1\}$ is the most informative single-topic summary, $G_2 := \{v_1, v_2\}$—the most informative summary of size 2, and so on. Arguably, the simplest way to provide such "label" is by selecting the topics $v_1, v_2, \ldots, v_T$ in order from the graph G.

In principle, when choosing the topic v_{t+1} the annotator needs to consider the whole graph G, imagine the partial summary $G_t := v_{1:t}$, and evaluate every topic $v' \in G \setminus G_t$ as a candidate. To illustrate the typical sizes, the graph G may contain few hundred topics, the partial summary G_t may be anywhere from 1 to 20, with the number of candidate topics being again in hundreds. When evaluating a candidate topic v', the annotator should consider how good is the summary $G_t \cup \{v'\}$ in terms various quality characteristics (see Section 3.1.1). Overall, the task of providing the labels is quite demanding, considering also that the annotator should at least be familiar with the topics in question.

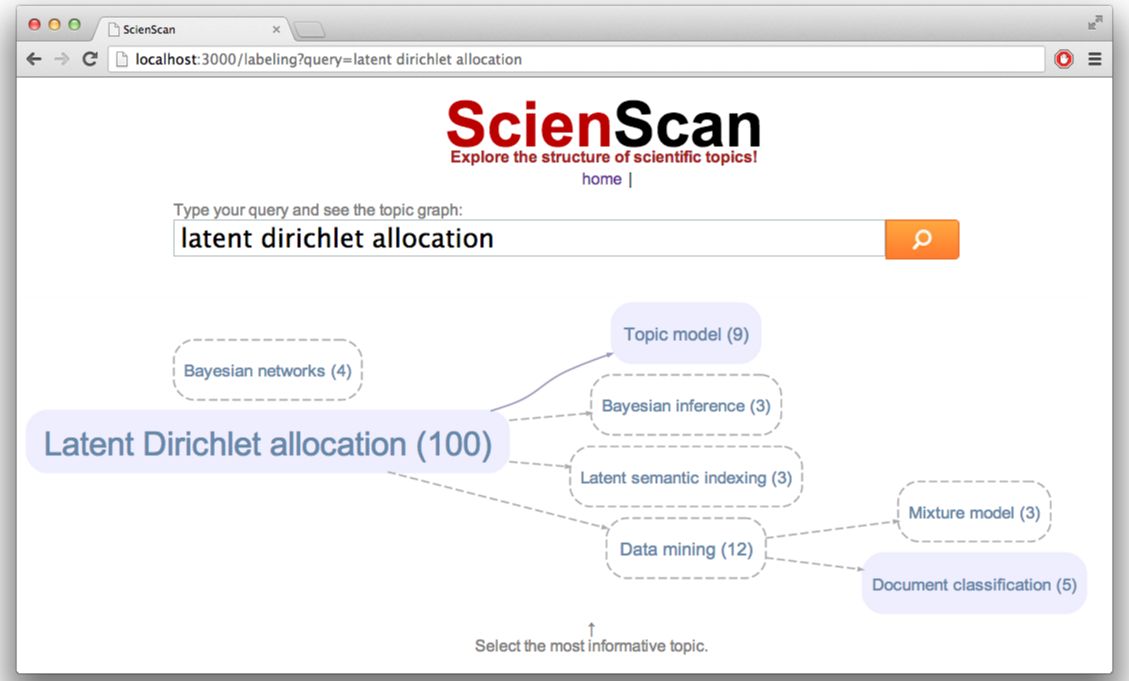

ScienScan
localhost:3000/labeling?query=latent dirichlet allocation
ScienScan
Explore the structure of scientific topics!
home |
Type your query and see the topic graph:
latent dirichlet allocation
Topic model (9)
Bayesian networks (4)
Bayesian inference (3)
Latent Dirichlet allocation (100)
Latent semantic indexing (3)
Mixture model (3)
Data mining (12)
Document classification (5)
Select the most informative topic.

Figure 5.1: The labeling interface of `ScienScan`. The topics with solid background are those selected by the annotator. The topics with the dashed border are the suggested topics.

5.1.1 The labeling interface

In order to facilitate the provision of the ground truth topic summaries, we extended the ScienScan tool (Section 3.2) with the labeling interface (Figure 5.1). The interface works as follows. First, the annotator submits a query on some familiar topic, and the tool retrieves and presents the search results along with the *interactive labeling topic map*. At a given state in the annotation process, the topic map displays the topics $v_1, v_2, \ldots v_t$ selected so far (topics with solid background in Figure 5.1), and the set of candidates for being selected next. The links in the labeling topic map are computed in the same way as for the ordinary topic maps (see Section 3.4.6). The suggested topics are therefore shown in the context of the graph selected so far, and the annotator can easily assess how the topic map will look like after adding any of the candidates. When a topic is selected, the set of candidates is recomputed to take into account the new information. If the annotator considers a certain suggested topic useless, he/she can chose to "hide" or discard it, at which point it is replaced with a new candidate and is never suggested again in this session. Both selecting and hiding a topic takes a single click.

In order to suggest topics, the labeling interface maintains a policy trained on the previously collected sequences. The policy evaluates the graphs $G_t \cup \{v'\}$ for all $v' \in G \setminus G_t$ and suggests the top k highest-scoring topics as candidates.

Complexity of providing labels. We assume that the annotator recognizes the best topic v_{t+1} when he/she sees it. When the model used for suggesting the candidates is good, the topic v_{t+1} will be among the suggested candidates. In this case, evaluating a small set of suggestions is clearly much easier than evaluating all possible topics. When the model is bad, v_{t+1} will not be suggested at once, and in the worst case the annotator will have to look through all possible topics, by discarding the worst suggestions one by one (and having them replaced by new candidates). In the following section we describe the algorithm that can incrementally learn from the annotator's feedback, so that the suggestions improve over time, reducing the effort of selecting the next topic.

5.2 Online DAgger

The *online DAgger* algorithm is similar to the offline DAgger in most respects. (See Algorithm 1 in Section 2.6.1 for the general DAgger algorithm, and Section 3.4.4 for its specific version for summarizing topic maps.) At each iteration, the current policy is applied to the initial states in the training set, producing a trajectory of states per training example. Based on the ground truth trajectories in the training set, the expert actions are generated for every state visited by the new trajectories. The generated state-action pairs are added to the dataset, on which the next policy is trained.

The main distinction of the *online* algorithm is that the complete "ground truth" trajectories are not available at once, and we collect them from the expert throughout iterations, one state at a time. In the following algorithm and the convergence results we assume that we make exactly one iteration per collecting one ground truth state. Suppose for simplicity that we only have one ground truth trajectory $S_1^*, \dots S_N^*$. At iteration i we collect the i_{th} ground truth state S_i^* from the expert. From the ground truth states collected through the first i iterations we can automatically compute the expert actions for the states $0 : (i-1)$ of any trajectory. In order to compute the expert actions for step i, we need to "wait" until the next ground truth state S_{i+1}^* becomes available. Upon receiving the i_{th} ground truth state we, therefore, compute the expert actions for:

1. $\mathcal{S}_{lower}^i$ – the first i states on the trajectory generated by the current policy,

2. $\mathcal{S}_{upper}^i$ – the set of i_{th} states on every trajectory produced by the previous policies.

Tables 5.1 illustrate the sets of states, for which the expert actions are computed at each iteration in the offline and online versions of DAgger.

Table 5.1: The states generated by the policies at each iteration of the offline and the online DAGGER. Alternatively these also represent the states for which the expert actions are generated at each iteration. The alternating colors mark transitions between iterations.

(a) Offline DAGGER

π^1	S_1^1	S_2^1	S_3^1	S_4^1	S_5^1
π^2	S_1^2	S_2^2	S_3^2	S_4^2	S_5^2
π^3	S_1^3	S_2^3	S_3^3	S_4^3	S_5^3
π^4	S_1^4	S_2^4	S_3^4	S_4^4	S_5^4
π^5	S_1^5	S_2^5	S_3^5	S_4^5	S_5^5

(b) Online DAGGER

π^1	S_1^1	S_2^1	S_3^1	S_4^1	S_5^1
π^2	S_1^2	S_2^2	S_3^2	S_4^2	S_5^2
π^3	S_1^3	S_2^3	S_3^3	S_4^3	S_5^3
π^4	S_1^4	S_2^4	S_3^4	S_4^4	S_5^4
π^5	S_1^5	S_2^5	S_3^5	S_4^5	S_5^5

5.2.1 One training example

Denote by $\mathcal{S}^i$ the union of $\mathcal{S}^i_{lower}$ and $\mathcal{S}^i_{upper}$. (In Table 5.1b the sets $\mathcal{S}^i$ for $i = 1 : 5$ are highlighted by alternating colors.) At each iteration of the *online* DAGGER, the dataset of state-action pairs is augmented with the states $\mathcal{S}^i$ and the corresponding expert actions. In order to prove the regret bound similar to that of the offline DAGGER algorithm, we need the states $\mathcal{S}^i$ to be produced by the same policy π^i. The latter requirement brings a change in the way we generate the trajectories with respect to offline DAGGER. Consider the mixture policy π^i_{mix} that generates the first i states of the trajectory $S^i_1, S^i_2, \ldots, S^i_i$ using the policy π^i, and the rest of the states $S^i_{i+1}, S^i_{i+2}, \ldots, S^i_N$ using the policies $\pi^{i+1}, \pi^{i+2}, \ldots, \pi^N$ respectively. To put it differently, with mixture policies $\pi^{1:N}_{mix}$, a state $S^{i_1}_{i_2}$ gets generated by $\pi^{\max(i_1,i_2)}$. If we use policies π^i_{mix} to generate trajectories $S^i_{1:N}$, we will get that all states in $\mathcal{S}^i$ are produced by the policy π^i.

