# Defining and Measuring Data-Driven Quality Dimension of Staleness

Oleksiy Chayka, Themis Palpanas, Paolo Bouquet

April 2012

Technical Report # DISI-12-016
Version 2.0

# Defining and Measuring Data-Driven Quality Dimension of Staleness

Oleksiy Chayka[1], Themis Palpanas[1], Paolo Bouquet[1]
{oleksiy.chayka; themis; bouquet}@disi.unitn.it
[1]University of Trento, Italy

**ABSTRACT**

With the growing complexity of data acquisition and processing methods, there is an increasing demand in understanding which data is outdated and how to have it as fresh as possible. Staleness is one of the key, time-related, data quality characteristics, that represents a degree of synchronization between data originators and information systems possessing the data. However, nowadays there is no common and pervasive notion of data staleness, as well as methods for its measurement in a wide scope of applications.

Our work provides a definition of a data-driven notion of staleness for information systems with frequently updatable data. For such a data, we demonstrate an efficient exponential smoothing method of staleness measurement, compared to naïve approaches, using the same limited amount of memory, based on averaging of frequency of updates.

We present experimental results of staleness measurement algorithms that we run on history of updates of articles from Wikipedia.

**Keywords:** Data quality, information quality, data freshness, exponential smoothing, data quality measurement.

## 1. INTRODUCTION

Growing diversity of data sources and technologies makes solutions for data synchronization problems more and more complex. Nowadays, data is created, processed, and eventually consumed by the time when it can be already obsolete. The dynamics of today’s world raise new challenges for data management, one of which is having data fresh at instant of its consumption.

Since the notion of “data freshness” (or “staleness”, as the opposite) is not ubiquitous among researchers, in the next section we will first give a motivating example where such a notion, as a time-related quality dimension, is crucial. Necessary limitations on possible approaches for data staleness measurement are followed by a description of an information system that may incorporate an approach we will present later in the paper.

## 1.1 Motivation and Outline of the Work

While there are numerous works on data caching and synchronization, including those that consider system-driven data freshness issues (see Section 2), there are underexplored aspects such as understanding and measuring data-driven quality dimensions relevant to time. In particular, when users are the only source of data, system-driven aspects cannot help us as much to detect if a data element[1] is fresh, since those aspects deal with management of data rather than with interpretation of its nature. We will demonstrate this statement in our following examples involving use cases of getting insights from articles of Wikipedia and from entries representing real world entities in an information system called Entity Name System ( [1], [2], [3]).

In this work, scope of interest in Wikipedia analysis lays in analysis of its articles' revisions history, or series of updates of high frequency which is about 0.3 to 3 updates per day. The lower limit of the update rate is driven by characteristics of the algorithms we explore in this work (Section 5) while the upper boundary is dictated by the natural limitations of Wikipedia of having insignificant number of articles with high update rate throughout selected evaluation interval (we selected history for about 8 years – see details in Section 6). Hence, we target to deal with articles of least 1000 revisions per their lifetime.

Updates to Wikipedia may result not only from legitimate updates from users, but also from robotic or vandalism actions, which are presumed by openness of this resource. Even different viewpoints on the same aspect may result in updates. In our work, we consider all those kinds of updates as a part of an "eco-system", where those events are natural with respect to provided resources and corresponding data alteration rules. Hence, we can build a model that will predict an update of a data element based on history of updates, but not reasons for them. Given that, one can apply our approach to data-driven staleness measurement in other information systems operating with data-driven updates.

One of main questions we want to answer is *How stale is an article now?* For example, given an extraction from revision history of article from Wikipedia "List of web application frameworks" (Figure 1), a disruption of nearly daily updates for more than 2 years may indicate a staleness issue of the article.

[1] *Data element* is an abstraction that denotes attributes ("name-value" pairs) or entities (aggregated set of attributes of notions like person, place, event, abstract object, etc.).

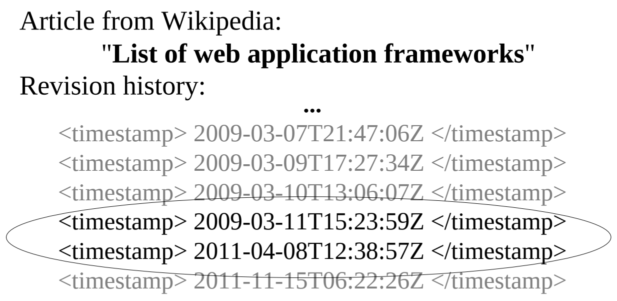

Article from Wikipedia:
"**List of web application frameworks**"
Revision history:
**...**
<timestamp> 2009-03-07T21:47:06Z </timestamp>
<timestamp> 2009-03-09T17:27:34Z </timestamp>
<timestamp> 2009-03-10T13:06:07Z </timestamp>
<timestamp> 2009-03-11T15:23:59Z </timestamp>
<timestamp> 2011-04-08T12:38:57Z </timestamp>
<timestamp> 2011-11-15T06:22:26Z </timestamp>

**Figure 1. Example of potential data staleness issue in an article from Wikipedia.**

Note, that from the other hand, for the measurement of staleness we want to account its semantics as close to real world as possible. In the example above, it would be improbable, that somebody forgot about updates of a popular article on Wikipedia. Thus, for a reader inquiring on how fresh the article is, we want a measurement method resulting in "2 days stale" on 2009-03-14, but "fresh" a month or more after last day in a series of updates in 2009[2].

Since we want to know at any time how stale is a data element, we impose the following a strict limitation on the required memory budget, and consequently on the use of the revision history. We want to measure data staleness without storing and analyzing even a (recent) part of update history, but considering it indirectly with help of a few variables. Those variables must represent update history concisely, at the same time been adequate for the measurement accuracy. In this work, we will show how we accomplish such a requirement with an exponential smoothing method, comparing it to averaging methods.

Without a limitation on time-series representation, our solution for the problem becomes infeasible in real time information systems that must convey to end users staleness measures of millions of entities, with at least thousands of timestamps each. In fact, the problem would fall into another extensively explored problem area – analysis and prediction of time-series with performance optimization aspects.

An example demonstrating not only need for staleness measurement and exposure it to end user, but also its potential propagation to an information system, is an Entity Name System (for the rest of this paper, we will refer to it as *ENS*, or simply, *repository*) developed within OKKAM[3] project. ENS aims at management of global unique identifiers for entities on the web, storing description of those as sets of attribute name and value pairs for each entity. Users of the system can modify the repository at any time. Usually, this event reflects updates that took place

[2] Naturally, a measure of staleness in this case is relevant only soon after interruption of updates (on 2009-03-11), but not even a month after it: the interruption was due to a new status of the original item – redirecting article.

[3] http://www.okkam.biz

for corresponding entities in the real world, but as we mentioned before, in this work we study a set of updates as is, without delimiting reasons for them.

The questions in staleness measurement in the ENS are similar: *How stale is each attribute of each entity?* Given a staleness measure, *Should an attribute be updated?* The latter question is out of scope of the current work, but it is of particular interest for data synchronization issues, whenever a system like ENS will decide to synchronize some entities with such a source like Wikipedia (which may already have data staleness measured, as we mentioned in the example before). In fact, some works for data caching and synchronization presuppose presence of either fresh data at sources or its degree of staleness (see Section 2) that can serve for decision on which source to query.

By calculating with help of statistical methods when user (as a source of truth) should have updated data element, we can estimate its degree of staleness. Based on such estimation, one can distribute resources of an information system in such a way to keep the repository partially or entirely as fresh as needed, by means of various synchronization techniques.

In summary, our contributions in this work are the following:

- we define a data-driven notion of staleness and show how it satisfies key requirements for data quality metrics;
- we demonstrate different approaches for staleness measurement without analyzing historical timestamps of updates, but considering them indirectly via as many as three variables;
- for analysis of revision history from Wikipedia, we implemented staleness measurement algorithms based on averaging and exponential smoothing methods.

The rest of this paper is organized as follows. In Section 3 we will first set a data-driven notion of staleness that is based on existing state of the art (Section 2). After that, we will provide a satisfiability analysis of our notion to base data quality requirements (Section 4), and demonstrate different approaches to measure data staleness (Section 5). Section 6 shows experimental results of those approaches implemented and tested on a Wikipedia update metadata, with comparative analysis of their predictive accuracy. Concluding remarks are presented in Section 6.1.

## 2. RELATED WORK

In spite of the fact that time-related data quality dimensions have been studied by computer scientists ( [4] explored how to ensure required currency level of materialized views) even before main data quality research in this area began ( [5], [6]), they still lack a comprehensive measurement methodology that can be applied in practice.

Though both academics and practitioners find a time-related quality dimension among the most important ones [7], there is still need in common understanding and defining ubiquitous notions of those. Because of this fact, such time-related quality terms as freshness, timeliness, currency, up-to-dateness, age, staleness, obsolescence may be used to denote the same quality problems (up to antonymous equivalence). Bouzeghoub and Peralta [8] have presented a structured view of some of those terms in the table below.

**Table 1. Freshness factors and metrics according to [8].**

| Factor | Metric | Definition |
|---|---|---|
| Currency | Currency | The time elapsed since data was extracted from the source (the difference between query time and extraction time). |
| | Obsolescence | The number of updates transactions/operations to a source since the data extraction time. |
| | Freshness rate | The percentage of tuples in the view that are up-to-date (have not been updated since extraction time). |
| Timeliness | Timeliness | The time elapsed from the last update to a source (the difference between query time and last update time). |

Though Bouzeghoub and Peralta [8] studied freshness-related metrics, they have concentrated on analysis of definitions of data freshness in literature. Measurement of those metrics was out of their scope.

Heinrich et al. [9] is one of recent works where authors focus on evaluation of time-related data quality metrics, elaborating a set of corresponding requirements for them. We discuss satisfiability of our notion of data-driven staleness metric to those requirements in Section 4.

Authors of other related works (expanded list of the notions, definitions and measurement methods of those works see in Appendix 1) measure data freshness using known update rate of a monitored element [10]. In [11] they study incorporation of a freshness parameter into OLAP queries processing. The more recent work of Guo et al. [12] presented an integration of currency and consistency requirements of users into SQL queries; Labrinidis and Roussopoulos in [13] proposed an algorithm that allows users to get data based on their performance and freshness preferences. Qu and Labrinidis [14] introduced Quality contracts as a way to express user preferences for *speed* of data delivery vs. delivery of *fresh* data; in [15] authors propose a model that allows user to specify freshness constraints and read out-of-date data within a serialized transaction. For example, if prices for an item at an auction change, user may want to get quick response with old prices if staleness of the data will not be beyond a specified threshold. In our work, instead of aiming at satisfaction of user freshness preferences expressed in user queries, we measure current staleness of monitored elements based on prediction of the most recent updates that should have taken place in the past.

Golab et al. [16] define the notion of data staleness (“a difference between time *t* and the timestamp of the most recent tuple in table *T*”) and study this quality measure for scheduling of updates for a real-time warehouse, focusing on management aspects of data synchronization between two systems, rather than on data-driven characteristics we pursue in this paper.

Definition of freshness and approaches to measure it for data replication was given by [17], where authors study a problem of maintaining a cache at required level of currency, consistency, completeness and presence (defined by users). Cho and Garcia-Molina [18] have shown how web crawler should update its cache to keep it as fresh as possible. Xiong et al. [19] described how to plan updates for data objects with known validity intervals. Akbarinia et al. [20] showed an approach to keeping replicated data in P2P systems consistently updated and fresh; in [21] authors demonstrate lazy replication technique with freshness guarantees by processing timestamps. In one of the most recent works, Xiey et al. [22] provide a method to ensure that outsourced database system correctly performs update operations (and hence, has data fresh). For this purpose, they have a twofold approach: adding timestamps to data signatures or adding fake operations of insert, delete or update. Afterwards they check for correctness of execution by an outsourced DB processing data modified by either of the two above-mentioned ways. Following the defined notion of freshness, we abstract from data freshness guarantees driven by system properties (choice of the most fresh sources, assurance that sources execute update operation correctly, etc.).

In this work, we study the nature of data staleness that is a data-driven characteristic that depends on data element’s frequency of updates. It does not depend on system properties that are usually considered in works studying data replication and caching techniques.

Cho and Garcia-Molina [23], [24] gave the most relevant notions that we can adapt in our work. They study a problem of keeping a cache of web data as fresh as possible by means of discovery update rate of web data and defining a strategy to query the data.

