Automatic Document Timestamping

Kristoffer Berg Gumpen
Øyvind Nygard

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Kjetil Nørvåg, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

When searching for information, the temporal dimension of the results is an important factor regarding the information quality. Using temporal intent as a condition when searching for information is a field that is gaining increasing interest. When we search for information on search engines, such as Google, we have the option to use time of creation as a part of the search criteria. Unfortunately, when searching on the web we have no guarantee that the timestamps for the results corresponds to the actual date the content was created. Since the timestamps provided on the Internet can not be trusted it would be of great use if there existed a method for timestamping documents without knowing the actual date of creation. In this thesis, we have presented and implemented some existing approaches to this problem, modified them and added some parameters for tweaking and fine tuning the results. These approaches are so called content based approaches, and they use statistical analysis on the textual contents of documents in a collection in order to predict a document's time of origin. In order to evaluate our implementation, we have performed extensive experiments and compared our results with results achieved in earlier research.

Sammendrag

Når man søker etter informasjon vil den temporale dimensjonen av resultatene være en viktig faktor med hensyn til informasjonskvaliteten. Å bruke temporalitet når man søker etter informasjon er et felt hvor interessen vokser. Dersom man søker etter informasjon på søkemotorer, som Google, kan man velge å bruke opprinnelsesdato som en del av søkekriteriet. Uheldigvis, når man søker på internett har man ingen garanti for at datostemplingen for resultatene korresponderer med den faktiske datoen innholdet ble opprettet. Grunnet det faktum at denne datostemplingen ikke er til å stole på ville det vært til stor hjelp dersom det fantes en metode for å tidsbestemme dokumenter uten å vite deres faktiske opprinnelsesdato. I denne oppgaven har vi presentert og implementert noen allerede eksisterende metoder for å løse dette problemet, gjort endringer på dem og lagt til parametere for å finjustere sluttresultatet. Disse metodene er såkalte innholdsbaserte metoder, og bruker statistisk analyse på det skriftlige innholdet av dokumenter i en samling for å predikere opprinnelsestiden for et dokument. For å kunne evaluere vår implementasjon har vi utført omfattende eksperimenter og sammenlignet våre resultat med allerede oppnådde resultat fra tidligere studier.

Preface

This thesis is written by Kristoffer Gumpen and Øyvind Nygard, and is the product of our final year at the Norwegian University of Science and Technology with specialization in Databases and Search at the Department of Computer and Information Science.

We would like to gratefully thank our supervisor and Professor Kjetil Nørvåg for his contributions and for being available and helpful. He has given us constructive feedback and provided invaluable guidance during the period of writing this thesis.

Table of Contents

Chapter 1

Introduction

In this chapter we will give an introduction to our thesis. Section 1.1 explains the motivation behind our work, Section 1.2 provides a definition of the problem we are trying to solve, Section 1.3 defines the research questions we are trying to answer, Section 1.4 summarizes our contribution to the field of research and Section 1.5 outlines the structure of our report.

1.1 Motivation

In the area of information retrieval, taking the temporal dimension into account when searching for documents is a field that is of increasing interest and importance. In [24], Metzger identifies the timeliness of search results as one of five key aspects when it comes to determining a document's quality. Currently, search engines such as Google provides the user with the possibility of specifying which time they want documents from. Unfortunately the user will have no guarantee that the resulting documents' timestamps corresponds to the actual time the documents were written. This is because the timestamps are collected from the documents' meta data which is often corrupted as a result of the size and dynamic nature of the web. These timestamps could be the date that the content was uploaded from another source to the website, the date for the last time the website article was updated and so on. This gives a need for alternative, more sophisticated approaches for determining the age of a document. Under the assumption that documents describing the same events have lexical similarities and in most cases originate from the same time period [29], several researchers have tried to come up with methods that analyze textual similarities in documents with the purpose of determining a document's age. These sort of methods are referred to as content-based approaches. There is yet to be seen a content-based approach for document timestamping that is precise enough and scales well enough to be applied on web search engines. If such a method were to exist, it could be used to automatically detect a query's temporal intent as well as ensuring that returned documents originate from the desired point in time.