5.2.2 Multiple training examples

Consider the case when we have n training examples (ground truth trajectories). When we have collected the last (T_{th}) ground truth state for the first example, we "roll over" to the next training example. In order to simplify the indexing notation, we imagine that for a given DAGGER policy π^i, the trajectories corresponding to different training examples are concatenated together, so that the last T_{th} state of the k_{th} trajectory is merged with the initial 0_{th} state of the $k + 1_{st}$ trajectory. Thus, in a given trajectory $S^i_{1:N}$ the states $S^i_{0:T-1}$ correspond to the first ground truth example, the states $S^i_{1,2T-1}$ to the second, and so on. With this notation, the definitions π^i, $\mathcal{S}^i_{upper}$, $\mathcal{S}^i_{lower}$, $\mathcal{S}^i$ extend to the case $i > T$.

Table 5.2 shows which states are generated by which policies during the execution of the *online* DAGGER in the first $T(k + 1)$ iterations. The lower and the upper triangles of Table 5.2 contain the states covered by $\mathcal{S}^{1:N}_{lower}$ and $\mathcal{S}^{1:N}_{upper}$ respectively. An interesting consequence of "concatenating" the trajectories in our notation, is that the extended trajectories $S^i_{1:N}$ generated by different policies π^i all coincide in the states $S^i_0, S^i_T, \ldots, S^i_{Tk}$, which correspond to the initial states for various ground truth examples. As a result, the trajectories of the first $Tk + j$ DAGGER policies will have only j different states at step $Tk + j$—meaning that $|\mathcal{S}^{Tk+j-1}_{upper}| = j, \ \forall j \in 1 : T$. Due to the latter observation, the states in the upper triangle blocks of Table 5.2 (those shown as blank) do not have to be computed, as they are covered by the states in the diagonal blocks.

Table 5.2: The states generated by the policies at each iteration of the online DAGGER. The alternating colors mark transitions between iterations.

π^1	S^1_1	S^1_2	S^1_3	$\cdots$	S^1_T											
π^2	S^2_1	S^2_2	S^2_3	$\cdots$	S^2_T											
π^3	S^3_1	S^3_2	S^3_3	$\cdots$	S^3_T											
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$											
π^T	S^T_1	S^T_2	S^T_3	$\cdots$	S^T_T											
π^{T+1}				$\cdots$		S^{T+1}_{T+1}	S^{T+1}_{T+2}	S^{T+1}_{T+3}	$\cdots$	S^{T+1}_{2T}						
π^{T+2}				$\cdots$		S^{T+2}_{T+1}	S^{T+2}_{T+2}	S^{T+2}_{T+3}	$\cdots$	S^{T+2}_{2T}						
π^{T+3}				$\cdots$		S^{T+3}_{T+1}	S^{T+3}_{T+2}	S^{T+3}_{T+3}	$\cdots$	S^{T+3}_{T+3}						
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$						
π^{2T}				$\cdots$		S^{2T}_{T+1}	S^{2T}_{T+2}	S^{2T}_{T+3}	$\cdots$	S^{2T}_{2T}						
$\cdots$				$\cdots$					$\cdots$		$\ddots$					
π^{Tk+1}				$\cdots$					$\cdots$			S^{Tk+1}_{Tk+1}	S^{Tk+1}_{Tk+2}	S^{Tk+1}_{Tk+3}	$\cdots$	S^{Tk+1}_{Tk+T}
π^{Tk+2}				$\cdots$					$\cdots$			S^{Tk+2}_{Tk+1}	S^{Tk+2}_{Tk+2}	S^{Tk+2}_{Tk+3}	$\cdots$	S^{Tk+2}_{Tk+3}
π^{Tk+3}				$\cdots$					$\cdots$			S^{Tk+3}_{Tk+1}	S^{Tk+3}_{Tk+2}	S^{Tk+3}_{Tk+3}	$\cdots$	S^{Tk+3}_{Tk+T}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$		$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
π^{Tk+T}				$\cdots$					$\cdots$			S^{Tk+T}_{Tk+1}	S^{Tk+T}_{Tk+2}	S^{Tk+T}_{Tk+3}	$\cdots$	S^{Tk+T}_{Tk+T}

5.2.3 Defining the policies

The *online* DAGGER algorithm is presented in the listing of Algorithm 3. The algorithm outputs a mixture policy $\pi_{mix}[\pi^{1:N}]$, which we will define shortly. The way we define π_{mix} is guided by the scheme of proving the bound on the imitation loss (similarly, to the proofs in [87]). Recall that DAGGER uses an online learner (such as Follow-The-Regularized-Leader) to generate the policies $\pi^{1:N}$. The loss suffered by the online learner at iteration i is equal[1] to the loss of the policy π^i on the states $\mathcal{S}^i$:

$$L^{Tk+j}_{onl}(\pi) := \sum_{S \in \mathcal{S}^i} l(S, \pi, \pi^*(S)). \tag{5.1}$$

For our proof to work, we need the imitation loss $L_{imit}(\pi_{mix})$ to decompose into the sum of the losses $L^i_{onl}(\pi^i)$ suffered by the online learner. We will define the policy π_{mix} as a probabilistic mixture of the policies $\{\pi^{k,j}_{mix} \mid k \in 0 : (n-1),\ j \in 1 : T\}$. The policies $\pi^{k,j}_{mix}$ and the mixture proportions will be defined in such a way that the states involved in the computation of the online losses $L^{1:N}_{onl}(\pi)$ will be equally probable in the distribution of states generated by π_{mix}.

[1]In Section 2.6.1, we defined the online loss as the loss of the i_{th} policy, *normalized by the number of states.* In this section it is more convenient for the analysis *not to normalize* the online loss.

Algorithm 3: Online DAGGER algorithm

```
1  $D \leftarrow \varnothing$ // empty set
2  $S_0^* \leftarrow S^0$
3  $\pi^0 \leftarrow$ random policy
4  for $i \in 0 : N-1$ do
5      Collect the ground truth action $a_{i+1}^* := \pi^*(S_i^*)$ form the user
6      Generate the sequence $\mathcal{S}_{lower}^i \leftarrow (S_1^i, \ldots, S_i^i) \sim \pi^i$
7      Generate states $\mathcal{S}_{upper}^i \leftarrow (S_i^1, S_i^2, \ldots, S_i^{i-1}) \sim \pi^i$
8      $\mathcal{S}^i \leftarrow \mathcal{S}_{upper}^i \cup \mathcal{S}_{lower}^i$
9      Generate expert actions $\{\pi^*(S) \mid S \in \mathcal{S}^i\}$
10     Build the dataset $D^i \leftarrow \{(S, \pi^*(S)) \mid S \in \mathcal{S}^i\}$
11     Aggregate the datasets: $D \leftarrow D \cup D^i$
12     Train the policy $\pi^{i+1}$ on the dataset $D$
13 end
14 return $\pi_{mix}[\pi^{1:N}]$
```

The definition of $\pi_{mix}^{k,j}$. We define each $\pi_{mix}^{k,j}$ as a probabilistic policy that with probability $\frac{k}{k+1}$ executes π^{Tk+j} for the entire trajectory, and with probability $\frac{1}{k+1}$ executes π^{Tk+j} for the first j steps and then generates the next $T-j$ states by executing the policies $\pi^{Tk+j+1}, \pi^{Tk+j+2}, \ldots, \pi^{T(k+1)}$, one policy per step. Alternatively, $\pi_{mix}^{k,j}$ can be seen as a probabilistic mixture of π^{Tk+j} and π_{mix}^{Tk+j} (defined earlier) with proportions $\frac{k}{k+1}$ and $\frac{1}{k+1}$. Visually, $\pi_{mix}^{k,j}$ corresponds to the $Tk+j_{th}$ line in Table 5.2. The first case—executing π^{Tk+j} for the entire trajectory—corresponds to the first $k-1$ segments of the line (contained in the lower triangle blocks), while the second case corresponds to the last (k_{th}) segment of the line (the diagonal block).

We state without proof the following lemma about the imitation loss of $\pi_{mix}^{k,j}$:

Lemma 1. *The sum of the imitation losses of the policies $\pi_{mix}^{k,1}, \pi_{mix}^{k,2}, \ldots \pi_{mix}^{k,T}$ equals the sum of the losses suffered by the online learner at iterations $(Tk+1) : (Tk+T))$, divided by (k+1):*

$$\sum_{j=1}^{T} L_{imit}(\pi_{mix}^{k,j}) \equiv \frac{1}{(k+1)} \sum_{j=1}^{T} L_{onl}(\pi^{Tk+j}). \tag{5.2}$$

Visually, Lemma 1 can be understood by examining the rows $(Tk+1) : (Tk+T)$ of Table 5.2. The factor $\frac{1}{k+1}$ arises from the fact that the imitation losses are normalized by the number of training examples, while the online losses are not.

The definition of π_{mix}. The policy π_{mix} executes $\pi_{mix}^{k,j}$ where k is first picked from $0 : (n-1)$ with probability proportional to $k+1$, and then j is picked form $1 : T$ uniformly. These proportions ensure that the states in each block of Table 5.2 have equal probability in the distribution of states generated by π_{mix}. The exact probability of picking a policy $\pi^{k,j}$ equals $\frac{2(k+1)}{T \cdot n(n+1)}$.

5.2.4 The bound on the imitation loss

Let ϵ_{class} denote the minimum per-state classifier loss on the states visited by $\pi^{1:N}$:

$$\begin{aligned}\epsilon_{class} := &\min_{\pi} \frac{1}{\sum_{i=1}^{N} |\mathcal{S}^i|} \sum_{k=0}^{n-1} \sum_{j=1}^{T} \sum_{S \in \mathcal{S}^{Tk+j}} l(S, \pi, \pi^*(S)) \\ = &\min_{\pi} \frac{2}{N \cdot T \cdot (n+1)} \sum_{k=0}^{n-1} \sum_{j=1}^{T} L_{onl}^{Tk+j}(\pi)\end{aligned} \tag{5.3}$$

The factors $\frac{1}{N}$, $\frac{2}{n+1}$, and $\frac{1}{T}$ account for the number of policies, the average number of training examples seen by any policy, and the number of states per training example respectively.