In the next section, we show how our data-driven notion of *staleness* comprehends necessary semantics of existing system-driven notions of freshness and age elaborated by Cho and Garcia-Molina ( [23], [24]).

## 3. NOTION OF DATA STALENESS

Quantitative measurement of a data quality (DQ) dimension requires strict notion defined. Due to diversity of application areas, there is no coherent view on notions of DQ dimensions even in a DQ community. This section aims at a deeper analysis (started in Section 2) of notions for time-related DQ dimensions presented so far. In particular, we will inspect notions of “freshness” and “age” that we found to be the most relevant ones to the goals of our work.

Cho and Garcia-Molina [23], [24] define *freshness* as a binary measure that indicates whether a data element *e* from the real world is synchronized with its copy in information system. Another quality indicator employed in their works is *age* that has the following semantics: age of fresh data is always zero while age of non-fresh data continuously and linearly grows starting from synchronization disalignment point (i.e., when a copy of a data element is not synchronized anymore). Thus, age measures time elapsed from synchronization disalignment point until instant of measurement.

One of major differences in notions of DQ measures between work of Cho and Garcia-Molina and ours is in definition of disalignment point. They study correspondence of a local copy of web data to its "real-world state" on a remote system, while in our work we do not have a remote system to query the data updates. We rather consider users that may update the data at any time.

From the other hand, both "freshness" and "age" in [23] served for studying best strategies to keep cache of web data fresh. In our work, we consider a different problem, namely prediction of next updates of each monitored data element without analysis of history of updates (revisions). As we mentioned before, the history, nevertheless, must be considered indirectly. Thus, for enabling measurement of data staleness, in our work we define it via linear function corresponding to that of age defined by Cho and Garcia-Molina [23], but with their semantics of freshness.

## 3.1 Definition of Data Staleness Measure

Ideally, a data-driven notion of staleness should not consider the fact of synchronization between systems, but it will rather approach to derive a measure of correspondence (in time) of a data element's value to its real world's instance. In reality, it is often too hard or just impossible to get real-world's value. Hence, we consider user updates, which potentially represent alterations in the real world, but in any case, those updates represent changing web data we want to study in this work.

We suppose that whenever a system misses update that has to take place, it will have data element that does not correspond to the real world. Instant of such a missing update we call potential synchronization disalignment point. To measure staleness of a data element $e$ at current time $t_{cur}$ we have to find potential synchronization disalignment points of that element, in respect to its real updates (e.g., made by users to our system) and those predicted by an algorithm.

Whenever predicted (P) update will precede the corresponding prospective real (R) one, there is a possibility of presence of staleness. In this case, data staleness $S^e_{t_{cur}}$ of element $e$ at current time $t_{cur}$ can be measured as difference between current time $t_{cur}$ and time of predicted update $t^e_{pr}$; otherwise, if predicted update will take place on or after current instant, staleness will remain zero:

$$S^e_{t_{cur}} = \begin{cases} t_{cur} - t^e_{pr}, & t_{cur} > t^e_{pr} \\ 0, & t_{cur} \le t^e_{pr} \end{cases} \qquad (1)$$

Graphical examples of two possible cases of staleness measurement (according to eq. 1) are shown in Figure 2, where diamonds represent real updates, whether they already took place (solid lines) or they are potential future ones (dotted lines).

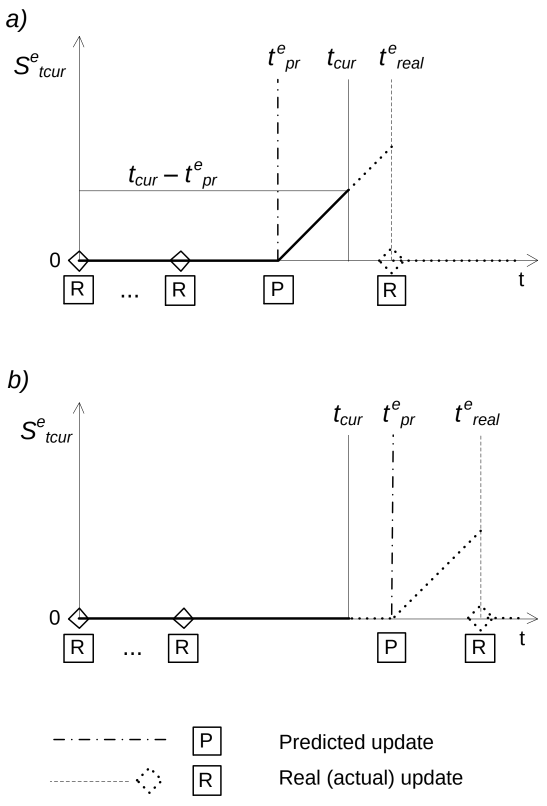


**Figure 2. Possible cases of staleness measurement (eq. 1):**
**a) instant of measurement ($t_{cur}$) falls between predicted and real updates;**
**b) instant of measurement precedes both predicted and real updates.**

## 4. REQUIREMENTS FOR TIME-RELATED QUALITY METRICS

There are various requirements for DQ metrics from researches of the field. Heinrich et al. [9] have gathered a coherent set of key requirements for time-related metrics. Among those are (R1) normalization, (R2) interval scale, (R3) interpretability, (R4) aggregation, (R5) adaptivity and (R6) feasibility. In the following subsections, we will describe each of these requirements, and demonstrate satisfiability to each of them of our notion and approach to measurement of data staleness. We will also demonstrate compliance of our notion to principal requirements for DQ dimensions outlined by Pipino et al. [25].

### 4.1 NORMALIZATION

One may need normalization of a data quality metric while operating with different metrics, instead of a sole one needed for our examples (Section 1.1). Data values normalization in this context means mapping of all possible measurement values of a data quality metric to interval [0,1].

We can get normalized measure of staleness via exponential function as follows:

$$S_{N,t_{cur}}^{e} = \begin{cases} \exp(-S_{t_{cur}}^{e}), & t_{cur} > t_{pr}^{e} \\ 1, & t_{cur} \le t_{pr}^{e} \end{cases} \tag{2}$$

with values ranging from 0 (absolutely stale data element) to 1 (the best possible quality of data element on the dimension of staleness), according to [25].

Equation 2 is a generalized formula for automated calculation of normalized staleness values, where value of 0 is reachable at infinity. In the applications that are enforcing update policies, state of "absolute stale data" is reachable at a finite time. For example, consider a system where each user must change own password every 6 months. Those passwords without been updated during more than 6 months, are not valid in the system, and can be treated as absolute stale elements. However, such cases are outside of scope of our work, since policy enforcement and data invalidation is covered to a higher extent by another quality dimension – validity.

## 4.2 Interval scale

To support interpretability of quality measurement results for their comparison with measurements of other dimensions, those results should support interval scale property. This means that the same interval should denote the same quality improvement or degradation.

This property is supported by our (non-normalized) definition of staleness due to its linearity: difference of an element's staleness between values 3 and 4 (days) means the same as the one between 6 and 7 (days).

Note, that interval scale function may differ from the linear function (eq. 1), or be impossible to pursue in some cases – normalization is one of those for our notion. Hence, depending on application, staleness measurement function may be either normalized (to facilitate comparison between quality metrics), or interval scaled (for better interpretation of the results).

## 4.3 Interpretability

Easiness of interpretability of measurement results by end-users is also important for definition of a quality metric. For example, when an analyst has data with freshness metric equals to 0, does it mean to have fresh data at hand? What about freshness equals to 10 (suppose, we do not stick to the notion proposed in [23])? Is it even fresher? Similar issues may arise with the notion of age: e.g., with age *A(e) = 0*, we cannot undoubtedly speak about positive or negative data characteristic because of a semantic meaning of "age" that mostly corresponds to a neutral notion of "period of time" [26]. Unless specific notion of freshness or age is communicated to the end-user, interpretation of that may be ambiguous. To reduce such an ambiguity, we came with a notion that comprehends time-related characteristics of data, simplifying its perception by end user.

Considering the abovementioned example, with staleness $S^e = 0$, we speak about absence of (a time-related) negative feature, while $S^e > 0$ clearly indicates problems with data. Hence, from a user perspective, the notion of data staleness satisfies the requirement of interpretability, suggested by [27].

## 4.4 Aggregation

While measuring quality metric at one level, it is important to get aggregate value at higher one(s). For example, having result of staleness measurement for attribute "name", how it will influence staleness of a corresponding entity, table and entire database?

We define aggregation property at database level as ratio of weighted sum of all measures of staleness of all measured data elements (attributes, entities, etc.) in a database, to their amount:

$$S^{DB} = \frac{\sum_{e \epsilon DB}[imp^e \cdot S^e]}{|e \epsilon DB|} \tag{3}$$

where importance rate $imp^e$ represents weight of an element $e$ in a database.

Definition of importance rate depends on its application and context. Since in this work we focus on measuring data staleness dimension, for simplicity reasons we consider normalized number of queries for a data element as its measure of importance.

## 4.5 Adaptivity

Usually, to interpret measurements of quality metrics, those metrics should be adopted for a given context. While this is true only for some metrics, as [5] noted, most of them are context-independent and can be objectively evaluated without such an adaptation.

By definition, staleness is one of those objective metrics. However, as we have mentioned before, one can enforce adaptability of low-level measurement results for higher-level DQ assurance goals. For example, data administrator may set a warning if attribute's staleness reaches a certain threshold, and may set an automated request for update if staleness will reach even higher threshold.

## 4.6 Feasibility

Techno-economical requirements of applications where quality measurement takes place, imply feasibility of getting the results. For example, getting a measure of reputation of an external source may be infeasible in some cases.

As we will show in the next section, our approach for getting staleness measure relies on parameters that are essential and normally easy to get for a data element at a source system – total number of updates, timestamps of first and last update, etc.

## 5. APPROACHES TO DATA STALENESS MEASUREMENT

In the previous sections, we have established a data-driven notion for quality dimension of staleness (Section 3) and properties, that the corresponding measurement methodology should possess (Section 4). In this section, we will demonstrate examples of possible approaches to measure data staleness under given constraints on variables for representation a history of updates (Section 1.1).

In particular, in Section 5.3 we will present an exponential smoothing method that indirectly considers (by means of a few representative variables) history of updates. This approach is preceded by two naïve ones, enhanced averaging method and shifting window, that directly calculate instant of predicted update $t_{pr}^{e}$ for eq. 1. In Section 6, the naïve approaches serve as a baseline to compare accuracy of prediction results of the proposed methods.

### 5.1 ENHANCED AVERAGING METHOD

Consider a data element $e$ that has nearly periodic updates committed by users of an information system. We call such updates "real" ones (as opposed to "modeled" that are approximations of real updates, extrapolated by a measurement algorithm, and "predicted" that is a sole next update from time of measurement, forecasted by the algorithm).

According to eq. 1, to measure data staleness $S_{t_{cur}}^{e}$ of element $e$ at current time $t_{cur}$ (or another test point), we need to have an instant of time of predicted update $t_{pr}^{e}$, that most probably should have taken place for $e$ in the period between instant of last update and $t_{cur}$.

For periodically updatable data element, this task seems to be straightforward. Number of all updates committed to a data element up to the test point $t_{cur}$, gives us an average update rate during past time interval. By adding (extrapolating) one more period, we can have an instant of potential update $t_{pr}^{e}$, and hence, $S_{t_{cur}}^{e}$ (see Figure 3).

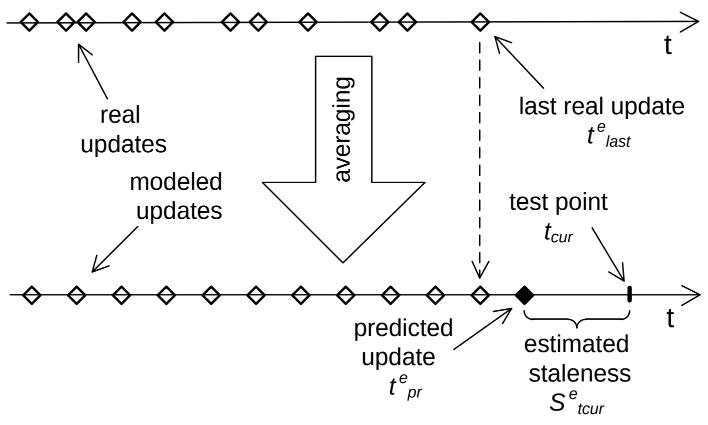


**Figure 3. Example of prediction of element's update by enhanced averaging method.**

Since this method provides averaging of updates given that we have timestamp of last update (in addition to lifetime elapsed and total number of updates), we called it *enhanced averaged-based* method.