1.2 Problem Formulation

The problem presented in this thesis is the problem of incorrectness and ambiguity of temporal information in meta data describing documents. This problem is of importance for anyone who uses information retrieval systems. If this problem was to be resolved it could ensure higher relevance in search results, because it would be possible to predict the temporal intent of the query as well as the time of origins from the result documents. More specifically, this paper will explore and evaluate existing approaches for timestamping documents by looking at *term burstiness* [17]. Here, a *bursty* interval for a term is a time period where the term exhibits atypically high frequencies. The underlying assumption for such a method to work is that as events change, the text used to describe documents will change as well [29]. This means that the terms used to describe a given event should be simultaneously bursty in the time frame around the event's time of origin. This assumption applies particularly for news document collections, since news documents mostly describe current events. In addition the paper will explore a method that solely relies on lexical similarity between documents as well as different techniques for preprocessing the documents to be used for dating. Finally we are going to explore how the accuracy of the dating algorithm can be increased by adding a threshold for dating the documents.

1.3 Research Questions

From the problem definition, this thesis will answer the following research questions:

RQ1 : What impact does the preprocessing of the dataset have on the final result when timestamping different documents?

RQ2 : When deciding the timestamp for a document by comparing it to similar documents, how will using different weighting methods for the reference documents alter the results?

RQ3 : What will be the optimal number of documents to use for comparing when deciding a timestamp for the query document?

RQ4 : Can we achieve a fair trade off between accuracy and the fraction of the dataset estimated by introducing a threshold?

To be able to answer these research questions we have implemented the *burstySimDater* described in [14]. We have modified it so that the program is able to receive different kinds of input and parameters which can be used as comparison to the original algorithm. We have run several experiments with both different datasets as well as the different parameters we wanted to explore. The results have been evaluated, discussed and compared, both with earlier work and with each other.

1.4 Contributions

The contributions of our project can be summarized as follows:

- Our own implementation of the the *burstySimDater* method [14].
- Our own dating method, that slightly outperforms the *burstySimDater* method [14] in terms of accuracy.
- An extensive experimental evaluation on how different preprocessing techniques affect the result of both the the *burstySimDater* method and our own approach. Additionally we experiment on the influence of other parameters regarding these methods (Chapter 6.2.1).
- Experiments with different thresholds for estimating the timestamp of a document and how it affects the number of documents estimated.

1.5 Project Outline

The remainder of our thesis is organized as follows:

Chapter 2 - Related Work: Presents the underlying theory behind the related work regarding our thesis.

Chapter 3 - Preliminaries: Provides an explanation of the various concepts and techniques used in the implementation.

Chapter 4 - Document Dating: Describes different content-based dating methods in detail.

Chapter 5 - Approach: Explains our approach to implementing the document dating system.

Chapter 6 - Experimental Setup: Introduces the data sets that we used for testing and describes the different parameters/factors tested as well as describing the setup for our experiments.

Chapter 7 - Results and Evaluation: Evaluates and discusses how different factors affected the results.

Chapter 8 - Conclusion and Future Work: Provides a concluding summary of the thesis. Additionally, this chapter describes some propositions regarding further work.

Chapter 2

Related Work

In this chapter we will provide a survey on previous work that are of significance regarding our project. This includes previous work done on temporal laguage models, which is described in Section 2.1, and burstiness, which is described in Section 2.2 as well as an explanation of work done on content based dating algorithms given in Section 2.3.

2.1 Temporal Language Models

In [29], Shaparenko et al. make the following assumption: ”There is an implicit assumption that as real-world events change, the text used in the documents will change as well”. Under this assumption there has been proposed several approaches to modeling the trends of term usage over time. Among the first to investigate which kind of statistical models are appropriate for modeling the temporal dimension of term usage we have Swan and Jensen [31]. They presented a system named TimeMines that automatically selects the most significant topics from a time-tagged news corpus. They achieved this by using simple contingency tables to determine if a term is significant for a given time. A term is here defined as significant if it occurs noticeably more frequent at a given time. Later, studies by Li and Croft [20] and Diaz and Jones [4] explore the use of statistical language models for modeling the temporal dimension of term usage. Li and Croft experimented with a number of TREC ad-hoc queries, analyzing the relationship between time and relevance. During this experiment they observed that documents that are relevant for a specific query tend to originate from the same time frame. Diaz and Jones [4] made similar observations when they experimented with using temporal profiles of queries in order to improve prediction of precision. By using the temporal features of a query, they indeed managed to improve prediction of the query's average precision. The findings by Diaz and Jones [4] and Li and Croft [20] substantiates that there is a correlation between the terms used in documents and the creation time of documents. For a more detailed overview of the topic see [10].