Let ϵ_{regret} denote the average regret of the sequence of policies $\pi^{1:N}$:

$$\epsilon_{regret} := \frac{2}{N \cdot T \cdot (n+1)} \sum_{k=0}^{n-1} \sum_{j=1}^{T} L_{onl}^{Tk+j}(\pi^{Tk+j}) - \epsilon_{class}. \tag{5.4}$$

When Follow-The-Regularized-Leader with convex losses and strongly convex regularizer is used to produce the policies π^i, the average regret ϵ_{regret} is $O(\frac{1}{\sqrt{N}})$ (provided the appropriate choice of the regularization constants).

Theorem 2 (Convergence of the *online* DAgger). *For a convex loss function l, a strongly convex regularizer R, and any $\epsilon > 0$, after $N = O(T^2/\epsilon^2)$ iterations of the* online *DAgger,*

$$L_{imit}[l](\pi_{mix}[\pi^{1:N}]) = T \cdot \epsilon_{class} + O(\epsilon).$$

Proof. Without the loss of generality, let $N = T \cdot n$.

$$\begin{aligned}L_{imit}[l](\pi_{mix}[\pi^{1:N}]) &= \sum_{k=0}^{n-1} \sum_{j=1}^{T} \frac{2(k+1)}{T \cdot n(n+1)} L_{imit}[l](\pi^{k,j}) \\ &= \frac{2}{T \cdot n(n+1)} \sum_{k=0}^{n-1} (k+1) \sum_{j=1}^{T} L_{imit}[l](\pi^{k,j}) \\ &= \frac{2}{T \cdot n(n+1)} \sum_{k=0}^{n-1} \sum_{j=1}^{T} L_{onl}[l](\pi^{Tk+j}) \\ &= T \cdot (\epsilon_{class} + \epsilon_{regret}).\end{aligned} \tag{5.5}$$

The first equation follows from the definition of π_{mix}. The third equation follows from Lemma 1. The statement of the theorem follows from ϵ_{regret} being $O\left(\frac{1}{\sqrt{N}}\right)$ due to the no-regret properties of Follow-The-Regularized-Leader. □

5.2.5 Implications and limitations

Theorem 2 establishes the convergence of the mixture policy π_{mix} in case when we make one iteration per collecting one ground truth state from the expert. The condition for the number of iterations N to be $O(T^2/\epsilon^2)$ in this case corresponds to the number of processed training examples n being $O(T/\epsilon^2)$. In other words, the number of training examples the *online* DAgger needs to converge is linear in the size of a single example.

Similarly to the proofs of the *offline* DAgger, the guarantee for the mixture policy π_{mix} immediately implies the guarantee for the best policy $\pi^{k,j}_{mix}$. This follows from the fact that the minimum is always less or equal to the weighted average, when the weights are positive and sum to one. As the policies $\pi^{k,j}_{mix}$ tend to π^{Tk+j} as k increases, the guarantee for the best $\pi^{k,j}_{mix}$ *should* in turn imply the guarantee for the best π^i, however we do not have a formal proof for this claim. We leave the investigation of this question for the future work. Another issue left for the future work is extending the results of Theorem 2 to the case when multiple iterations are performed between collecting the ground truth states from the expert.

5.3 Extracting domain-specific categories from Wikipedia

In Section 5.2 we proposed the *online* DAgger algorithm, which collects the ground truth labels for sequential prediction one "action" at a time. Equipped with the interface that suggests actions to the annotator, and learning from the actions in the online fashion, the algorithm reduces the labeling effort by making better and better suggestions over time. At first the annotator may need to evaluate all the possible actions for a given state in order to provide the correct action, but as the suggestions improve, the correct action will be suggested to the annotator ever sooner.

In this section we look into a slightly different setting of sequential prediction, in which

- the size of the output (the number of states in the trajectory) is too big, so the annotator cannot possibly provide the complete sequence (ground truth output);
- the number of possible actions in each state is too large to evaluate, so only the approximate feedback can be provided by the annotator;
- it is possible to query for actions individually, without regard to the context, and the loss decomposes over the structure of the output.

This is the setting of extracting the domain ontologies from the category network of

Wikipedia, which we described in Chapter 4. Specifically, we focus on the task of selecting the set of categories relevant for a given domain (Section 4.3.1). We developed a learning-to-search algorithm, which we call *active DAgger*, which interactively queries the user about the partial labels (individual topics), in order to speed up the learning process.

In the following sections we first re-formulate the problem of selecting the relevant categories as sequential prediction problem, and then present the *active DAgger* algorithm and the initial analysis thereof.

5.3.1 Selecting the relevant topics as sequential prediction.

Summary of the algorithm of Section 4.3.1. The algorithm we described in Section 4.3.1 traverses the category network in the breadth-first manner, starting from the root category (the one which defines the domain), and descending into the sub-categories. At each step, the algorithm decides whether the currently examined category is relevant and should be traversed further. The output is a connected graph of categories, in which every node can be reached from the root. The decision about including specific categories is made by a classifier, which is trained beforehand on a sample of topics.

Effectively, the algorithm of Section 4.3.1 sequentially builds a structured output, by moving in the space of graphs—from the graph containing only the root, to the complete selected graph. Transitions between the states (graphs) is guided by the classifier that decides whether a certain category should be included. Overall, the procedure can be viewed as an ad hoc sequential prediction algorithm. We will now formalize the problem as an instance of sequential structured prediction, and present a more principled algorithm.

Formalization of the problem. Consider the Wikipedia category network $\mathbb{G}$. We assume that for a given query (domain) there exists a ground truth output $G^* \subseteq \mathbb{G}$, a subgraph of the whole Wikipedia category network, which contains all the categories relevant to the domain and no other categories. The judgment about which categories belong to the domain is subjective, so the graph G^* will differ across opinions of different experts (users). Furthermore, the exact graph G^* is unknown, as the user will never get to see the complete graph due to its size. However, we assume that for a given user the graph G^* exists. We assume that G^* has the following properties:

- G^* contains the root node v_0 which describes the domain,
- every node in G^* is a descendant of the root node wrt. to the sub-category relation.

Let $\mathcal{G}$ be the set of all graphs that satisfy these properties. As in standard structured prediction, we seek the estimate G of the graph G^* by maximizing a linear scoring function

$$G := \operatorname*{argmax}_{G' \in \mathcal{G}} F[\mathbf{w}](G') = \operatorname*{argmax}_{G' \in \mathcal{G}} \langle \mathbf{w}, \Psi(G') \rangle. \tag{5.6}$$

We introduce an abstract space $\mathcal{S}$, which will be used to sequentially build the prediction G. Let $S_0 \in \mathcal{S}$ be the *initial state*, and $Term\,(\mathcal{S})$—the set of *terminal states.*

The terminal states $Term\,(\mathcal{S})$ correspond to candidate solutions to equation (5.6). In the simple case $\mathcal{S}$ may consist of the solutions, and $Term\,(\mathcal{S})$ be equal to $\mathcal{S}$. In a more general case it is only required that a candidate solution could be constructed from any terminal state by a mapping f:

$$f : Term\,(\mathcal{S}) \to \mathcal{G}.$$

For example, $\mathcal{S}$ may be the states of the beam search algorithm on the space of graphs $\mathcal{G}$.

We use linear classifiers $\pi[\mathbf{w}] : \mathcal{S} \to (\mathcal{S} \to \mathcal{S})$ to transition between states. In order to construct the solution, we recursively apply π to S_0, producing a sequence of states

$$S_0, S_1, \ldots, S_T,$$

where $S_{t+1} = a_{t+1}(S_t),\ a_{t+1} = \pi[\mathbf{w}](S_t)$ for $t = 0, 1, \ldots, T-1$. We only consider policies that terminate after a finite number of steps T and output a terminal state $S_T \in Term\,(\mathcal{S})$. The solution G is then obtained as

$$G = f\,(S_T)\,.$$

For simplicity, we focus on the greedy best-first search formulation, in which the sequence of states directly corresponds to the sequence of graphs:

$$(S_0, S_1, \ldots, S_T) = (G_0, G_1, \ldots, G_T)\,.$$

In this case, the initial graph-state G_0 consists of only the root node v_0. Every next state G_{t+1} in the sequence is obtained from the previous state G_t by adding one single node v_{t+1}, such that v_{t+1} has at least one *parent* node in G_t:

$$G_{t+1} = G_t \cup \{v_{t+1}\}.$$

It is also clear in this case that there is a one-to-one correspondence between the sequences of graphs and the sequence of nodes:

$$(G_0, G_1, \ldots, G_T) \Leftrightarrow (v_0, v_1, \ldots, v_T)\,.$$

To account for the terminal states, we introduce the empty node $v^{\perp}$. By choosing $v_{t+1} = v^{\perp}$ the policy terminates returning the current graph G_t.

The policy $\pi[\mathbf{w}]$ acts by greedily maximizing the scoring function in equation 5.6. It selects the node v_{t+1} from all possible nodes including $v^{\perp}$ that maximally increases the score of the current graph. Given the input state G_t, the policy $\pi[\mathbf{w}]$ evaluates its score $F[\mathbf{w}](G_t) = \langle \mathbf{w}, \Psi(G_t) \rangle$. It then selects the **highest-scoring next state**

$$v_{t+1} = \pi[\mathbf{w}](G_t) := \operatorname*{argmax}_{v \in \mathbb{G} \cup \{v^{\perp}\}} F[\mathbf{w}](G_t \cup \{v\})\}. \tag{5.7}$$

The specific case when $\pi[\mathbf{w}](G_t) \equiv v^{\perp}$ means that the graph G_t achieved the local maximum of the score F. This behavior of the policy $\pi[\mathbf{w}]$ can be generalized to the non-greedy case, such as the beam search, in which a number of best-scoring graphs are maintained in a truncated priority queue.

In order to measure the quality of the predicted solution on the training set, we use the target loss function:

$$L_{target}(G) = \frac{|G \triangle G^*|}{|G|} = \frac{|G \setminus G^*| + |G^* \setminus G|}{|G|} \tag{5.8}$$

$L_{target}(G)$ is the Jaccard distance between the sets of nodes in G and G^*.