Formally, we define instant of predicted update as following:

$$t^e_{pr} = t^e_{last} + \frac{1}{\lambda^e} \tag{4}$$

or, substituting update rate $\lambda^e$ with number of updates $N^e$ during element's lifetime from its creation $t^e_0$ to its last update $t^e_{last}$, we have:

$$t^e_{pr} = t^e_{last} + \frac{t^e_{last} - t^e_0}{N^e} = \frac{t^e_{last} \cdot (N^e + 1) - t^e_0}{N^e} \tag{5}$$

This allows us to obtain the final formula for staleness measurement of a data element *e* by means of enhanced averaging method:

$$S^e_{t_{cur}} = \begin{cases} t_{cur} - \dfrac{t^e_{last} \cdot (N^e + 1) - t^e_0}{N^e}, & t_{cur} > t^e_{pr} \\ 0, & t_{cur} \le t^e_{pr} \end{cases} \tag{6}$$

Hence, to measure data staleness using this naïve method we would need to store 3 variables for each data element: instants of first and last updates $t^e_0$, $t^e_{last}$ and total number of its updates $N^e$.

In spite of the fact that most of web data we operate with, is following Poisson distribution (showed by [23]), entities with aperiodic or irregular updates motivates us to consider more accurate prediction methods that have to account such irregularities. Those methods, from the other hand, must be at least as efficient (in terms of variables per data element) and accurate as the enhanced averaging method.

## 5.2 SHIFTING WINDOW

One of the simplest methods that account entities with aperiodic updates without boosting number of variables to feed the measurement algorithm, is shifting window.

A key advantage of this method is in localizing periods with different frequencies of updates $\lambda_1^e, \lambda_2^e, \ldots$ within certain intervals – windows (Figure 4). Shifting window can be seen as a refinement of the averaging method described before (Section 5.1) in such a way, that prediction of next update considers only recent updates (of current window) rather than older ones, unless current window has a few of them, as we will show later.

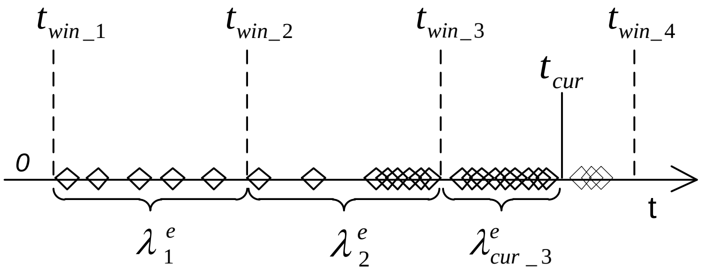


**Figure 4. Irregular updates delimitation by shifting window.**

In this work, we focus on implementation of this method with window of fixed size, rather than on analysis of window size itself (which may also require from a data analyst manual processing of history of updates of entities of the same type). Hence, we will take a shifting window of fixed length $|t_{win}|$, instances of which will sequentially cover entire lifetime of each data element under staleness measurement, as in Figure 4.

From the definition of this method it is obvious, that in most cases with web data even a few windows per lifecycle of an entity should improve accuracy of staleness measurement (as we mentioned, we target entities with update rate of order at least 1000 per lifecycle). From the other hand, computational resources and intervals with sparse updates dictate upper limit on number of windows for each application.

Assigning time periods $t = \{0,1, \ldots\}$ that correspond to each of these windows, we will operate with time stamps indicating start of each of such window $t_{win\_t} = \{t_{win\_0}, t_{win\_1}, \ldots\}$.

For each window of current period $t$, we keep tracking number of updates committed since instant of start of the window $t_{win\_t}$ and until current (or test) instant $t_{cur}$. This gives us current frequency of updates in current window of period $t$:

$$\lambda_{cur\_t}^e = \frac{N_{cur\_t}^e}{t_{cur} - t_{win\_t}} \qquad (7)$$

However, as we mentioned before, until there are enough updates in the beginning of each window, we cannot realistically estimate its expected update frequency. Hence, we set a threshold of three updates for each window. This threshold triggers calculation of current window's update rate as follows. Until we have sufficient updates in current window ($N^e_{cur\_t} < 3$), we consider previous window's update frequency also as current one; otherwise, current update frequency is calculated as ratio of committed updates in the current window to elapsed time from instant of start of the window to current instant:

$$\lambda^e_{cur\_t} = \begin{cases} \lambda^e_{t-1}, & N^e_{cur\_t} < 3 \\ \dfrac{N^e_{cur\_t}}{t_{cur} - t_{win\_t}}, & N^e_{cur\_t} \geq 3 \end{cases} \tag{8}$$

Combining eq. 1 for case $t_{cur} > t^e_{pr}$ with eq. 4 and eq. 8 (given that $\lambda^e_{cur\_t} = \lambda^e$), we have:

$$S^e_{t_{cur}} = \begin{cases} t_{cur} - \left(t^e_{last} + \dfrac{t_{cur} - t_{win\_t}}{N^e_{cur\_t}}\right), & N^e_{cur\_t} \geq 3 \\ t_{cur} - \left(t^e_{last} + \dfrac{1}{\lambda^e_{t-1}}\right), & N^e_{cur\_t} < 3 \end{cases} \tag{9}$$

As one can see, this method requires the following variables: 1) number of updates in current window $N^e_{cur\_t}$, 2) previous window's update rate $\lambda^e_{t-1}$ and 3) time of last update $t^e_{last}$. Hence, this methods turns out to be as efficient in terms of required variables, as enhanced averaged-based method.

Note, that since we operate with entities of the same high order of update frequency (see Section 1.1), we can adjust boundaries of all shifting windows $t_{win\_0}, t_{win\_1}, \ldots$ for all data elements, with precision of order $t$. In fact, such an adjustment can reduce only length of $t_{win\_0}$ that we treat as time of entity initialization. Thus, we can store and track all boundaries for shifting windows for entire data repository, instead of individual elements.

For the entities with high update rate, we may not necessarily need a precise calculation of predicted instant of update. Instead, we can try to operate with small intervals, within which a method will predict presence or absence of update(s). The next section demonstrates implementation of this idea via exponential smoothing method.

## 5.3 EXPONENTIAL SMOOTHING

As we mentioned before, staleness measurement with help of shifting window uniformly considers all updates within a window of size $|t_{win}|$. In this section, we will show how one can exponentially account a wider (up to entire lifecycle) scope of updates, reducing potentially unnecessary precision of prediction for each next update instant for elements of high order of updates that we deal with.

Among a wide variety of exponential smoothing methods [28], we focus on the N-N methods, which do not consider trend and seasonality of time-series. We adjusted one of those methods for prediction of presence of an update during time period. This method is similar to shifting window method (Section 5.2), but with much finer granularity with respect to update frequency.

One of key elements in measuring data staleness with exponential smoothing method is a time interval that for better reflection of its semantics we called Predictable Time Interval (PTI). Such an interval is an equivalent of shifting window of granularity of order of updates. This means that, on average, for each update event there should be a few PTI's.

For all updates in each PTI we assign exponentially growing weights, indicating importance of those in prediction of future updates (see Figure 5), and hence, measuring current staleness value (we will give more details on the weights later on in this section).

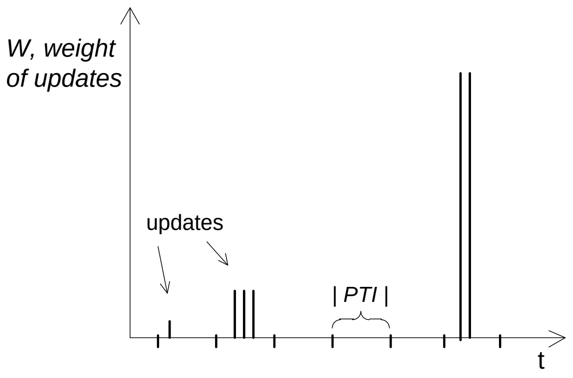


**Figure 5. Example of exponential growth of weights associated with updates in each PTI.**

The choice of length of those intervals depends on foreseen update rate of observed data element, which can be also estimated based on historical analysis of entities of same type. Hence, one can approach to calculation of Predictable Time Interval length as follows:

$$|PTI| = \frac{\frac{\sum t^e}{|e|}}{\frac{\sum N^e}{|e|} \cdot PTI_{PC}} \cdot 24 = \frac{1}{\lambda \cdot PTI_{PC}} \cdot 24 \tag{10}$$

where $\frac{\sum t^e}{|e|}$ is average lifetime of elements in a set; $\frac{\sum N^e}{|e|}$ is expected average number of updates of elements in a set during their lifetime; $\lambda$ is an average update rate of elements of a set; $PTI_{PC}$ is a precision coefficient, indicating required granularity of a predictable time interval. This coefficient depends on application, and to a higher extent, on expected update frequency of data.

Since our ultimate goal is measurement of data staleness, which is a data characteristic that can be useful in a decision making, we impose the following empirical granularity limits on rate of PTIs (or "tick size") to frequency of updates. From one side, a frequency of updates higher than 2 per tick interval will deprecate value of the prediction result of the presented algorithms (in most predictions the algorithms will foresee an updatable tick). Accountability of the recent updates (a few ticks range) imposes the lower limit – we require for the input data to have no less than 1 update per 3 ticks, therefore setting $PTI_{PC} = 1.2$. Hence, for the groups of selected articles with update rates $0.4 \leq \lambda \leq 2.5$ (1000 to 7500 updates during about 6 years) the formula 10 suggests use of PTI intervals from about 8 to 49 hours.

Now, we want to assign a weight to each PTI that will depend on all previous updates in each PTI, given that for prediction of the next update we consider more recent updates to a higher extent. In this way, we define averaged weighted linear combination of all updates of a data element *e*:

$$Savg_t^e = \alpha \cdot N_t^e + \alpha \cdot (1\text{-}\alpha) \cdot N_{t-1}^e + \alpha \cdot (1\text{-}\alpha)^2 \cdot N_{t-2}^e + \ldots \qquad (11)$$

where $N_t^e$ is number of updates that element $e$ had for time interval $t$ (the most recent); $\alpha$ is a smoothing constant such that $0 \leq \alpha \leq 1$ (this constant drives scale of growth of weights, associated with each PTI).

From eq. 11 we have the following recursive formula:

$$Savg_t^e = (1\text{-}\alpha) \cdot Savg_{t-1}^e + \alpha \cdot N_t^e \qquad (12)$$

We consider weighted linear combination of each PTI $Savg_t^e$ as a probability that the next PTI will have an update. Whenever $0.5 \leq Savg_t^e < 1$, it shows us that the next time interval should have at least one update. The same prediction applies to $Savg_t^e \geq 1$, which shows us that previous PTI accommodated more than one update. $Savg_t^e < 0.5$ indicates that next PTI most probably will not have an update.

While measuring data staleness with this method, we also need for each PTI interval a staleness coefficient $PTI_t$ that will indicate how stale a data element is (or it is not stale but fresh). This makes the method a perfect fit as a solution to our motivating example (Section 1.1), where we imposed a requirement for a measurement method that it should result in a corresponding element's staleness value soon after last update. From the other hand, it should consider that element fresh if last update took place long time ago, compared to expected update rate of element.

Staleness coefficient $PTI_t$ has the following logic. As soon as predicted presence or absence of updates during the next PTI corresponds to real case, $PTI_t$ equals to 0; in case of inconsistency between prediction and real case, $PTI_t$ is increased by 1 if $Savg_t^e$ suggests an update, unless it is not reset to 0 by presence of a real update event during past PTI – see Table 2.