2.2 Burstiness

A central problem in text data mining is to derive meaningful structure from document streams that arrive continuously and chronologically over time. Streams such as e-mails and news articles are natural examples where the documents can be characterized by topics that appear, grow in intensity, and then fade away. A burst, or a ”burst of activity”, can be viewed as a time period where e.g. a word or collection of words exhibit an atypically high frequency. Identifying such bursts can prove to be useful in the area of topic detection and tracking (TDT). In [13], Kleinberg proposes a formal approach for modeling such ”bursts”. The goal is to model these ”bursts” in such a way that they can be efficiently and robustly identified, and can provide an organizational framework for analyzing the underlying content [13]. Kleinberg's method is based on a Hidden Markov Model and uses an infinite-state automaton, where bursts appear naturally as state transitions. The resulting algorithms proved to be highly efficient, and return a nested representation of the set of bursts which can be used do establish a hierarchical structure on the overall stream.

More recently Lappas et al. [17] proposed a different algorithm for using burstiness to search for document sequences. Instead of using a Hidden Markov Model, as in [13], the algorithm is based on the method by Ruzzo et al. [27] for solving the *All Maximal Scoring Subsequences Problem*. In [17], two algorithms are proposed. The first algorithm **MAX-1** simply considers the occurrences of terms over the timeline generating scores with respect to a baseline (e.g. average occurrence), and applies the algorithm proposed by Ruzzo et al. [27] on the scores. The second algorithm **MAX-2** bases itself on the **MAX-1** algorithm, but instead of terminating after computing the initial set of bursty intervals I, it reapplies **MAX-1** on I. This way the **MAX-2** algorithm captures a second level of bursty intervals when the event was particularly popular and extensively covered in the news. These intervals cover typically when the event first occurs or recent development causes it to reappear on the front pages of the newspapers. The **MAX-1** method uses the all maximal scoring subsequences algorithm described in section 3.6 and the pseudocode for the **MAX-2** method is shown in Algorithm 1. Applying multiple iterations of **MAX-2** could be used to obtain a hierarchical structure of bursty intervals, but as demonstrated by Lappas et al. in [17] a single iteration is enough to capture the burstiness patterns of events. In the same article Lappas et al. also provide extensive experiments comparing **MAX-1** and **MAX-2** with Kleinberg's [13] method **KLEIN**. These experiments include comparing the different methods' abilities regarding document ranking, interval ranking and index statistics. Table 2.1 shows the results that these methods achieved for interval ranking. In the experiment they used the three different methods to evaluate queries from a manually composed list of major events. The goal was to identify the interval that is closest to the actual date of the event. As seen from the results both **MAX-1** and **MAX-2** produces reasonable intervals for the evaluated query. Expectedly, **MAX-2** gives shorter intervals, commonly spanning a few days or weeks around the actual date. **KLEIN** produced intervals similar of that to **MAX-1**, but failed to identify intervals for some of the queries. Since finding intervals in which events occur is the application of these methods that are relevant to our thesis, we will not go into details on the other two experiments in [17]. However it is worth mentioning that **MAX-2** and **KLEIN** achieved similar results regarding both document ranking and index statistics.

Actual Date	MAX-1	Max-2	KLEIN
Jan 17 1900	5 Jan - 3 Apr (1900)	5 Jan - 26 Jan (1900)	5 Jan - 23 Jan (1900)
June 30 1900	25 Jan - 12 Jul (1900)	1 Jul - 12 Jul (1900)	1 Jul - 12 Jul (1900)
Jul 29 1900	15 Jul - 19 Aug (1900)	30 Jul - 5 Aug (1900)	-
Sep 8 1900	3 Sep - 10 Mar (1900/01)	9 Sep - 6 Oct (1900)	10 Sep - 14 Sep (1900)
Jan 22 1901	5 Oct - 17 Mar (1900/01)	28 Dec - 8 Feb (1900/01)	-
May 3 1901	24 Apr - 29 Jul (1901)	27 Apr - 20 May (1901)	4 May - 23 May (1901)
Jun 11 1903	11 Jun - 25 Oct (1903)	12 Jun - 25 Jun (1903)	12 Jun - 19 Jun (1903)
July 20 1903	5 Jul- 4 Jan (1903/04)	7 Jul - 22 Jul (1903)	20 Jul - 22 Jul (1903)
Dec 30 1903	22 Dec - 20 Aug (1903/04)	31 Dec - 26 Jan (1903/04)	31 Dec - 17 Jan (1903/04)
Feb 7 1904	19 Jul - 20 Mar (1903/04)	5 Feb - 20 Feb (1904)	8 Feb - 20 Feb (1904)
Mar 31 1904	1 Apr - 6 Apr (1904)	3 Apr - 5 Apr (1904)	1 Apr - 6 Apr (1904)
Jun 15 1904	14 May - 30 Oct (1904)	16 Jun - 20 Jun (1904)	-
Jun 16 1904	20 Mar - 30 Oct (1904)	17 Jun - 31 Jul (1904)	20 Jun - 23 Jun (1904)
Feb 1 1908	2 Feb - 20 Feb (1908)	2 Feb - 11 Feb (1908)	-
Jul 25 1909	5 Mar - 10 Nov (1909)	19 Jun - 8 Aug (1909)	18 Jul - 27 Jul (1909)
Dec 28 1909	28 Nov - 28 Oct (1908/09)	26 Dec - 18 Jan (1908/09)	-