We will use a sequential prediction algorithm, a modification of DAgger, to train the policy $\pi[\mathbf{w}]$ (find the vector $\mathbf{w}$). Similarly to what we described in Section 2.6.2, in order to apply DAgger to structured outputs, we need to replace the target loss function L_{target} defined on the complete outputs G with the imitation loss L_{imit} defined on the trajectories $v_0, v_1, \dots v_T$. Consider the imitation loss function that measures the number of mistakes made by the policy:

$$L_{imit}(\pi) := \sum_{t=0}^{T-1} l_{mistake}(G_t, \pi), \ \textit{where } G_t \sim \pi. \tag{5.9}$$

The mistakes can be of two kinds: 1) including an irrelevant category v_{t+1} or 2) terminating too early, when more relevant categories could have been included:

$$l_{mistake}(G_t, \pi) := \mathbb{1}(\pi(G_t) \in \mathbb{G} \setminus G^* \vee (\pi(G_t) = v^{\perp} \wedge G^* \setminus G_t \neq \varnothing)). \tag{5.10}$$

Proposition 3 (Connection between the target and the imitation losses)**.** *Let the graph G be produced by the policy π. With definitions of the target loss L_{target} (5.8) and the imitation loss L_{imit} (5.9):*

$$L_{target}(G) \leq \frac{2}{T} * L_{imit}(\pi) + \frac{|G^*| - |G|}{|G|}. \tag{5.11}$$

Proof. Let $l_{early\_stop} := \mathbb{1}(G^* \setminus G \neq \varnothing)$. First, we note that $|G| = T$, where T is the number of steps taken by the policy π. Second, we can rewrite the imitation loss (5.9):

$$L_{imit}(\pi) = \sum_{t=1}^{T-1} \mathbb{1}(v_t \notin G^*) + l_{early\_stop} = |G \setminus G^*| + l_{early\_stop}.$$

$$\begin{aligned}
L_{target}(G) &= \frac{1}{|G|}\left(|G \triangle G^*|\right) \\
&= \frac{1}{|G|}\left(|G| + |G^*| - 2|G \cap G^*|\right) \\
&= \frac{1}{|G|}\left(|G| + |G^*| - 2\left(|G| - |G \setminus G^*|\right)\right) \\
&= \frac{1}{|G|}\left(2|G \setminus G^*| + |G^*| - |G|\right) \\
&= \frac{2}{T} * \left(L_{imit}(\pi) - l_{early\_stop}\right) + \frac{|G^*| - |G|}{|G|} \\
&\leq \frac{2}{T} * L_{imit}(\pi) + \frac{|G^*| - |G|}{|G|}.
\end{aligned} \tag{5.12}$$

□

It follows from Proposition 3 that in order to bound the target loss function it is sufficient to 1) bound the imitation loss, 2) ensure that G is similar in size to G^*.

The problem of the large number of states and actions. As we know from Section 2.6, DAgger reduces the imitation loss in iterations, in which every next policy seeks to minimize the loss on the states produced by all the previous policies. The main complication in applying DAgger in the settings described previously in this section is that we cannot afford computing the loss in all states exactly—as this would require asking the user/expert about decisions taken by the policy in every state (the number of states being in thousands). We can reasonably query the user about only a handful of states, and approximate the imitation loss L_{imit} by computing $l_{mistake}$ in these states. Furthermore, even for a single state G_t the loss $l_{mistake}(G_t, \pi)$ cannot be computed exactly for every policy π, as is required by the DAgger framework. Specifying $l_{mistake}(G_t, \pi)$ exactly would mean querying the user about the relevance of every possible category v_{t+1} that could be added to G_t (with the number of candidates being again in thousands). As we can only query the user about a small number of categories in each iteration, we need to estimate the imitation loss L_{imit} from this very limited partial feedback. In the following section consider the implications of the partial feedback on the guarantees of DAgger.

5.4 DAgger with partial feedback.

Consider the execution of DAgger, in which at each iteration i we collect only partial information about the states visited by π^i and the loss in these states. Let $L^i_{class}(\pi)$ be the approximate per-state loss of the classifier π in the states visited by π^i:

$$\hat{L}^i_{class}(\pi) :\approx \frac{1}{T} \sum_{S \sim \pi^i} l(S, \pi, \pi^*(S)).$$

Suppose that we provide $\hat{L}^i_{class}$ to the DAGGER's online learner at iteration i (see Section 2.6.1):

$$L^i_{onl}(\pi) := \hat{L}^i_{class}(\pi).$$

An example of $\hat{L}^i_{class}$ could be the average loss l in a *sample* of states, drawn from the states visited by π^i. The i_{th} iteration of the data-aggregating implementation of DAGGER for this specific example would amount to generating (or collecting) the expert's actions for the states in the sample and adding the state-action pairs to the dataset. In general, if the exact loss function L^i represents the exact information about the cost (loss) associated with taking various actions in the states produced by π^i, the approximate loss represents some **partial *feedback*** about these actions. Even the partial feedback $\hat{L}^i(\pi^i)$ provides some information regarding how policy π^i can be improved. Under the convexity assumptions of Theorem 1, the guarantees of DAGGER will hold with respect to the approximate losses $\hat{L}^i_{class}$. Specifically, the following decomposition will be true:

$$\frac{1}{N}\sum_{i=1}^{N}\hat{L}_{imit}(\pi^i) = T\cdot(\epsilon_{class}+\epsilon_{regret}), \tag{5.13}$$

where $\hat{L}_{imit}(\pi^i) := T\cdot\hat{L}^i_{class}(\pi^i)$, and $\epsilon_{class} = \min_{\pi}\frac{1}{N\cdot T}\sum_{i=1}^{N}\hat{L}^i_{class}(\pi)$ is the minimum attainable per-state classifier loss on the states visited by $\pi^{1:N}$, and $\epsilon_{regret} = O(\frac{1}{\sqrt{N}})$.

In order to obtain similar guarantees for the exact loss L_{imit} we introduce the notion of α-informative loss (analogous to the α-informative feedback in [92]).

Definition 10. *The sequence of the approximate loss values $\{\hat{L}^i(\pi^i)\}_{i=1}^N$ is α -informative with respect to the sequence of exact loss values $\{L^i(\pi^i)\}_{i=1}^N$, some $\alpha\in(0,1]$ and the sequence $\{\xi^i\}_{i=1}^N$ if:*

$$\sum_{i=1}^{N}\left(\hat{L}^i(\pi^i)-\alpha\cdot L^i(\pi^i)+\xi^i\right)\geq 0. \tag{5.14}$$

With the loss functions viewed as *feedback*, α-informative loss means that feedback, on average, provides no less than the fraction α of the information about the true loss. The terms ξ^i can account for errors and noise in the feedback. With the α-informative approximate loss functions it is possible to formulate the following

Theorem 3 (Convergence of DAGGER with partial feedback)**.** *For the convex loss functions $\hat{L}^i_{imit}$, such that the sequence $\{\hat{L}^i_{imit}(\pi^i)\}_{i=1}^N$ is α-informative with respect to the sequences $\{L_{imit}(\pi^i)\}_{i=1}^N$ and $\{\xi^i\}_{i=1}^N$, a strongly convex regularizer R, and any $\epsilon>0$, after $N = O\left(\frac{T^2}{\epsilon^2\cdot\alpha^2}\right)$ iterations of DAGGER with approximate loss functions, there exists* *a policy $\pi^j \in \pi^{1:N}$, such that*

$$L_{imit}(\pi^j) \le \frac{T}{\alpha} \cdot \epsilon_{class} + \frac{1}{\alpha}\sum_{i=1}^{N} \xi^i + O(\epsilon).$$

Proof. The proof follows from the definition of the α-informative loss functions (5.14), and the equation (5.13). □

It follows from Theorem 3 that in order for the policies of DAGGER to improve it is sufficient to provide consistent feedback about their loss.

As we saw in the previous section, eliciting the exact loss from the user (expert) can be infeasible or impractical due to 1) the large number of states in a trajectory, 2) the large number of actions possible in each state. In the following sections we will examine the ways of reducing the number of actions and the number of states about which we query the user, while maintaining the α-informativeness of the approximate loss. We assume that we are dealing with the problem of selecting the relevant categories from Wikipedia described in Section 5.3.1. One property of the problem that we will rely on is that the user can be queried about individual actions (categories v_{t+1}) independently of the context, and provide binary "yes/no" feedback on the relevance of the categories.

5.4.1 Reducing the number of actions

First, let us consider approximations to the loss $l_{mistake}(G_t, \pi)$ (defined in (5.10)) in a single state G_t of the trajectory. Recall that the action taken by the current policy $v_{t+1} = \pi[\mathbf{w}]$ maximizes the $F[\mathbf{w}](G_t \cup \{v\})$ (equation (5.7)). There are two possibilities for v_{t+1}: either it is a node of the graph $\mathbb{G}$ or the terminating node $v^{\perp}$. If $v_{t+1} \in \mathbb{G}$, a single binary query to the user is sufficient to determine if the action v_{t+1} was a mistake. Specifically, we can query whether v_{t+1} is relevant ($v_{t+1} \in G^*$). Let y_{t+1} denote the user's feedback about the category v_{t+1}:

$$y_{t+1} = \begin{cases} 1, & \text{if } v_{t+1} \in G^*, \\ -1, & \text{if } v_{t+1} \in \mathbb{G} \setminus G^*. \end{cases} \tag{5.15}$$

The user's feedback can be encoded as the hinge loss:

$$\begin{aligned} \hat{l}(G_t, \pi[\mathbf{w}]) &:= l_h((G_t \cup \{v_{t+1}\}, G_t), \pi[\mathbf{w}], y_{t+1}) \\ &:= \max(0, 1 - y_{t+1} \cdot \langle \mathbf{w}, \Psi(G_t \cup \{v_{t+1}\}) - \Psi(G_t) \rangle). \end{aligned} \tag{5.16}$$

The hinge loss specifies that adding a relevant node to the graph should increase the score (by a margin), while adding an irrelevant one should decrease it. We can also show that the hinge loss defined in this way is an upper bound for $l_{mistake}$:

$$l_h((G_t \cup \{v_{t+1}\}, G_t), \pi[\mathbf{w}], y_{t+1}) \ge l_{mistake}(G_t, \pi[\mathbf{w}]).$$

In case when $v_{t+1} = v^{\perp}$, one query does not suffice to determine the presence or absence of mistake. In general, we need to query the user about the candidate nodes v' that could be added to the graph G_t—whether $v' \in G^*$ or not. If $v' \in G^*$, than the policy made a mistake by terminating instead of including v'. This information can be encoded by the hinge loss, requiring that $G_t \cup \{v'\}$ should score higher than G_t:

$$\begin{aligned}\hat{l}(G_t, \pi[\mathbf{w}]) &:= l_h((G_t \cup \{v'\}, G_t), \pi[\mathbf{w}], 1) \\ &= \max(0, 1 - \cdot\langle \mathbf{w}, \Psi(G_t \cup \{v'\}) - \Psi(G_t)\rangle).\end{aligned} \tag{5.17}$$

If we sample the candidate actions v' independently and uniformly, about $\log \frac{1}{\delta} \big/ \log \frac{1}{1-\varepsilon}$ queries are required to ensure that the fraction of relevant nodes among candidates is less that ε with confidence $1 - \delta$.