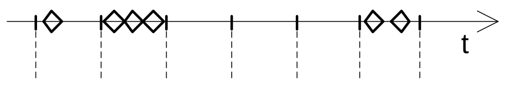
**Table 2. Logical table for $PTI_t$ staleness coefficient**

| Presence of updates (prediction by $Savg_t^e$ / real case) | |
|---|---|
| No / No | $PTI_t = 0$ |
| No / Yes | $PTI_t = 0$ |
| Yes / No | $PTI_t = PTI_t + 1$ |
| Yes / Yes | $PTI_t = 0$ |

Using $PTI_t$ staleness coefficient, we can get staleness value by the end of each PTI by its multiplication by length of Predictable Time Interval as follows:

$$S_t^e = PTI_t \cdot |PTI| \tag{13}$$

Figure 6 demonstrates on a sample history of updates, an instantiation of the exponential smoothing method, with $\alpha$ = 0.8. In this figure, we can see initial 6 intervals (PTI's) delimited on the timeline by instants $t = 0..6$. Each rhombus in the figure denotes an update to an element. According to eq. (12), Table 2 and eq. (13), we calculate values of $Savg_t^e$, $PTI_t$ and eventually, staleness value $S_t^e$, at each instant of time $t = 1..6$ (setting 0 as corresponding initial values at the first step).



| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $Savg^e_t$ | 0 | **0.8** | **2.56** | **0.512** | 0.1 | 0.02 | **1.6** |
| $PTI_t$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $S^e_t$ | 0 | 0 | 0 | 1*\|PTI\| | 0 | 0 | 0 |

| | |
|---|---|
| $Savg^e_t$ | averaged weighted linear combination of history of updates |
| $PTI_t$ | staleness coefficient of Predictable Time Interval (PTI) |

**Figure 6. Example of staleness estimation with exponential smoothing method.**

As one can see, in terms of stored variables, this method is as efficient as those presented in this section before: it requires 3 individual variables for measurement of data staleness, namely 1) weighted linear combination for previous time interval $Savg_{t-1}^e$; 2) number of updates for current interval $N_t^e$ and 3) staleness coefficient $PTI_t$. The rest of required variables ($|PTI|$, $\alpha$, boundaries of PTI's) one can set common for entire data repository, or individual for some of data elements, depending on availability of resources and a given task.

In the next section we will demonstrate that exponential smoothing method gives the most accurate update prediction results (and hence, more accurate staleness measurement at equal other conditions) with fewer errors for entities with aperiodic updates.

## 6. PREDICTIVE ACCURACY EXPERIMENTAL COMPARISON

In our experiments, we compared the best predictive accuracy of the approaches proposed in Section 5, given the least error rates of each of the method with different values of the corresponding parameters (tick size, shifting window size and α coefficient). In particular, we measured how often a method can correctly predict every next update of each article, and how many errors it makes. By erroneous prediction we consider presence of non-predicted update and absence of a predicted one. In the table Table 3 one can see a summary table representing cases of correct and erroneous predictions accountability in our analysis methodology.

**Table 3. Accountability of correct and erroneous predictions of updates with respect to the real ones.**

| | | **Real update** | |
|---|---|---|---|
| | | present | absent |
| **Predicted update** | yes | OK | Error |
| | no | Error | n/a |

As one can see from their description, enhanced averaged-based approach and shifting window can predict a precise instant of update, while exponential smoothing method predicts presence of an update in an interval. To compare predictive accuracy of those methods, in our experiments we have adopted the two former approaches for prediction of next update during a time interval (predictable time interval, PTI) of the same length that exponential smoothing method will use.

As a dataset, we took revision history of articles in English from Wikipedia[4], covering around 3.8 million items from January 16, 2001 to December 01, 2011 (we used for reference purposes a time-series[5] representing metadata of articles till May 12, 2009; in particular, it helped us to verify motivating example we gave in Section 1.1). As we intended to explore the algorithms targeted to deal with a data that is getting updates at high frequency (at least 1000 updates per lifecycle, which is at most 11 years for the Wikipedia's dump we analyzed), we selected groups of articles with the following number of revisions (number of articles in each group is given in

[4] http://dumps.wikimedia.org/enwiki/20111201/enwiki-20111201-stub-meta-history.xml.gz
[5] http://download.wikimedia.org/enwiki/20090512/enwiki-20090512-stub-meta-history.xml.gz

parentheses): 1000±10 (no. 601), 1500±10 (191), 2000±10 (95), 2500±10 (54), 3000±10 (19), 3500±10 (15), 4000±10 (16), 4500±10 (14), 5000±10 (2). Articles of these groups we used as an input data for exploration of properties of the described algorithms. Due to the natural limitations of Wikipedia (among all articles, there is significant decline of number of those with a few thousands revisions), we also selected articles with revisions 7500±150 (no. 5) that we used for extrapolation and confirmation of the results at higher update rates.

For more competitive comparison of the results from the presented algorithms, we want to align articles revision histories lengths, i.e., eliminate recently (close to the final point of the revision metadata taken) created articles that fell into one of our groups, therefore, making update density of each group of articles more equal. For example, Figure 7 represents updates distribution of two sample articles of the same update rate – 5000. Among those two kind of articles we are interested the most to experiment with the one having wider distribution in time of the same number of revisions, and with less global peaks, which should potentially give more credit to those prediction methods that can adapt faster to varying update density (which shifting window and exponential smoothing will demonstrate in the following experiments).

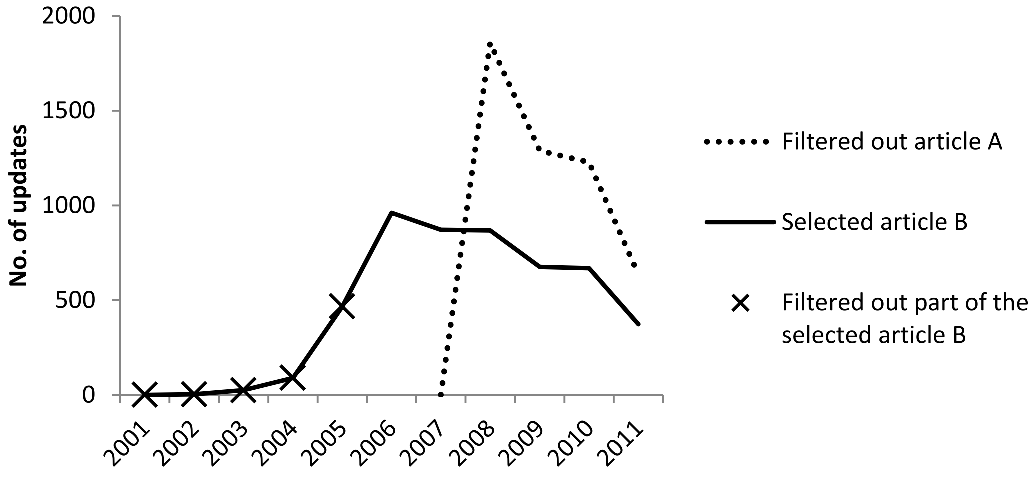


**Figure 7. Yearly updates for two sample articles (A and B) of update rate 5000.**

Hence, in all the initially selected groups we filtered out articles with their first revision after 2006-01-01 (like article A in Figure 7). This allowed us to leave for the analysis the following number of items of their corresponding groups: 495 (1000±10), 157, 83, 47, 15, 12, 13, 14, 1 and 3 (7500±150).

Taking into account the empirical granularity limits mentioned before (no more than 2 updates per tick interval, but no less than 1 update per 3 ticks, where “ticks” is a variable parameter taken as 1 day for preliminary data selection purposes), we want to take out of the experiments a period of time when the least updatable group of the selected articles (1000) got revisions less than 1 per 3 days on average during one year. As our preliminary data analysis showed (Figure 8), before 2006-01-01 average number of updates among all the selected articles of update rate 1000 was at most 79 per year, while average yearly revisions for 2006 became about 180 for this group.

Hence, for all the selected articles, we set the starting point to begin accounting the updates as January 01, 2006, when there were near 1 million articles in the Wikipedia (English part).

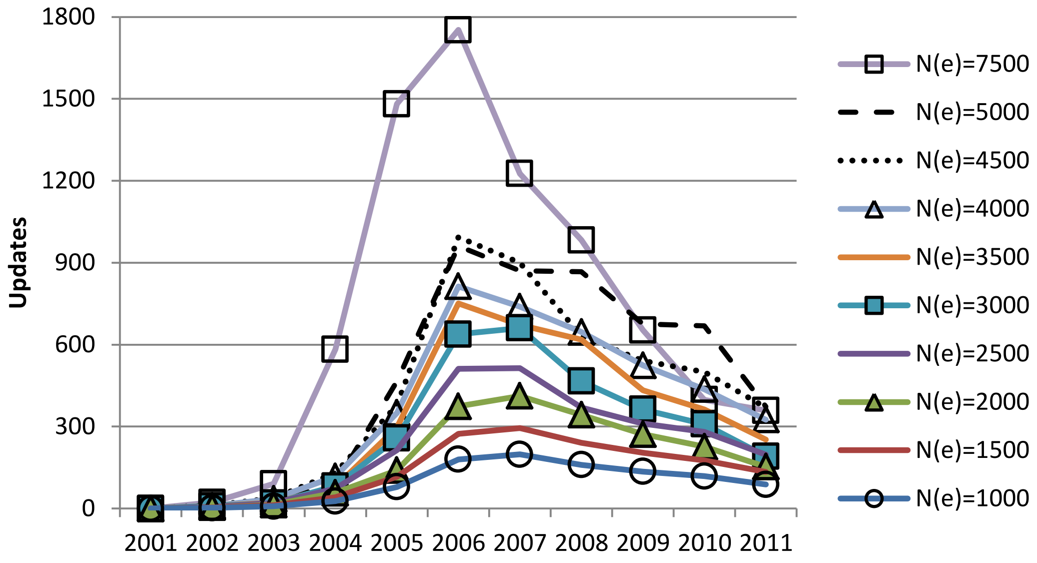


**Figure 8. Average yearly updates of the selected articles with the first revision no later than 2006-01-01.**

For the selected articles, leaving out updates prior to 2006-01-01 lets us having items with average number of revisions decreased by at most 12% of the corresponding groups' size (882 for 1000±10, etc.), except for the control group 7500±150 which got 5371 accountable updates on average.

Hence, we got a period of revisions committed for the selected articles from 2006-01-01 to 2011-12-01, which is about 71 months, or 2160 days, meaning that for the groups of articles 1000 to 7500 revisions we have on average $0.4 \leq \lambda \leq 2.5$ updates per day. This drives us in getting corresponding tick sizes we can explore for the algorithms, coupled with their valid update ranges (see below).

For the given range of frequencies of updates ($\lambda$), eq. 10 suggests use of tick sizes from about 8 to 49 hours. However, for more extensive studies of prediction accuracy of the algorithms presented in this work, we used an increased range in our experiments – from 6 to 72 hours. More specifically, we used the following values for tick sizes of 6, 12, 24, 48 and 72 hours.

For each tick size, we want to account only *valid* range of updates. As we mentioned before, such validity is driven by the empirical granularity limits. For example, for a tick size 24 hours and lifetime of about 3000 days the valid range of updates would be from 1000 to 6000. For all the tick sizes see Table 4 for the corresponding valid ranges of revisions and the ranges applicable to the groups of selected articles (1000 to 7500).

**Table 4. Validity periods for corresponding tick sizes.**

| Tick size, hours | Valid range of revisions | Valid range of revisions in the selected groups |
|---|---|---|
| 6 | 4000 - 24000 | 4000 - 7500 |
| 12 | 2000 - 12000 | 2000 - 7500 |
| 24 | 1000 - 6000 | 1000 - 5000 |
| 48 | 500 - 3000 | 1000 - 3000 |
| 72 | 350 - 2000 | 1000 - 2000 |

## 6.1 AVERAGING METHOD

Having adopted implementation of the enhanced averaging prediction method (Section 5.1) for an interval prediction (ticks), we tested accuracy of the prediction on the entire dataset (articles with revisions ranging from N(e) = 1000 to N(e) = 7500) with tick sizes from 6 to 72 hours. Leaving only valid ranges of update rates for each tick (Table 4), we got results presented in Figure 9.

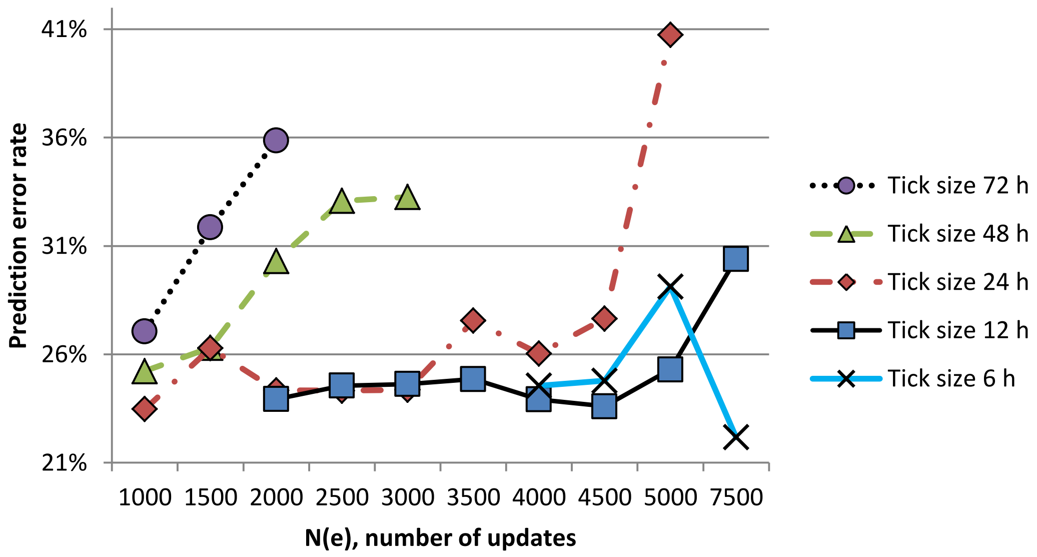


**Figure 9. Error rates of enhanced averaging method.**

As one can see from the figure, throughout the entire range of updates, the best accuracy of this method is achievable for tick sizes from 6 to 48 hours. To compare predictive accuracy of this method with the other two, we will take the best accuracy curve which includes error rate at tick size of 24 hours for articles with revisions N(e) = 1000, 48 hours for N(e) = 1500, 12 hours for N(e) = 2000, etc.

## 6.2 Shifting window

We have explored shifting window method (Section 5.2) over a wide range of values of shifting window parameter, coupled with the set of defined above tick sizes. As the range of values of shifting window parameter we took the following values: 0.5, 1, 2, 3, 6, 12 and 24 months. As we will show later, this range and granularity of step is sufficient enough to find the best prediction performance of this method to compare it with performance of others.

Initially, for each value of shifting window, we obtained prediction error rate while accounting corresponding validity intervals of each tick size (Table 4). Figure 10 represents one of such graphs for window size of 1 month.

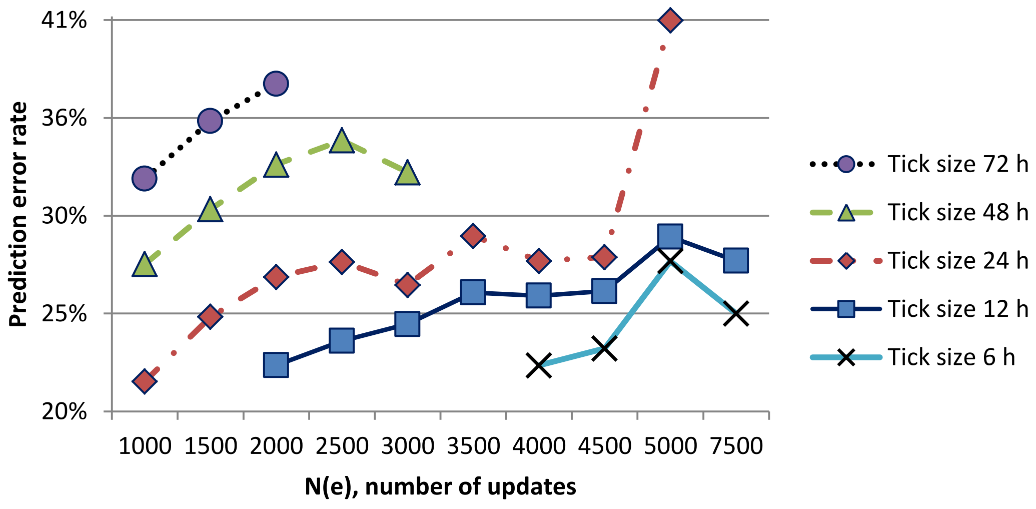


**Figure 10. Error rates of shifting window method with window size of 1 month.**

For further analysis, we need to see which tick size values give the best prediction results for this method throughout entire range of update rate groups 1000 to 7500. In fact, for all window sizes from 0.5 to 24 months, only ticks from 6 to 24 hours drove this method to the least prediction errors, as on Figure 10. Now, to compare the best accuracy among entire range of shifting window sizes for these 3 tick sizes, we will take each of those ticks and explore behavior of the method in the corresponding update rate groups.

Figure 11 represents error rates for tick size of 6 hours, for all shifting windows from 0.5 to 24 months.

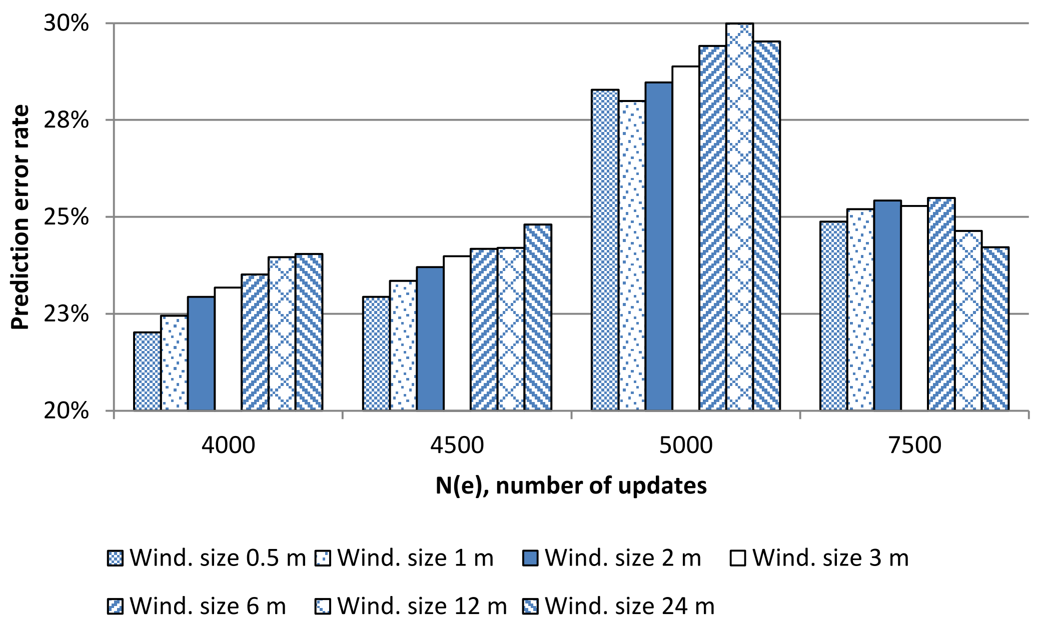


**Figure 11. Error rates of shifting window method for tick size of 6 hrs.**

As one can see, the best performance of this method for tick size of 6 hours was achieved with shifting window of sizes from 0.5 month (update rate 4000 and 4500) to 1 month (rate 5000) and 24 months (rate 7500).

Similarly, for the other two tick size parameters (12 and 24 hours) error rate is represented on Figure 12 and Figure 13 for the corresponding valid intervals of update rate groups.

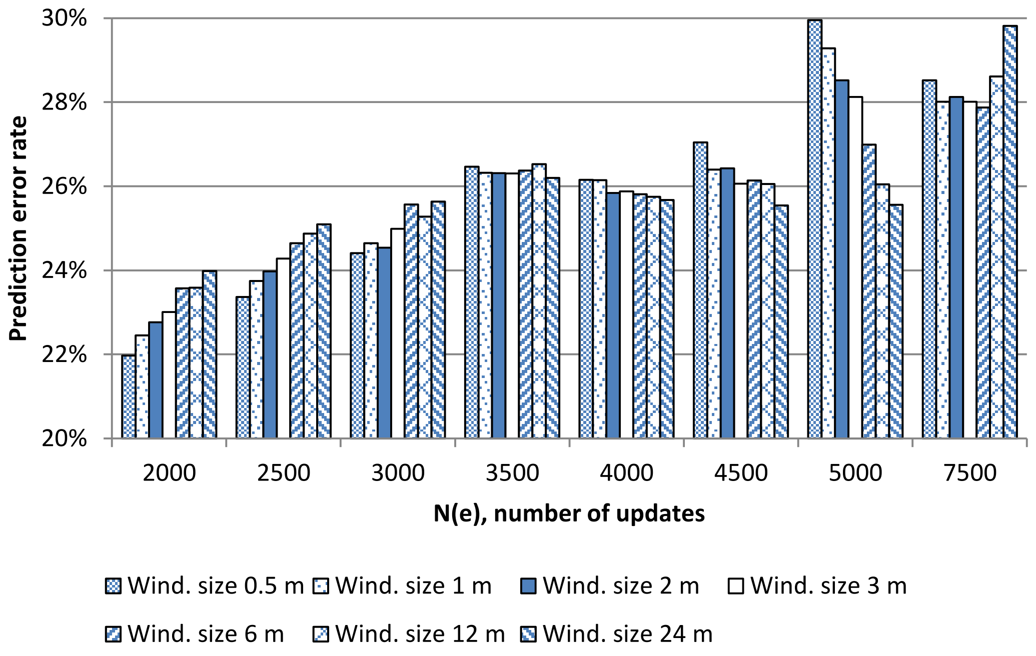


**Figure 12. Error rates of shifting window method for tick size of 12 hrs.**

For both tick sizes of 12 and 24 hours this method demonstrates the best prediction for window sizes of 0.5, 6 and 24 months.

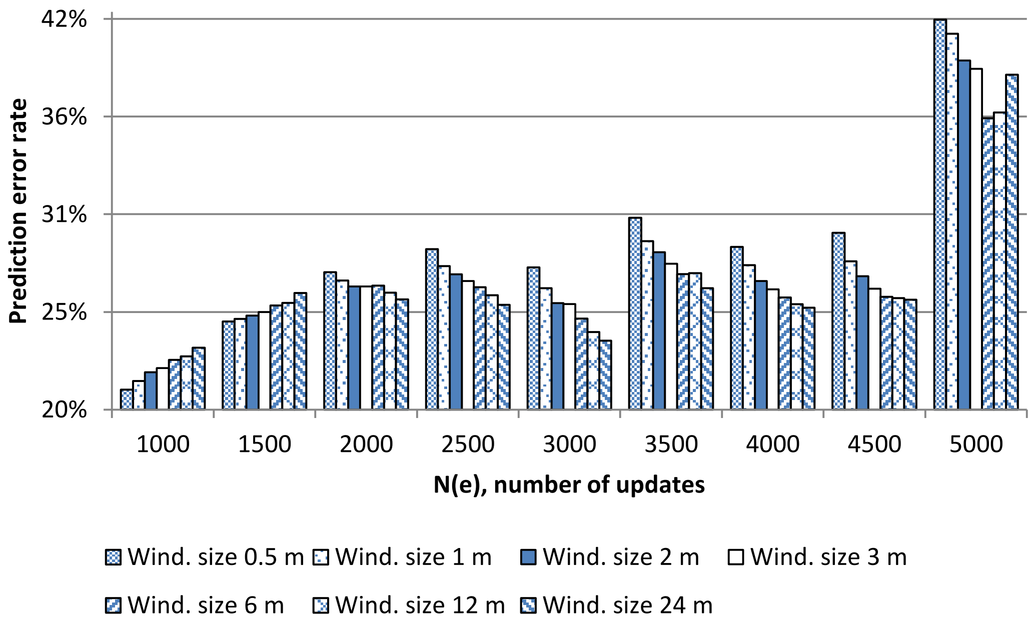


**Figure 13. Error rates of shifting window method for tick size of 24 hrs.**

We should remind here, that for a group N(e) = 5000 we have only 1 article, which may cause the outlier behavior on Figure 11 and Figure 13. However, since this is the best approximation we could have from the Wikipedia, we accounted this group for our analysis, keeping in mind its nature.

As one can see from the graphs, depending on a tick size and density of updates, this method gives the best results with a wide range of shifting window sizes. However, there is a common trend of better prediction accuracy with window size of 0.5 month for lower update rates (up to N(e) of 4500, 3000, 1500 for tick sizes of 6, 12 and 24 hours correspondently), and window size of 24 months for higher update rates.

To compare the best prediction performance of this method with the other two, we will take minimal error at each group of update rate, combining graphs for each tick size shown above, from Figure 11 to Figure 13. The results of such a comparison are given in Section 6.4.