The binary query on the action v_{t+1} taken by current policy represents the minimum useful information that can be asked of the user. As an extension, we could request binary feedback about sets of nodes.

5.4.2 Reducing the number of states

In order to estimate the loss of the policy on a trajectory, it is sufficient to query the user about only a sample of states. We will review different ways to construct such a sample.

Uniform sampling. The idea of sampling the states from the trajectories is described in the original works on DAgger [84, 87]. At each iteration, an independent sample of m states can be collected to compute the estimate $\hat{L}_{imit}(\pi^i)$ of the loss $L^i_{imit}(\pi^i)$. As follows from the guarantees presented in that paper, with probability $1 - \delta$ the sequence of approximate losses is α-informative with respect to the true losses, with $\alpha = 1$ and

$$\sum_{i=1}^{N} \xi^i \leq \sqrt{\frac{\log \frac{1}{\delta}}{m \cdot N}}.$$

This guarantee alone is not sufficient for our problem, as it requires the number of iterations to be $O(T^2)$. With truly independent samples the number of queries to the user will also have to be $O(T^2)$, which is impractical. With the length T of the trajectory being in thousands, we can only afford a number of queries to be $O(\log(T))$, or at maximum $O(\sqrt{T})$. In practice we can relax the requirement of independence, and reuse the previously collected samples in order to avoid querying the user at most iterations.

Sampling until the first mistake. One way to reduce the number of samples is to stop sampling as soon as the first mistake made by the policy is discovered. Suppose that the states are sampled independently and uniformly, and the first encountered mistake is discovered on state number $m_{mistake}$. Suppose that this mistake corresponds to the action the policy π^i took in state G_t. If T is the length of the trajectory, then $\frac{T}{m_{mistake}}$ is an unbiased estimate of $L_{imit}(\pi^i)$. Therefore, the approximate loss functions $\hat{L}_{imit}(\pi^i) := \frac{T}{m_{mistake}}\hat{l}(G_t, \pi^i)$ are α-informative with $\alpha = 1$. The noise term $\sum_{i=1}^{N} \xi^i$ is sill likely to have the same bound as in case of collecting the complete sample.

Active sampling: latest states. Rather than sampling the states at random, one can define query selection procedures in the spirit of active learning, which may be more likely to discover mistakes. One of the procedures that we propose is based on the hypothesis that the policies on average are more likely to err at the end of the trajectory. The hypothesis is motivated by 1) the general fact that the errors accumulate in sequential prediction, and 2) by the shape of the Wikipedia category graph, in which the categories tend to become less relevant at distance from the root. According to this hypothesis, it makes more sense to first query for the last state on the trajectory of the current policy. The rule of selecting the last state, as well as the other described rules, can be used in conjunction with (e.g. prior to) random sampling.

Active sampling: smallest margin. Many ways to actively select samples have been described in the literature on active learning [90]. One of the main computationally efficient principles used in active learning is to query the user about the least confident decisions of the classifier. In structured prediction, the confidence have been measured through the *margin* of the classifier. Given the scoring function as in (5.6), the margin can be defined as the difference in score between the first and the second highest-scoring solutions. Relevant to our settings is the idea of the smallest margin in structured prediction when the partial labels can be queried independently ([93]). In our settings, in which the decisions are made with respect to the current state G_t, a natural measure of confidence of a decision v_{t+1} is the score difference $\langle \mathbf{w}, \Psi(G_t \cup \{v_{t+1}\}) - \Psi(G_t)\rangle$. We suggest querying the user about a number of the least confident decisions of the current policy. A similar approach was described by Culotta and McCalum [20] for conditional random fields (CRFs).

Reusing the collected feedback is the strongest tool for reducing the number of queries to the user. Recall that in our setting the partial label for an action (category) v_{t+1} does not depend on the state G_t (graph built so far). Once provided by the user, the partial label for v_{t+1} can be reused to generate feedback for multiple different states G_t. Suppose that by iteration i we have collected a number of positive (relevant) nodes V_+ and a number of negative nodes V_-. Nodes from V_- can be used to generate negative feedback: if $v_{t+1} \in V_-$ we require that $F(G_t) > F(G_t \cup \{v_{t+1}\})$. Similarly, nodes from V_+ can produce positive feedback: if $v_{t+1} = v^{\perp}$ and $\exists v' \in V_+ \setminus G_t$, we specify that $F(G_t) < F(G_t \cup \{v'\})$[2].

Suppose that for a given trajectory we have detected the set of states $\mathcal{G}_{mistake}$, in which

[2]In the last example we assumed that v' is a candidate node (is a sub-category of some node in G_t); if this is not true, we can compute v''—a node on the shortest path from v' to the root, such that v'' has a parent category in G_t

the current policy made a mistake, according to V_- and V_+. Define the approximate loss $\hat{L}(\pi^i) := \frac{T}{|V_-| + |V_+|} \sum_{G_t \in \mathcal{G}_{mistake}} \hat{l}(G_t, \pi^i)$. We hypothesize that in practice the approximate losses $\hat{L}(\pi^i)$ will be α-informative with $\alpha = 1$. Additional benefit of reusing the collected feedback is that it provides for the early detection of mistakes, so we often can proceed to the next iteration without generating the complete trajectory.

Controlling the early stopping. As we have shown in Proposition 3, in order to achieve low target loss, we need not only to bound the imitation loss, but also make sure the graph is sufficiently large. In other words, we need to avoid early termination of the trajectory. The procedure that we proposed for this relies on the collected set V_+ of the nodes (categories) that we know should be present in the final graph. When $V_+ \setminus G_T$ is not empty, we can conclude without querying the user that the policy terminated too early. When $V_+ \subseteq G_T$, we need to sample categories from the set of categories that could be added to G_T. Different sampling strategies can be used—from querying the highest-scoring nodes first to uniform sampling. When no relevant categories not belonging to the current solution G_T have been discovered through sampling, a statistical test can be used to conclude that the percentage of undetected relevant categories is within ε with probability at least $1 - \delta$. With uniform sampling, the number of required samples is roughly $\log \frac{1}{\delta} \big/ \log \frac{1}{1-\varepsilon}$. In order to simplify the procedure and reduce the number of queries we suggest sacrificing statistical soundness and estimate the number of mistakes using all previously collected feedback.

5.4.3 Active DAgger with partial feedback

We informally summarize the presented ideas on reducing the labeling effort for the task of selecting the relevant categories from Wikipedia in Algorithm 4. The algorithm differs from the standard DAgger in the following ways:

- Similarly to *online DAgger* (Section 5.2), the algorithm trains the policies *interactively*, interleaving the iterations of training with querying the user for feedback.
- The feedback that is asked of the user is *partial* in two ways, namely because the user is queried about
  - *a single state* on the trajectory generated by the current policy;
  - *a single action* in that state, namely the one taken by the current policy.
- The algorithm *actively* selects the queries about the most uncertain decision of the current policy.
- The feedback collected throughout iterations is used to judge the confidence of the algorithm about its own loss, and in particular about the termination condition.

Discussion. The analysis we presented in the previous sections only partially describes Algorithm 4. Furthermore, we have only started to experiment with the implementation.

Algorithm 4: Active DAgger with partial feedback

```
1  $D \leftarrow \varnothing$ // empty set
2  $V_-, V_+ \leftarrow \varnothing$
3  $\pi^0 \leftarrow$ random policy (or any reasonable policy)
4  for $i \in 0 : N-1$ do
5      Produce the trajectory $(G_1^i, \ldots, G_{T^i}^i) \sim \pi^i$
6      Estimate the target loss $\hat{L}^i \approx L_{target}(\pi^i)$ through $V_-, V_+$
7      if $\hat{L}^i$ is low and the confidence is high then
8          return $G_{T^i}^i$
9      end
10     if $\hat{L}^i$ is low and confidence is low then
11         Query the user about the least confident decisions of $\pi^i$
12         Update $V_-, V_+$ with user answers
13     end
14     Update $D$ with generated feedback
15     Train $\pi^{i+1}$ on the updated $D$
16 end
17 return $G_{T^i}^i$
```

Therefore, at the moment we only speculate about the properties of the algorithm, such as the expected convergence and the number of queries made to the user.

Querying the user actively about the least confident decisions of the policy *should* increase the probability of discovering a policy's mistake, with respect to querying about random states. The higher probability of detecting mistakes *should* keep the loss estimated via the user feedback α-informative with respect to the true loss, ensuring the convergence of the algorithm according to Theorem 3. The feedback collected throughout iterations *should* allow estimating the loss at most iterations without querying the user, making the total number of queries much less than the guaranteed number of iterations ($O(T^2)$). We hope for the number of queries to be $O(\sqrt{T})$.

Deeper theoretical analysis and experimental validation are needed to verify these claims, which we leave for the nearest future work.

Chapter 6

Conclusion

Navigating and searching through scientific publications is a difficult task, which requires experience and skill to be done efficiently. We believe that current academic search interfaces with keyword-based search and attribute-based navigation are still in their infancy. With access to publication metadata, keywords, citations, and the background knowledge, there is a great potential of improving the support these services could provide in a variety of the academic search tasks. With this thesis we advocate building structured visual representations that summarize the academic search results at the semantic level, and contribute a number of ideas and methods pertaining to the problem.

6.1 Representing the academic search results with topic maps

In Chapter 3 we presented the idea of topic maps—a structured representation of the academic search results in terms of the relevant topics. A topic map shows the most relevant topics mentioned in the query results, along with the subsumption relations between the topics. We proposed a novel method for building the topic maps based on the network of articles and categories of Wikipedia. Due to the properties of Wikipedia, the topics in the map have concise meaningful titles, cover a broad range of subjects in sufficient detail, are collaboratively maintained, and are reasonably up to date. To the best of our knowledge, no other method of representing documents possesses this combination of merits (see Section 2.2).

The sub-topic relations introduce some useful redundancy in the representation of documents: if a document belongs to a topic, it also belongs to all its parent topics in the map. This redundancy provides for a way of summarizing the topic maps by selecting a subset of the topics. We proposed a method for summarizing topic maps to a given size, which learns how to build *the most informative summaries* from examples. The method of summarizing topic maps presents a novel application of a state-of-the-art structured prediction framework DAGGER. We implemented the idea of topics maps in a prototype academic search interface ScienScan, deployed on top of a third-party academic search tool. The idea of topic maps, and the methods for building and summarizing them are the main contributions of this thesis.

Future work

In our work we provided mostly theoretical argument for using topic maps, and performed a limited offline evaluation of the topic summarization method. In the future work we would like to validate the idea of topic maps from the point of view of human-computer interaction. A controlled user study should be performed in order to assess the benefits of topic maps from the user perspective. Although the methods we proposed in this thesis are independent of the scientific domain, in the evaluation and the development of the prototype tool we mostly focused on the field of computer science, for clear reasons. In the future we would like to evaluate the topic maps with other domains as well.

With this thesis we made only a few steps towards structured representation of the academic search results. A lot more can be done in this direction. One further step that we find promising is summarizing the query results not only at the level of topics, but also at the level of individual papers. For instance, the structured summary could highlight the most significant or the most representative contributions in each topic. Chapters 4 and 5 represent two more directions of future work.

One direction is building **semantically reach topic maps**. We envision structured representations that distinguish different types of research papers, such as “survey”, “methodological paper”, “experimental evaluation”, different types of topics and domain concepts, and different types of relations between them, such as “subtopic-of”, “applied-to”, “part-of”. As we have seen in Chapter 4, the meaningful semantic relations between topics are difficult (or perhaps even impossible) to determine without background knowledge. Whether the necessary background knowledge can be found or created for such a large, complex and evolving domain as science remains to be seen.

Another interesting direction is creating the representations that can **learn from user behavior**. We hypothesize that the user’s interaction with a browsing controls, such as a topic map, may convey important information about the usefulness of this representation the user. We can imagine that the search tool could learn how to represent the query results, similarly to how the ranking of the search results can be improved based on the user’s clicking behavior [92]. A starting point in this research could be the labeling interface that learns from the annotator’s responses, which we presented in Chapter 5.

We would like to mention one more extension of the ideas presented in this thesis that we find particularly interesting. The topic maps we proposed possess some limited interaction capabilities: namely, the user can 1) filter the search results based on any of the displayed topics, and 2) control the granularity/size of the topic map with a slider. We argue that the search result representations could be made *fully interactive*. Rather than representing only the query results, the topic map could represent the entire underlying collection of papers, and allow the user to navigate through this collection by changing the focus. We envision that interactive structured and focused summaries could become the main tool of performing exploratory academic search.

6.2 Extracting domain ontologies from Wikipedia categories

In Chapter 4 we proposed a method for extracting the "skeletons" of the domain ontologies from the categories of Wikipedia. The motivating application for us was building semantically rich topic maps and domain-specific instances of the ScienScan tool. Besides topic maps, facilitating the creation of large-scale domain ontologies is an important problem in its own right. We proposed a method that relies on the supervised machine learning, namely binary classification, to first select the set of categories relevant to the domain, then classify them into classes and individuals, and, finally, classify the relations between them. This approach is uniform, flexible, and can easily incorporate new features. The flexibility comes at the cost of annotating training examples. The annotation has to be performed at most once for a given domain. Whether the method will generalize across domains should be evaluated more thoroughly. Our experiments suggested that the method performs reasonably well on selecting and classifying the topics even without retraining on a new domain. We plan to further investigate the questions of domain dependence, and perform the evaluation on more domains in the future work.

One property of the proposed method is that it relies only on the information present in the categories, namely the titles and the sub-category relations. Despite the limited information, the method has performed well in selecting the relevant categories, identifying the classes and individuals, and identifying the `subclass_of` relation. The performance on the other two relations (`instance_of` and `part_of`) suggest that these types are more difficult to capture. In particular, it is clear that the category titles often contain insufficient information, even for a human, to determine the `part_of` relation: consider, for instance, `part_of`(Social_media, World_Wide_Web) or `part_of`(Data_warehousing, Business_intelligence). In order to address this problem, the future work should investigate into additional features based on the category structure, and the possibility to use external data sources. Additionally, one can developed more advanced classification algorithms that allow for joint prediction of relations. The sequential prediction method we presented in Section 5.3 provides the basis for making such joint predictions. Another extension of the method could include Wikipedia articles in addition to the categories.

6.3 Reducing the labeling effort in search-based prediction

In Chapter 5 we preliminarily addressed a technical (from the point of view of the main goal of this thesis) problem: facilitating the provision of labels for the sequential prediction tasks. The motivation for this work was the difficulty, or even impossibility, of providing complete and accurate ground truth data for training the sequential prediction methods that we presented in Chapters 3 and 4. To this end, we developed two modifications of the state-of-the-art sequential prediction framework, DAgger, and performed initial analysis of their theoretical guarantees.

In the first scenario the annotator has to provide the accurate labels for the whole ground truth trajectory, from which the expert actions are then automatically generated. We developed a labeling interface that suggests the labels to the annotator, learning from every provided label in the online manner. To support the labeling interface, we proposed a modification of the DAgger algorithm that performs one iteration per single ground truth label, enabling the online learning of policies. The modified algorithm enjoys the regret bound similar to that of the original DAgger, which suggests that the labeling interface should make better recommendations over time, reducing the labeling effort.

In the second scenario the feedback provided by the user can neither be complete nor accurate due to the size of the problem. We investigated the guarantees of DAgger in case when the loss functions provided to the algorithm are approximate. It was shown that the α-informativeness of the approximate loss allows retaining the regret guarantees up to the factor α, similarly to the results of [92].

The results presented in Chapter 5 are preliminary, and require more extensive theoretical analysis and empirical evaluation.

Bibliography

[1] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[2] Marcos Baez, Daniil Mirylenka, and Cristhian Parra. Understanding and supporting search for scholarly knowledge. *Proceeding of the 7th European Computer Science Summit*, pages 1–8, 2011.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[4] David M. Blei, Thomas L. Griffiths, Michael I. Jordan, and Joshua B. Tenenbaum. Hierarchical Topic Models and the Nested Chinese Restaurant Process. In *Neural Information Processing Systems*, 2003.

[5] D.M. Blei, T.L. Griffiths, and M.I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[6] D.M. Blei and J.D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd international conference on Machine learning*, pages 113–120. ACM, 2006.

[7] D.M. Blei and J.D. Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35, 2007.

[8] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[9] L. Bolelli, Ş. Ertekin, and C. Giles. Topic and trend detection in text collections using latent dirichlet allocation. *Advances in Information Retrieval*, pages 776–780, 2009.

[10] Sergey Bratus, Anna Rumshisky, Alexy Khrabrov, Rajenda Magar, and Paul Thompson. Domain-specific entity extraction from noisy, unstructured data using ontology-guided search. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(2):201–211, 2011.

[11] Volha Bryl, Claudio Giuliano, Luciano Serafini, and Kateryna Tymoshenko. Supporting natural language processing with background knowledge: coreference resolution case. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 80–95, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] C. Calli, G. Ucoluk, and T. Sehitoglu. *Improving search result clustering by integrating semantic information from Wikipedia.* PhD thesis, MS Thesis, Middle East Technical University, Department of Computer Engineering, 2010.

[13] C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of web clustering engines. *ACM Computing Surveys (CSUR)*, 41(3):17, 2009.

[14] C. Carpineto and G. Romano. Exploiting the potential of concept lattices for information retrieval with credo. *Journal of universal computer science*, 10(8):985–1013, 2004.

[15] Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. Dexter 2.0 - an open source tool for semantically enriching data. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, pages 417–420, 2014.

[16] S. Chowdhury, F. Gibb, and M. Landoni. Uncertainty in information seeking and retrieval: A study in an academic environment. *Information Processing & Management*, 47(2):157–175, 2011.

[17] W.W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480. ACM, 2002.

[18] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[19] A. Csomai and R. Mihalcea. Linking documents to encyclopedic knowledge. *Intelligent Systems, IEEE*, 23(5):34–41, 2008.

[20] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, pages 746–751, 2005.

[21] D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of*

the 15th annual international ACM SIGIR conference on Research and development in information retrieval, pages 318–329. ACM, 1992.

[22] Hal Daumé III. *Practical Structured Learning Techniques for Natural Language Processing.* PhD thesis, University of Southern California, Los Angeles, CA, August 2006.

[23] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. 2009.

[24] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005.

[25] L. Dietz, S. Bickel, and T. Scheffer. Unsupervised prediction of citation influences. In *Proceedings of the 24th international conference on Machine learning*, pages 233–240. ACM, 2007.

[26] S.T. Dumais, G.W. Furnas, T.K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285. ACM, 1988.

[27] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

[28] Vitaly Feldman, Venkatesan Guruswami, Prasad Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. *SIAM Journal on Computing*, 41(6):1558–1590, 2012.

[29] Paolo Ferragina and Ugo Scaiella. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1625–1628. ACM, 2010.

[30] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the 22nd international conference on Machine learning*, pages 217–224. ACM, 2005.

[31] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem phong Vo. A technique for drawing directed graphs. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 19(3):214–230, 1993.

[32] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

[33] T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.