## 6.3 Exponential smoothing

For the exponential smoothing method (Section 5.3) we will first explore a wide range of α parameter (which defines speed of gradually "forgetting" about recent updates). For this experiment, we took α = 0.01, 0.05, 0.1, 0.2, …, 0.9.

As a result, the best performance of this method was achieved with either some of the selected tick sizes (Figure 14) or all of them (Figure 15).

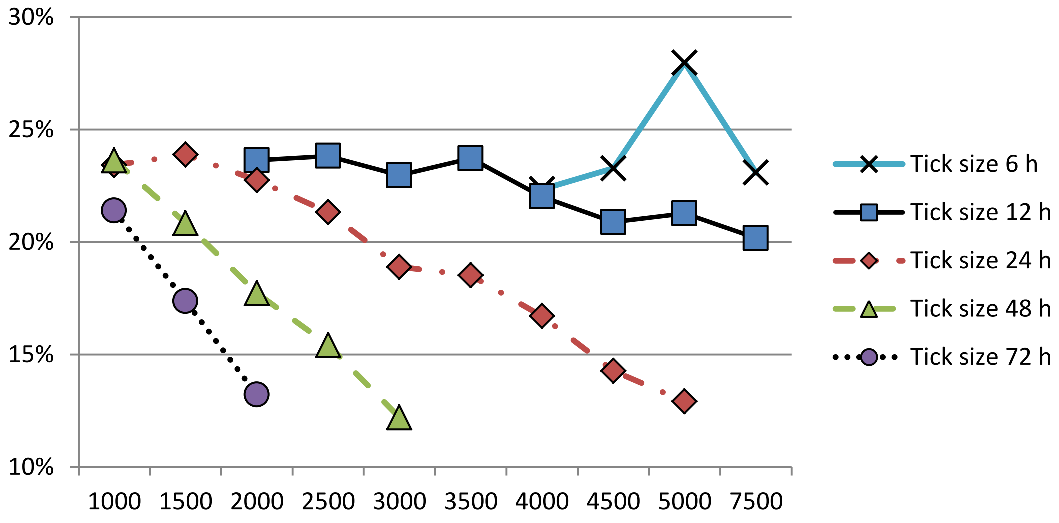


**Figure 14. Error rates of exponential smoothing method with parameter α=0.05.**

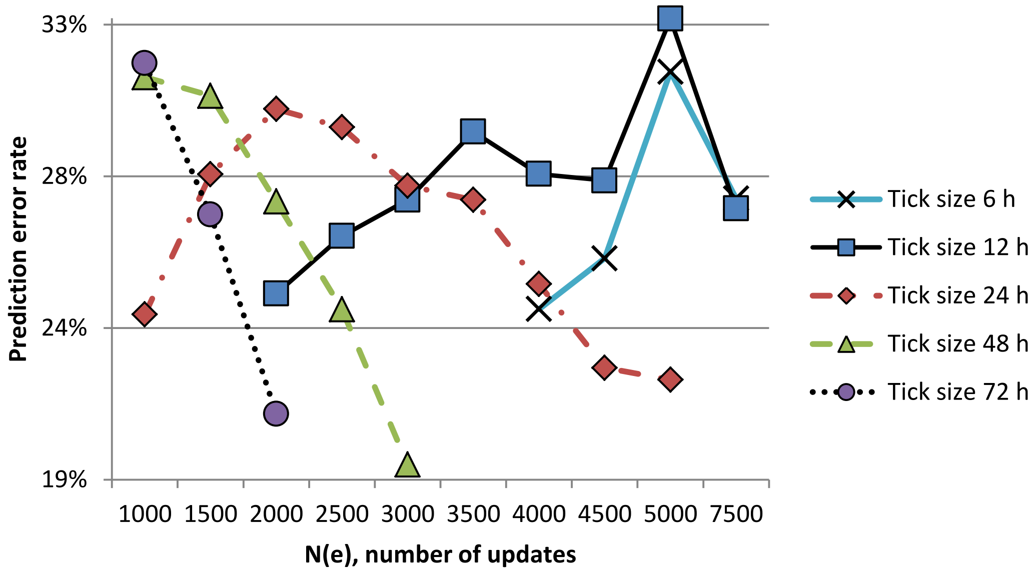


**Figure 15. Error rates of exponential smoothing method with parameter α=0.9.**

As study of the predictive accuracy of this method for each tick size revealed, only α from 0.01 to 0.5 led to the least error rates. In the following graphs we omitted error rates for α = 0.6 and higher for a better visualization of the results.

On the corresponding valid update rate interval for a tick size of 6 hours exponential smoothing demonstrated the best performance with α = 0.01 and 0.05 (Figure 16).

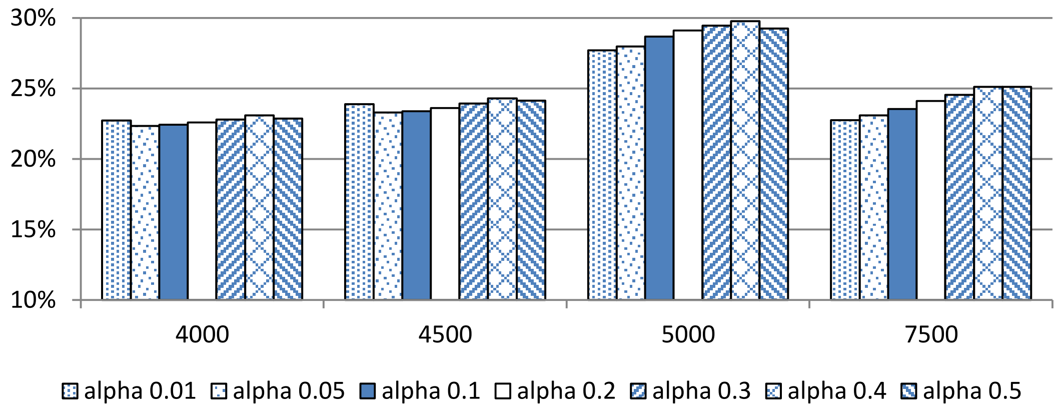


**Figure 16. Error rates of exponential smoothing method for tick size of 6 hrs.**

For the other tick sizes (from 24 to 72 hours) the best performance of this method was normally achieved for α = 0.01, 0.05 and 0.5 (Figure 17, Figure 18, Figure 19 and Figure 20). For tick size of 24 hours usage of α = 0.1 gave the least errors for group of rate 5000 as well.

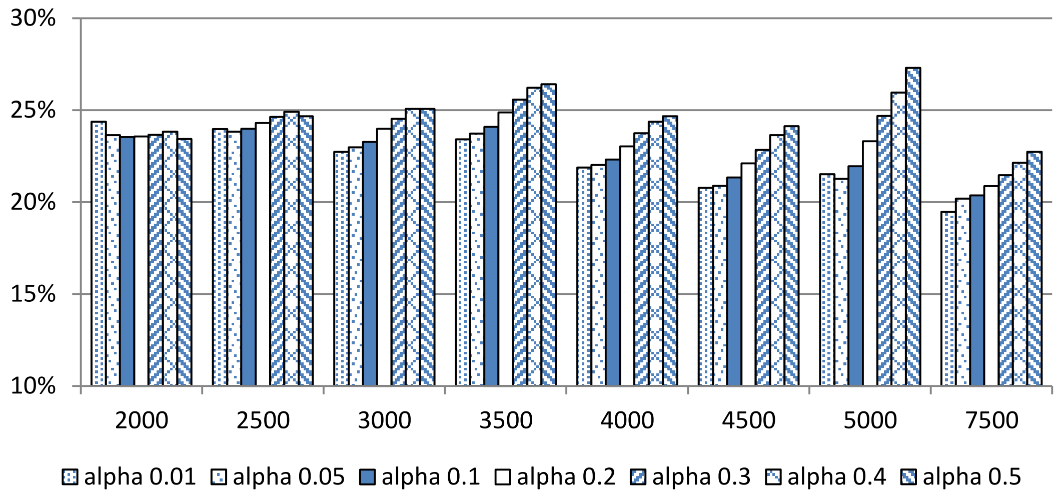


**Figure 17. Error rates of exponential smoothing method for tick size of 12 hrs.**

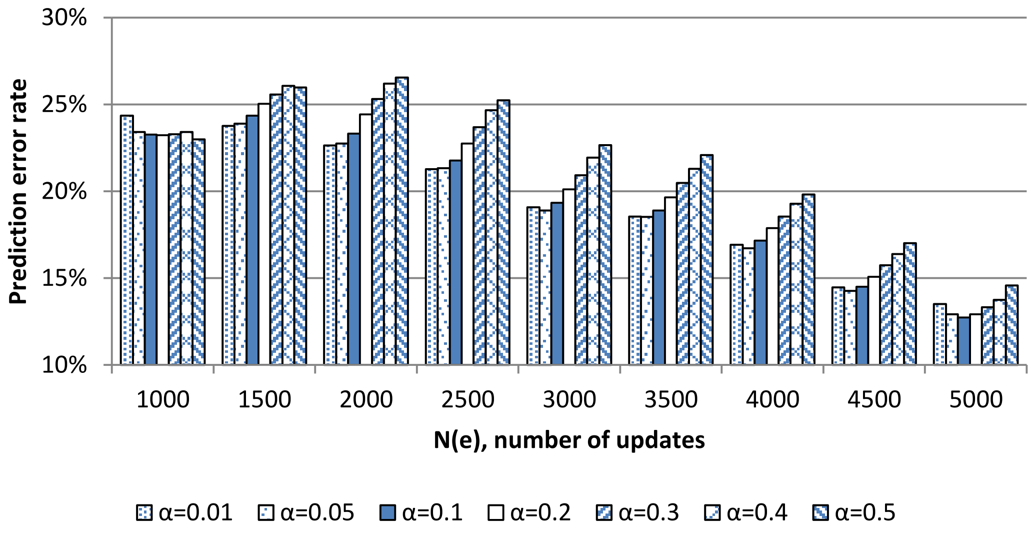


**Figure 18. Error rates of exponential smoothing method for tick size of 24 hrs.**

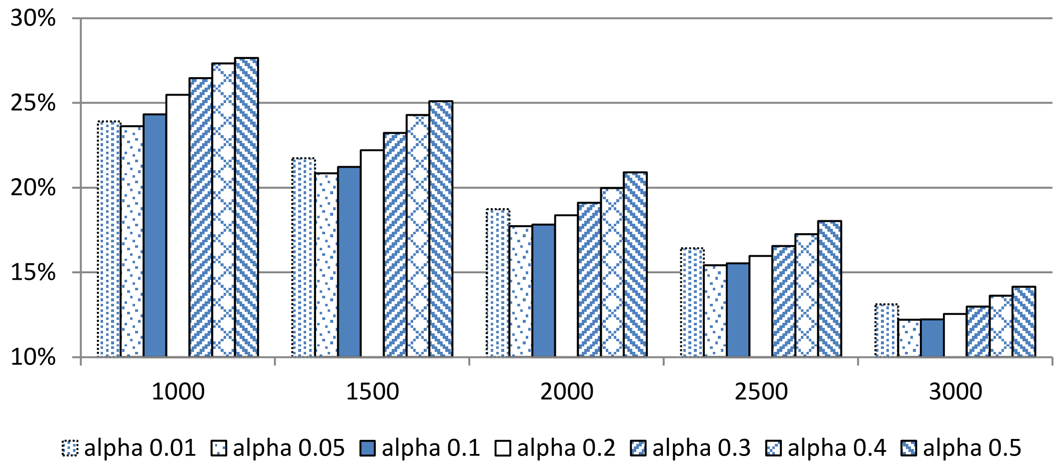


**Figure 19. Error rates of exponential smoothing method for tick size of 48 hrs.**

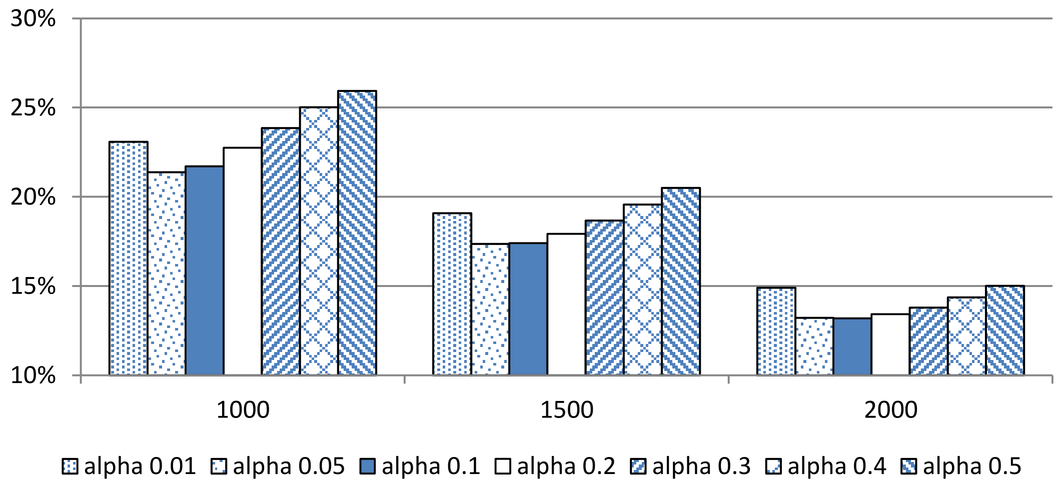


**Figure 20. Error rates of exponential smoothing method for tick size of 72 hrs.**

Having identified parameters for the best prediction accuracy of exponential smoothing (depending on update rate group and a tick size used), we want to compare the best possible prediction accuracy of this method with the other two, combining the lowest error rate of this method at each update frequency group (Figure 16 to Figure 20).

## 6.4 EXPERIMENTAL RESULTS COMPARISON

To compare predictive accuracy of the three algorithms presented in this work, we took the lowest boundary of error rate of each of them, explored with range of possible values of parameters specific for the corresponding algorithm (tick size, size of shifting window and α coefficient). As a result, for each algorithm we got the best predictive accuracy for every update rate group of the selected range – from 1000 to 7500 (revisions per lifecycle). Comparing these results (Figure 21), we can see that throughout the entire range of revision frequencies, except for group N(e) = 1000, exponential smoothing method gives the best prediction results. However, outperformance of enhanced averaging methods over shifting window on some groups of articles (3500, 5000 and 7500) is an interesting observation which may result from the exhaustive trend of articles at higher update rate on Wikipedia, and as a result, intolerance for possible specifics of revision history trend of a few articles in an update group.

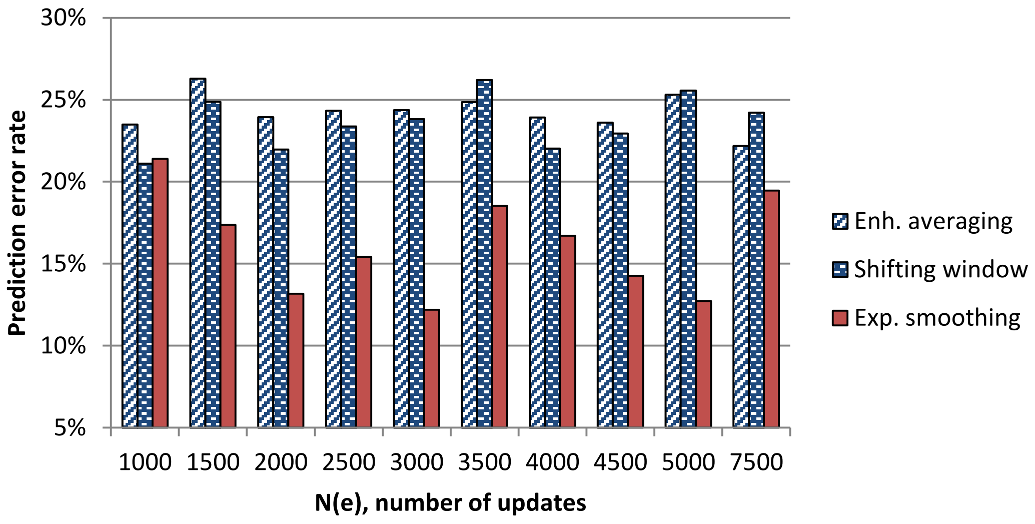


**Figure 21. Algorithmic minimal error rates for the selected.**

Average error rates by the corresponding prediction methods – enhanced averaging, shifting window and exponential smoothing – were observed as 24.2%, 23.6% and 16.1% correspondently, which is expected from design of the presented algorithms.

## 6.5 SUMMARY OF EXPERIMENTAL EVALUATION

Requiring the same number of variables for representation of update history of each data element (article from Wikipedia), exponential smoothing method predicts each next update (for N(e) > 1000 during about 6 years) more accurately than the averaging methods (enhanced averaging and shifting window) adopted for interval prediction. This fact is expected and can be explained by recency of significant updates for the former method, which allows accounting both global trends and local outliers.

The overall trend of exponential smoothing method to predict updates more accurately is due to its ability to gradually account them; from the other side, this is because of memoryless nature of the latter (averaging) methods – they neither store distribution of previous updates nor learn how they evolve.

## 7. CONCLUSIONS

In this paper, we studied nature of a data-driven notion of staleness, based on current state of the art and existing requirements for data quality metrics. After setting a proper notion, we described algorithms for data staleness measurement that are suitable for an information system acquiring data with aperiodic updates of high frequency (about $0.4 < \lambda < 2.5$ updates per day).

In particular, we presented an efficient (in terms of memory and CPU resources, and compared to naïve averaging approaches) exponential smoothing method to measure data staleness. Depending on accuracy of required prediction and available resources, one can expand this approach to consider trends, outliers, seasonality and other factors required for a given task.

In our experiments, we have explored prediction accuracy of this method, comparing it with the averaging algorithms presented in this work as well. The results demonstrate the best achievable prediction accuracy for articles of various update frequency and the corresponding parameters required for that. As a consequence, these results can serve as a basis for selection of a required method for prediction of updates, and hence, measuring a time related data-driven characteristic of staleness. In particular, for stochastically changing and frequently updatable web data, represented by revision history of selected articles from Wikipedia, exponential smoothing method presented in this work gives a basis for development of a staleness measurement algorithm for a custom application. Alternatively, it can serve as a ready-to-go approach.

Having staleness measurement mechanism of data at hand, one can either communicate the staleness level of data elements, or set necessary synchronization techniques with external data sources in such a way that own data would satisfy requirements imposed by a time-related quality dimension identified for an application.

# REFERENCES

[1] Paolo Bouquet, Themis Palpanas, Heiko Stoermer, and Massimiliano Vignolo, "A Conceptual Model for a Web-scale Entity Name System," in *ASWC*, Shanghai, China, 2009.

[2] Paolo Bouquet, Heiko Stoermer, and Barbara Bazzanella, "An Entity Name System (ENS) for the Semantic Web," in *ESWC*, 2008.

[3] Barbara Bazzanella, Junaid Ahsenali Chaudhry, Themis Palpanas, and Heiko Stoermer, "Towards a General Entity Representation Model," in *SWAP*, Rome, Italy, 2008.

[4] A. Segev and W. Fang, "Currency-based updates to distributed materialized views," in *ICDE*, 1990.

[5] R.Y. Wang and D.M. Strong, "Beyond accuracy: what data quality means to data consumers," *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5-33, 1996.

[6] Felix Naumann and Claudia Rolker, "Assessment Methods for information quality criteria," in *Proceedings of the International Conference on Information Quality*, 2000.

[7] Y.W. Lee, D.M. Strong, B.K. Kahn, and R.Y. Wang, "AIMQ: a methodology for information quality assessment," *Information and Management*, no. 40, pp. 133-146, 2002.

[8] Mokrane Bouzeghoub and Verónika Peralta, "A framework for analysis of data freshness," in *IQIS*, 2004.

[9] B. Heinrich, M. Klier, and M. Kaiser, "A Procedure to Develop Metrics for Currency and its Application in CRM," *ACM Journal of Data and Information Quality*, vol. 1, no. 1, June 2009.

[10] F. Naumann, U. Leser, and J.C. Freytag, "Quality-driven integration of heterogeneous information systems," in *VLDB*, 1999.

[11] U. Röhm, K. Böhm, H.J. Schek, and H. Schuldt, "FAS: a freshness-sensitive coordination middleware for a cluster of OLAP components," in *VLDB*, 2002.

[12] Hongfei Guo, Per-Åke Larson, Raghu Ramakrishnan, and Jonathan Goldstein, "Relaxed Currency and Consistency: How to Say “Good Enough” in SQL," in *SIGMOD*, 2004.

[13] A. Labrinidis and N. Roussopoulos, "Exploring the tradeoff between performance and data freshness in database-driven web servers," *VLDB Journal*, vol. 13, no. 4, pp. 240-255, 2004.

[14] Huiming Qu and Alexandros Labrinidis, "Preference-Aware Query and Update Scheduling in Web-databases," in *ICDE*, 2007.

[15] P.A. Bernstein, A. Fekete, H. Guo, R. Ramakrishnan, and P. Tamma, "Relaxed Currency Serializability for Middle-Tier Caching and Replication," in *SIGMOD*, 2006.

[16] L. Golab, T. Johnson, and V. Shkapenyuk, "Scheduling updates in a real-time stream warehouse," in *ICDE*, 2009.

[17] Hongfei Guo, Per-Åke Larson, and Raghu Ramakrishnan, "Caching with “Good Enough” Currency, Consistency, and Completeness," in *VLDB*, 2005.

[18] Junghoo Cho and Hector Garcia-Molina, "The evolution of the web and implications for an incremental crawler," in *VLDB*, 2000.

[19] M. Xiong, S. Han, K.-Y. Lam, and D. Chen, "Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results," in *26th IEEE International Real-Time Systems Symposium*, 2005.

[20] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez, "Data Currency in Replicated DHTs," in *SIGMOD*, 2007.

[21] Fuat Akal et al., "Fine-grained replication and scheduling with freshness and correctness guarantees," in

*VLDB*, 2005.

[22] Min Xiey, Haixun Wangz, Jian Yinz, and Xiaofeng Meng, "Providing Freshness Guarantees for Outsourced Databases," in *EDBT*, 2008.

[23] Junghoo Cho and Hector Garcia-Molina, "Synchronizing a database to Improve Freshness," in *SIGMOD*, 2000.

[24] Junghoo Cho and Hector Garcia-Molina, "Effective page refresh policies for Web crawlers," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, December 2003.

[25] L. Pipino, Y.W. Lee, and R.Y. Wang, "Data quality assessment," *Comm. ACM*, vol. 45, no. 4, pp. 211-218, 2002.

[26] *Collins English Dictionary*, 8th ed., 2006.

[27] Adir Even and G. Shankaranarayanan, "Utility-Driven Assessment of Data Quality," *The Database for Advances in Information Systems*, vol. 38, no. 2, pp. 75-93, May 2007.

[28] Everette Jr. Gardner, "Exponential smoothing: The state of the art — Part II," *International Journal of Forecasting*, no. 22, p. 637– 666, 2006.

[29] D. Ballou, R. Wang, H. Pazer, and G. Tayi, "Modelling Information Manufacturing Systems to Determine Information Product Quality," *Management Science*, vol. 44, no. 4, 1998.

[30] R.Y. Wang, M.P. Reddy, and H.B. Kon, "Toward quality data: an attribute-based approach," *Decision Support Systems*, vol. 13, 1995.

[31] Guzmán Llambías, Regina Motz, Federico Toledo, and Simon de Uvarow, "Learning to Get the Value of Quality from Web Data," *Lecture Notes in Computer Science*, vol. 5333, no. 2008, 2008.

[32] M. Bovee, R.P. Srivastava, and B. Mak, "A conceptual framework and belief-function approach to assessing overall information quality," *International Journal of Intelligent Systems*, vol. 18, no. 1, 2003.

[33] R. Wang and D. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of management information systems*, vol. 12, no. 4, 1996.

[34] C. Cappiello, C. Francalanci, and B. Pernici, "Time-related factors of data quality in multichannel information systems," *Journal of Management Information Systems*, vol. 20, no. 3, 2003.

[35] A. Even and G. Shankaranarayanan, "Utility-driven assessment of data quality," *The Database for Advances in Information Systems*, vol. 38, no. 2, 2007.

[36] H. Hinrichs, "Datenqualitätsmanagement in Data Warehouse-Systemen," Oldenburg, doctoral thesis 2002.

[37] B. Heinrich, M. Kaiser, and M. Klier, "How to measure data quality? – a metric based approach," in *International Conference on Information Systems (ICIS*, 2007.