[34] X. Han and J. Zhao. Topic-driven web search result organization by leveraging wikipedia semantic knowledge. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1749–1752. ACM, 2010.

[35] He He, Jason Eisner, and Hal Daume. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2012.

[36] Q. He, B. Chen, J. Pei, B. Qiu, P. Mitra, and L. Giles. Detecting topic evolution in scientific literature: how can citations help? In *Proceeding of the 18th ACM conference on Information and knowledge management*, pages 957–966. ACM, 2009.

[37] M.A. Hearst and J.O. Pedersen. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 76–84. ACM, 1996.

[38] F. Heylighen. Change and information overload: Negative effects. *Principia Cybernetica Web*, 1999.

[39] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[40] Hal Daumé III and Daniel Marcu. A bayesian model for supervised clustering with the dirichlet process prior. *J. Mach. Learn. Res.*, 6:1551–1577, December 2005.

[41] Rizwan Iqbal, Masrah Azrifah Azmi Murad, Aida Mustapha, Sharef, and Nurfadhlina Mohd. An analysis of ontology engineering methodologies: A literature review. *Research Journal of Applied Sciences, Engineering and Technology*, 6(16):2993–3000, 2013.

[42] P. Jaccard. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901.

[43] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

[44] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, and Yujiao Zhou. Logmap 2.0: Towards logic-based, scalable and interactive ontology matching. SWAT4LS '11, pages 45–46, New York, NY, USA, 2012. ACM.

[45] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[46] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[47] N. Kawamae. Author interest topic model. In *SIGIR*, pages 887–888. ACM, 2010.

[48] C.C. Kuhlthau. Seeking meaning: A process approach to library and information services. 2004.

[49] Claudia Leacock, George A Miller, and Martin Chodorow. Using corpus statistics and wordnet relations for sense identification. *Computational Linguistics*, 24(1):147–165, 1998.

[50] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web Journal*, 2014.

[51] Michael Ley. The dblp computer science bibliography: Evolution, research issues, perspectives. In *String Processing and Information Retrieval*, pages 1–10. Springer, 2002.

[52] Michael Ley. Dblp: some lessons learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500, 2009.

[53] Michael Ley and Patrick Reuther. Maintaining an Online Bibliographical Database: the Problem of Data Quality., 2006.

[54] W. Li and A. McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584. ACM, 2006.

[55] Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

[56] Carolyn E Lipscomb. Medical subject headings (mesh). *Bulletin of the Medical Library Association*, 88(3):265, 2000.

[57] Qiaoling Liu, Kaifeng Xu, Lei Zhang, Haofen Wang, Yong Yu, and Yue Pan. Catriple: Extracting triples from Wikipedia categories. In *The Semantic Web*, Lecture Notes in Computer Science. 2008.

[58] Y.S. Maarek, R. Fagin, I.Z. Ben-Shaul, and D. Pelleg. Ephemeral document clustering for web applications. 2000.

[59] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[60] Jon D Mcauliffe and David M Blei. Supervised topic models. In *Advances in neural information processing systems*, pages 121–128, 2008.

[61] Gerry McKiernan. arxiv. org: the los alamos national laboratory e-print server. *International Journal on Grey Literature*, 1(3):127–138, 2000.

[62] Olena Medelyan and Ian H Witten. Domain-independent automatic keyphrase indexing with small training sets. *Journal of the American Society for Information Science and Technology*, 59(7):1026–1040, 2008.

[63] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

[64] Alistair Miles and Sean Bechhofer. Skos simple knowledge organization system reference. *W3C recommendation*, 18:W3C, 2009.

[65] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[66] D. Milne and I.H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.

[67] David Milne and Ian H. Witten. An open-source toolkit for mining wikipedia. *Artificial Intelligence*, 194(0):222 – 239, 2013.

[68] Daniil Mirylenka and Andrea Passerini. Learning to grow structured visual summaries for document collections. In *ICML Workshop on Structured Learning*, 2013.

[69] Daniil Mirylenka and Andrea Passerini. Navigating the topical structure of academic search results via the Wikipedia category network. In *CIKM'13*, pages 891–896. ACM, 2013.

[70] Daniil Mirylenka and Andrea Passerini. Scienscan—an efficient visualization and browsing tool for academic search. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip elezn, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8190 of *Lecture Notes in Computer Science*, pages 667–671. Springer Berlin Heidelberg, 2013.

[71] Daniil Mirylenka and Andrea Passerini. Supervised graph summarization for structuring academic search results. 2013.

[72] D. Nahl. A conceptual framework for explaining information behavior. *SIMILE: Studies in Media & Information Literacy Education*, 1(2):1–16, 2001.

[73] R.M. Nallapati, A. Ahmed, E.P. Xing, and W.W. Cohen. Joint latent topic models for text and citations. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 542–550. ACM, 2008.

[74] Vivi Nastase and Michael Strube. Transforming Wikipedia into a large scale multilingual concept network. *Artificial Intelligence*, 194(0):62 – 85, 2013. Artificial Intelligence, Wikipedia and Semi-Structured Resources.

[75] C.T. Nguyen, X.H. Phan, S. Horiguchi, T.T. Nguyen, and Q.T. Ha. Web search clustering and labeling with hidden topics. *ACM Transactions on Asian Language Information Processing (TALIP)*, 8(3):12, 2009.

[76] Francesco Osborne and Enrico Motta. Mining semantic relations between research areas. In *The Semantic Web–ISWC 2012*, pages 410–426. Springer, 2012.

[77] Francesco Osborne, Enrico Motta, and Paul Mulholland. Exploring scholarly data with Rexplore. In *The Semantic Web ISWC 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 460–477. 2013.

[78] S. Osiriski, J. Stefanowski, and D. Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining: proceedings of the International IIS: IIPWM04 Conference held in Zakopane, Poland*, page 359, 2004.

[79] Simone Paolo Ponzetto and Roberto Navigli. Large-scale taxonomy mapping for restructuring and integrating Wikipedia. IJCAI'09, pages 2083–2088, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[80] Simone Paolo Ponzetto and Michael Strube. Deriving a large scale taxonomy from Wikipedia. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1440–1445. AAAI Press, 2007.

[81] A. Popescul and L.H. Ungar. Automatic labeling of document clusters. *Unpublished manuscript, available at http://citeseer. nj. nec. com/popescul00automatic. html*, 2000.

[82] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

[83] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494. AUAI Press, 2004.

[84] Stephane Ross. *Interactive Learning for Sequential Decisions and Predictions.* PhD thesis, Carnegie Mellon University, 2013.

[85] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

[86] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[87] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research - Proceedings Track*, 15:627–635, 2011.

[88] C. Sacarea, R. Meza, and M. Cimpoi. Improving conceptual search results reorganization using term-concept mappings retrieved from wikipedia. In *Automation, Quality and Testing, Robotics, 2008. AQTR 2008. IEEE International Conference on*, volume 3, pages 234–238. IEEE, 2008.

[89] U. Scaiella, P. Ferragina, A. Marino, and M. Ciaramita. Topical clustering of search results. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 223–232. ACM, 2012.

[90] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[91] Shai Shalev-Shwartz and Sham M Kakade. Mind the duality gap: Logarithmic regret algorithms for online optimization. In *Advances in Neural Information Processing Systems*, pages 1457–1464, 2009.

[92] Pannagadatta K Shivaswamy and Thorsten Joachims. Online learning with preference feedback. *arXiv preprint arXiv:1111.0712*, 2011.

[93] Kevin Small and Dan Roth. Margin-based active learning for structured predictions. *International Journal of Machine Learning and Cybernetics*, 1(1-4):3–25, 2010.

[94] Steffen Staab and Rudi Studer. *Handbook on ontologies.* Springer, 2010.

[95] B. Stein and S.M. Zu Eissen. Topic identification: Framework and application. In *Proc. International Conference on Knowledge Management*, volume 400, pages 522–531, 2004.

[96] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. *The NeOn methodology for ontology engineering*, pages 9–34. Springer, 2012.

[97] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Large Ontology from Wikipedia and WordNet. *Elsevier Journal of Web Semantics*, 2008.

[98] J. Tang, R. Jin, and J. Zhang. A topic modeling approach and its integration into the random walk framework for academic search. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 1055–1060. IEEE, 2008.

[99] P. Treeratpituk and J. Callan. Automatically labeling hierarchical clusters. In *Proceedings of the 2006 international conference on Digital government research*, pages 167–176. Digital Government Society of North America, 2006.

[100] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[101] Vladimir Vapnik. *The nature of statistical learning theory.* springer, 2000.

[102] M. Wahabzada, Z. Xu, and K. Kersting. Topic models conditioned on relations. *Machine Learning and Knowledge Discovery in Databases*, pages 402–417, 2010.

[103] Jianwen Wang, Xiaohua Hu, Xinhui Tu, and Tingting He. Author-conference topic-connection model for academic network search. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2179–2183. ACM, 2012.

[104] D. Weiss. *Descriptive clustering as a method for exploring text collections.* PhD thesis, Citeseer, 2006.

[105] R.W. White and R.A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.

[106] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[107] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. *Advances in neural information processing systems*, 15:505–512, 2002.

[108] Zaihan Yang, Liangjie Hong, and Brian D Davison. Academic network analysis: A joint topic modeling approach. In *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 324–333. IEEE, 2013.

[109] O. Zamir and O. Etzioni. Grouper: a dynamic clustering interface to web search results. *Computer Networks*, 31(11):1361–1374, 1999.

[110] Ying Zhao, George Karypis, and Usama Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 2005. 10.1007/s10618-005-0361-3.

[111] Căcilia Zirn, Vivi Nastase, and Michael Strube. Distinguishing between instances and classes in the Wikipedia Taxonomy. ESWC'08, pages 376–387, Berlin, Heidelberg, 2008. Springer-Verlag.

Appendix A

Summary of the notation

- $f : \mathcal{X} \to \mathcal{Y}$ – a function f with domain $\mathcal{X}$ and range $\mathcal{Y}$
- $\cdot : \cdot$ – used to denote sequences, e.g.:
  - $1 : T$ – sequence of natural numbers $1, 2, \dots, T-1, T$
  - $y_{1:T}$ – sequence $y_1, y_2, \dots, y_{T-1}, y_T$
  - $\pi^{1:N}$ – sequence $\pi^1, \pi^2, \dots, \pi^{N-1}, \pi^N$
- $[\cdot]$ – means "parametrized by", e.g.:
  - $\pi[\mathbf{w}]$ – a classifier π parametrized by a weight vector $\mathbf{w}$
  - $\mathbf{w}[L, D]$ – a weight vector parametrized by a loss function L and a dataset D
- $\mathbb{1}$ – the indicator function:

$$\mathbb{1}(A) \equiv \begin{cases} 1, & \text{if } A \text{ is } true, \\ 0, & \text{if } A \text{ is } false \end{cases}$$

- $\mathfrak{S}_k$ – the group of permutations of k elements
- $\succ$ – an order relation
- $\mathbb{R}_+$ – the set of *nonnegative* real numbers
- $\varnothing$ – the empty set
- $\setminus$ – set difference

$$A \setminus B := \{a \mid a \in A \text{ and } a \notin B\}$$

- $\triangle$ – symmetric set difference

$$A \triangle B := (A \setminus B) \cup (B \setminus A)$$

- $\circ$ – function composition

$$(f \circ g)(x) := f(g(x))$$

- $\cdot^{+}$ – transitive closure, e.g.:
  - E^{+} – transitive closure of relation E
  - G^{+} – transitive closure of graph G
- π – a policy, a classifier, a prediction function
- $\langle \cdot \rangle$ – a scalar product, e.g.
  - $\langle \mathbf{w}, \Psi(X) \rangle$
- $\mathbf{w}$ – a weight vector
- $\Psi,\ \Phi,\ \phi$ – feature maps
- $L(X, \mathbf{w}, Y)$ – a loss function, e.g.:
  - L_{imit} – imitation loss
  - L_{onl} – online loss
  - L_{target} – target loss
  - L_{class} – classifier loss
  - $L_{0/1}$ – zero-one loss
  - L_{h} – hinge loss
- $\Delta(Y, Y')$ – a dissimilarity (distance) function, e.g.:
  - $\Delta_{0/1}$ – zero-one distance
- $l(X, \mathbf{w}, Y)$ – a local loss function
- $\hat{\cdot}$ – an approximate value or function, e.g.:
  - $\hat{L}$ – an approximate loss function
  - $\hat{l}$ – an approximate local loss function
  - $\hat{\pi}$ – an approximate policy
- $\cdot^{*}$ – the "ground truth" or exact value or function, e.g.:
  - Y^{*} – the ground truth output
  - $y^{*}_{1:T}$ – the ground truth sequence
  - G^{*} – the ground truth graph
  - π^{*} – the expert policy

Appendix B

Example of a large topic map

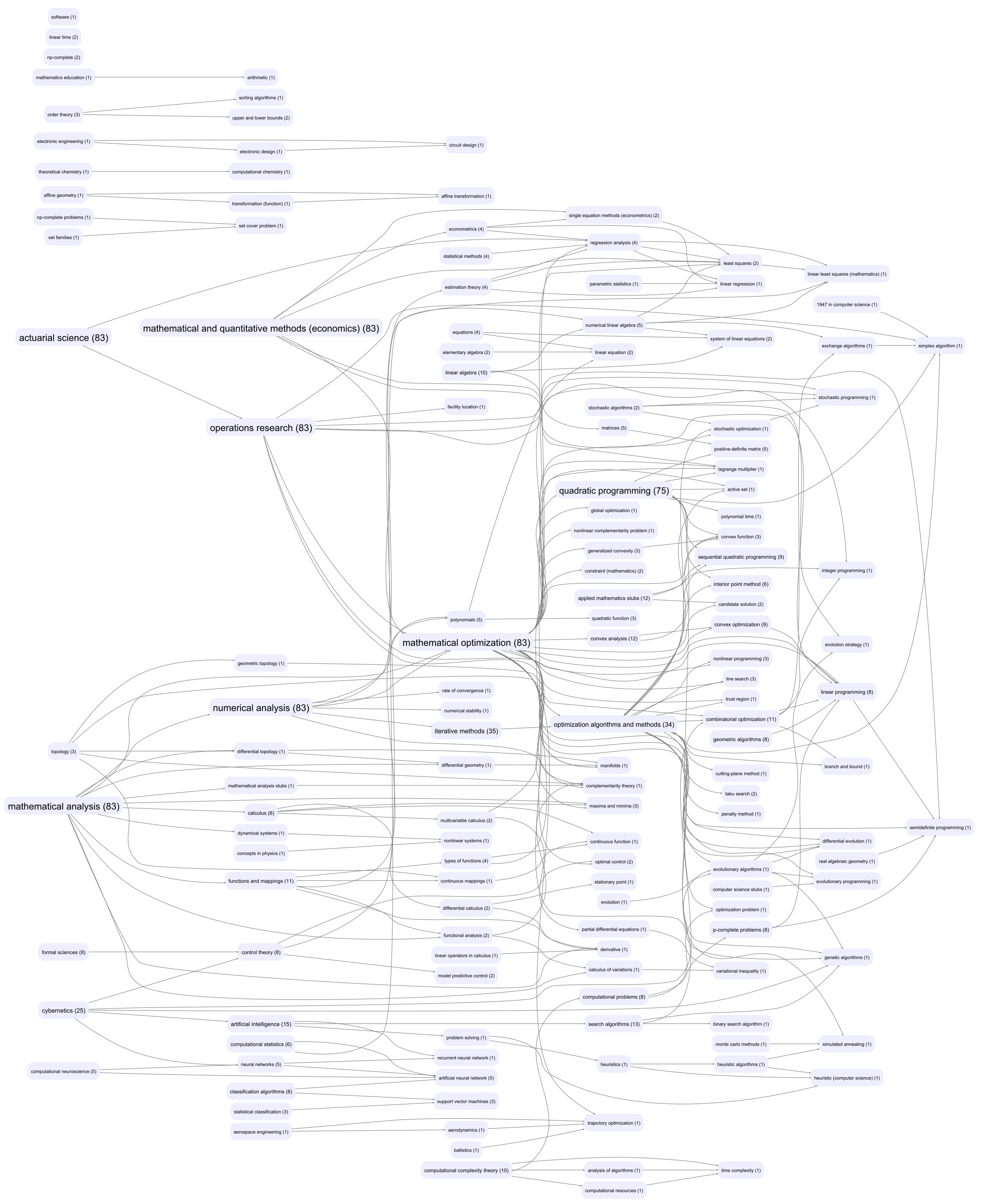

software (1)
linear time (2)
np-complete (2)
mathematics education (1)
arithmetic (1)
sorting algorithms (1)
order theory (3)
upper and lower bounds (2)
electronic engineering (1)
circuit design (1)
electronic design (1)
theoretical chemistry (1)
computational chemistry (1)
affine geometry (1)
affine transformation (1)
transformation (function) (1)
np-complete problems (1)
set cover problem (1)
set families (1)
single equation methods (econometrics) (2)
econometrics (4)
regression analysis (4)
statistical methods (4)
least squares (2)
linear least squares (mathematics) (1)
estimation theory (4)
parametric statistics (1)
linear regression (1)
1947 in computer science (1)
actuarial science (83)
mathematical and quantitative methods (economics) (83)
numerical linear algebra (5)
equations (4)
system of linear equations (2)
exchange algorithms (1)
simplex algorithm (1)
elementary algebra (2)
linear equation (2)
linear algebra (10)
stochastic programming (1)
facility location (1)
stochastic algorithms (2)
operations research (83)
matrices (5)
stochastic optimization (1)
positive-definite matrix (5)
lagrange multiplier (1)
quadratic programming (75)
active set (1)
global optimization (1)
polynomial time (1)
nonlinear complementarity problem (1)
convex function (3)
generalized convexity (3)
sequential quadratic programming (9)
constraint (mathematics) (2)
integer programming (1)
interior point method (6)
applied mathematics stubs (12)
candidate solution (2)
polynomials (5)
quadratic function (3)
convex optimization (9)
convex analysis (12)
mathematical optimization (83)
evolution strategy (1)
nonlinear programming (3)
geometric topology (1)
line search (3)
rate of convergence (1)
linear programming (8)
trust region (1)
numerical analysis (83)
numerical stability (1)
combinatorial optimization (11)
optimization algorithms and methods (34)
iterative methods (35)
geometric algorithms (8)
topology (3)
differential topology (1)
branch and bound (1)
differential geometry (1)
manifolds (1)
cutting-plane method (1)
mathematical analysis stubs (1)
complementarity theory (1)
tabu search (2)
mathematical analysis (83)
maxima and minima (3)
penalty method (1)
calculus (6)
multivariable calculus (2)
semidefinite programming (1)
dynamical systems (1)
nonlinear systems (1)
continuous function (1)
differential evolution (1)
concepts in physics (1)
types of functions (4)
optimal control (2)
real algebraic geometry (1)
evolutionary algorithms (1)
functions and mappings (11)
continuous mappings (1)
stationary point (1)
evolutionary programming (1)
computer science stubs (1)
evolution (1)
differential calculus (2)
optimization problem (1)
partial differential equations (1)
p-complete problems (8)
functional analysis (2)
formal sciences (8)
control theory (8)
derivative (1)
genetic algorithms (1)
linear operators in calculus (1)
model predictive control (2)
calculus of variations (1)
variational inequality (1)
computational problems (8)
cybernetics (25)
artificial intelligence (15)
search algorithms (13)
binary search algorithm (1)
problem solving (1)
computational statistics (6)
monte carlo methods (1)
simulated annealing (1)
recurrent neural network (1)
computational neuroscience (5)
neural networks (5)
heuristics (1)
heuristic algorithms (1)
artificial neural network (5)
heuristic (computer science) (1)
classification algorithms (8)
support vector machines (3)
statistical classification (3)
trajectory optimization (1)
aerospace engineering (1)
aerodynamics (1)
ballistics (1)
computational complexity theory (10)
analysis of algorithms (1)
time complexity (1)
computational resources (1)

Figure B.1: Topic map built from 100 abstracts on the topic "quadratic programming".