[38] B. Heinrich and M. Klier, "Assessing data currency — a probabilistic approach," *Journal of Information Science*, vol. 37, no. 1, 2011.

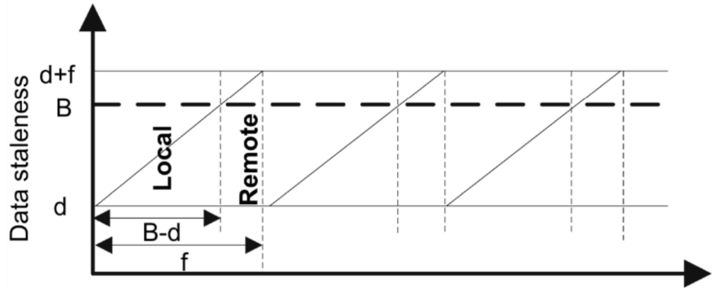
[39] Y. Wand and R.Y. Wang, "Anchoring data quality dimensions in ontological foundations," *Comm. ACM*, vol. 39, no. 11, 1996.

[40] G. Shankaranarayan, M. Ziad, and R.Y. Wang, "Managing data quality in dynamic decision environments: An information product approach," *Journal of Database Management*, vol. 14, no. 4, 2003.

# APPENDIX 1. TIME-RELATED NOTIONS, DEFINITIONS AND MEASUREMENT METHODS FROM THE RELATED RESEARCH WORKS

| **Notion, scope (System or/and Data)** | **Definition** | **Measurement** |
|---|---|---|
| Freshness (up-to-dateness) (S) | A state of correspondence of data in a replica (web cache) to the data in the source [18] | "freshness represents the fraction of up-to-date pages in the local collection" |
| Freshness (staleness, lateness, tardiness) (S) | "*freshness* of a data stream is the maximum timestamp of any of its tuples that have arrived at the warehouse by time t." "*data staleness* – the difference between the current time and the timestamp of the most recent tuple in a table." "(system) *lateness* (or *tardiness*) – the difference between the completion times of late tasks and their deadlines" [16] | $S_T = t - freshness(T, t)$ where $freshness(T, t)$ is the maximum timestamp of any tuple in table $T$ at time *t* |
| Freshness (up-to-dateness) (S) | Freshness (up-to-dateness) is a state of correspondence of data in a replica to the data in the source table [15], [21] | N/A: the work [15] presents a method of usage of existing freshness constraints in user queries; in [21] freshness requirements are specified by users |
| Freshness (up-to-dateness) (S) | Freshness: "the outsourced data is up-to-date, that is, all database update operations have been correctly carried out by the service provider" [22] | N/A: the work presents a method to ensure correctness of update operations made by outsources database |
| Freshness (staleness, up-to-dateness) (S) | Availability of updates for a replicated web database makes its fresh data stale (or out-of-date), until completion of application of the updates [14] | "staleness can be measured by the number of unapplied updates, as well as the time differential or value distance between the current and the most up-to-date data items" |
| Freshness (staleness) (S) | "(a data object) is stale when an update for it has arrived, but not yet executed" [13] | "AWebView (a fragment of a web page) $W_i$ is stale, if $W_i$ is materialized and has been invalidated, or if $W_i$ is not materialized and there exists a pending update for a parent relation of $W_i$. A WebView $W_i$ is fresh, otherwise." $f(W_i, t_k) = \begin{cases} 1, & if\ W_i\ is\ fresh\ at\ time\ t_k \\ 0, & if\ W_i\ is\ stale\ at\ time\ t_k \end{cases}$ |
| Freshness (temporal validity) (S) | "A real-time data object is *fresh* (or *temporally valid*) if its value truly reflects the current status of the corresponding entity in the system environment" [19] | N/A: given data validity intervals (which are "defined based on the dynamic properties of the data object" and they are also "application-dependent"), the work demonstrate a deferrable algorithm for a processor scheduling updates from sensors. |
| Freshness index (up-to-dateness) (S/D) | "reflects how much the data has deviated from the up-to-date version" [11] | N/A: "freshness index reflects how much the data has deviated from the up-to-date version. Intuitively, a freshness index of 1 means that the data is up-to-date, while an index of 0 tells us that the data is 'infinitely' outdated" |

| | | |
|---|---|---|
| Delay freshness index<br>(S/D) | “reflects how late a certain cluster node is as compared to the up-to-date OLTP node” [11] | $f(c) = \frac{\tau(c)}{\tau(c_0)}$<br>where $\tau(c)$ is the commit time of the last propagated update transaction on an OLAP node $c$, $\tau(c_0)$ is the commit time of the most recent update transaction on the OLTP node |
| Freshness<br>(age)<br>(S/D) | *Freshness, age, up-to-dateness*: a state of correspondence of data in a replica to the data in the source (real-world counterparts).<br>*Freshness* is the fraction of the local database that is up-to-date.<br>[23], [24] | Freshness of a database *S* at time *t*:<br>$F(S;t) = \frac{M}{N}$<br>where *M* is up-to-date elements, and *N* is total number of element in the database.<br>Freshness of a data element $e_i$ at time *t*:<br>$F(e_i,t) = \begin{cases} 1, & if\ e_i\ is\ fresh\ at\ time\ t \\ 0, & otherwise \end{cases}$<br>Age of a data element $e_i$ at time *t*:<br>$A(e_i,t) = \begin{cases} 0, & if\ e_i\ is\ fresh\ at\ time\ t \\ t - modification\ time\ of\ e_i, & otherwise \end{cases}$ |
| Volatility/shelf life<br>(D) | “How long the item remains valid” [29], [30] | N/A: “a quality parameter value [e.g., volatility] is the value determined by the user” |
| Web data Freshness<br>(age, volatility)<br>(S) | “freshness measures how much updated are data for a specific task” [31] | Freshness = max{0, 1-(age/volatility)}<br>“age suggests how old are data, captures the time interval between the creation or actualization of data and the time at what user receives data; volatility measures the frequency with which data change over time” |
| Age<br>(D) | “the time difference between when the real-world event occurred and when the data was entered” [29] | Age is the time difference between when the real-world event occurred and when the data was entered |
| Age<br>(D) | “*Age* measures how long ago information was recorded” [32] | Age is the time difference between when data was recorded and instant of measurement |
| Currency<br>(age)<br>(S) | “refers to the age of the primitive data units used to produce the information products” [29] | “The currency measure is a function of several factors: when the information product is delivered to the customer (Delivery Time); when the data unit is obtained (Input Time); and how old the data unit is when received (Age).”<br>Currency = (Delivery Time - Input Time) + Age<br>(Delivery Time - Input Time) represents how long the data have been in the system<br>Age is the time difference between when the real-world event occurred and when the data was entered |
| Currency<br>(up-to-dateness)<br>(D) | "it is easy to tell if the data are updated" [33] | N/A |
| Currency level<br>(S) | “Currency level is defined as the degree to which data are up-to-date in a given operational database” [34] | $currency\_level_{ij} = 1 - out\_of\_date_{ij}$<br>is a currency level of the i-th functionality of the j-th channel [of a multichannel information system] |
| Currency<br>(D) | “when the data item was stored in the database” [30] | N/A: “a quality parameter value is the value determined by the user” |
| Currency<br>(age)<br>(D) | “Currency measures the degree to which the data is recent and up to date.”<br><br>More specifically, currency is: | Currency measurement reflects the *age*, the time lag between present time and the last update of the data item. |

| | | |
|---|---|---|
| | "The age of data items – the time lag between last update and present time" [*Data Characteristics Observed*];<br>"The extent to which the data items in the dataset are recent" [*Impartial Interpretation*];<br>"The extent to which outdated data damages utility" [*Contextual Interpretation*] [35] | |
| Currency (staleness, up-to-dateness) (S) | A state of correspondence of data in a replica to the data in the source table [12] | Currency, staleness: linearly growing function with lower limit as *d*, an update propagation delay, and upper limit as *d+f*, where *f* is an update propagation interval. Authors show how a boundary on currency *B* (given by user) one can include into SQL queries, enforcing execution of queries on local replica or remote source table. Figure below "Synch cycle and data currency" taken from the paper demonstrates this:<br> |
| Currency (up-to-dateness) (S) | A state of correspondence of data in a replica to the data in the source table [17] | Currency of a copy of a database snapshot is measured by its staleness value, i.e., by the time elapsed from the instant when the source got update(s) which has not been propagated to the copy, to the instant of measurement. |
| Currency (freshness, staleness) (S) | Notions of *currency, freshness, staleness* relate to a state of correspondence of data in a replica to the data in the source;<br>*current* replica is such a replica with the latest updates [20] | N/A: the scope of the paper is getting replicas with the latest updates rather than measuring their currency or freshness metrics |
| Timeliness (age) (D) | "The metric for timeliness shall deliver an indication (not a verified statement under certainty) whether an attribute value has changed in the real world since its acquisition and storage within the system or not." [36], [37], [38] | $$Timeliness = \frac{1}{(mean\ attribute\ update\ time) \cdot (attribute\ age) + 1}$$ "This quotient serves as a metric, which quantifies if the current attribute value is outdated."<br>"…if the *mean attribute update time* is 0 (i.e. the attribute value never becomes out of date), timeliness is 1 (attribute value is up-to-date). If on the other hand *attribute age* is 0 (i.e. the attribute value is acquired at the instant of quantifying DQ) we get the same result. For higher values of *mean attribute update time* or *attribute age* the result of the metric approaches 0. I.e., that the (positive) indication (the attribute value is still corresponding to its real world counterpart) decreases." [36]<br><br>$$Q_{Time}(w, A) \coloneqq \exp(-decline(A) \cdot age(w, A))$$ "$Q_{Time}(w, A)$ denotes the probability that the attribute value is still valid." [37], [38]<br>(*age*(*w*, *A*) denotes the age of the attribute [A] with value *w*, which is computed by means of two factors: the instant when DQ is quantified and the instant of data acquisition; *decline*(*A*) of attribute *A*'s values can be determined statistically) |

| | | |
|---|---|---|
| Timeliness<br>(currency,<br>volatility)<br>(D/S) | "… timeliness of an information product is dependent upon when the information product is delivered to the customer" [29] | Timeliness = {max[(l-currency/volatility), 0]}$^{S}$<br>The exponent s is a parameter that allows us to control the sensitivity of timeliness to the currency-volatility ratio. Timeliness ranges from 0 ("the data is unacceptable from the timelines viewpoint") to 1 ("the data meets the most string timeliness standard") |
| Timeliness<br>(currency,<br>volatility)<br>(D/S) | "can be characterized by *currency* (when the data item was stored in the database) and *volatility* (how long the item remains valid)" [30] | N/A |
| Timeliness<br>(S) | "extent to which the age of the data is appropriate for the task at hand" [10] | N/A: timeliness is measured based on update information that is provided by the information source |
| Timeliness<br>(D) | "In our model, timelines refers only to the delay between a change of the real-world state and the resulting modification of the information system state" [39] | N/A: authors provide ontological foundations to further measure data quality dimensions |
| Time-to-Deliver<br>(S) | "The time-to-deliver an IP [information product] (or any component data) is defined as the time to completely generate the IP from any processing stage in the IPMAP [a set of modeling constructs to systematically represent the manufacture of an IP]" [40] | Time-to-Delivery = $\sum_{i=1,n} t_i$ (a.1)<br><br>Expected mean time at stage x $(ET_x) = (4 \cdot m_x + a_x + b_x)/6$ (a.2)<br>Variance in time at stage x $(\sigma^2_{<x>}) = [(b_x - a_x)^2]/36$ (a.3)<br>Probability (completing stage x) = $(ET_x - T)/(\sum \sigma_{<Critical\ Path\ to\ x>})$ (a.4)<br><br>Where $m_x$ is mean time at stage x; $a_x$, $b_x$, are the optimistic and pessimistic time estimates at stage x. These time estimates are assumed to follow a Beta distribution. The expected mean time and variance at stage x are computed based on the Beta distribution using equations a.2 and a.3. The normalized probability is specified by equation a.4. |
| Timeliness<br>(age)<br>(D) | "The extent to which the age of the data is appropriate for the task at hand." [33] | N/A |
| Timeliness<br>(relevancy,<br>availability)<br>(D/S) | "An additional [*to the notion of timeliness in* [30]] meaning of timeliness is whether information, relevant or not, was available in time to be useful" [32] | N/A |