Table 2.1: Predicted Intervals for Major Events [17].

Algorithm 1 MAX-2

Input: $\mathcal{I}$: Set of first-level maximal intervals for Y_t
Output: $\mathcal{I}'$: Set of second-level maximal intervals for Y_t

```
1: $\mathcal{I}' \leftarrow \varnothing$
2: **for** every interval $I \in \mathcal{I}$ **do**
3:     $\mathcal{I}' \leftarrow \mathcal{I}' \cup$ MAX-1($I$) // **MAX-1** returns 1st level intervals
4: return $\mathcal{I}'$
```

2.3 Content Based Dating Approaches

The concept of document dating has been studied in several domains and it is growing in importance. Several Information retrieval applications such as computing document relevance and labeling search queries with temporal profiles depend on knowledge of when documents were posted. Another area of importance is processing historical and heritage collections of text [2]. Previous approaches have tried to address the problem by looking for linguistic constructs with clear temporal interpretations (e.g the mention of a date or time). However these may not refer to relevant time frames which makes it hard to estimate a publishing date for the given document. An improvement to this is to consider the entire vocabulary of a document in order to identify its timestamp. Although such statistical content-based methods have shown some promising results automatic document timestamping has still proven to be a tough problem. A main reason for this is that many documents do not contain temporal information in their content which make them useless for testing and training purposes [14].

The work of De Jong et al. [9] is among the first that addresses content-based document dating. The authors propose a language model where the timeline is pre-segmented into fixed intervals and then builds a model for each of these intervals. Based on this the model selects an interval which is most likely to be the temporal origin of the query document. The idea is that words may be obscured by language evolution and usage and that a language model may be able to detect similarities between semantics in documents and semantic trends over a time span. In the paper, two approaches are used. In the first approach the normalized log likelihood ratio is computed between each document in the reference corpus and the query document. From this, the publication date from the top-k documents are used to build a temporal profile from which the query document's time partition is decided. In the second approach the word frequencies from all documents belonging to a time partition is summed up and a temporal language model for each time partition is built beforehand. The undated documents are then compared directly to the model.

In [11] and [12] Kanhabua and Nørvåg propose a method for document dating that extends the one proposed by De Jong et al. They propose the reference collection to be preprocessed based on its semantic, and apply a term-weighting scheme based on their previous work on temporal entropy [11]. Such preprocessing techniques can be Part-of-Speech Tagging, Collocation Extraction, Word Sense Disambiguation, Concept Extraction and Word Filtering. To improve the accuracy further the authors also propose to add word interpolation, temporal entropy and external search heuristics from Google Zeitgeist.

Chambers [2] proposes a discriminative model which outperforms the previous ones proposed by De Jong and Kanhabua. It combines both a Maximum Entropy classifier, as well as defining rules for processing temporal linguistic features. The method learns different constraints based on time expressions in the document. E.g. a sentence like *"The Planetarium does not open until February 2000"* should remove all future years beyond 2000 from consideration and a constraint that says *"This document was likely written before 2000"* should be learned. The limitations of Chambers' method is that it only works for year predictions because of the ambiguity of the temporal linguistic features that refers to months or days.

The different methods mentioned thus far all have the same limitations. They require a pre-segmentation of the timeline into fixed intervals. In [14], Kotsakos et al. propose

Time Frame Length	NLLR[9]	MaxEnt[2]	BurstySimDater[14]
1 month	18	-	23.4
3 months	24	-	32
6 months	25	36	40
1 year	38.4	48.6	49.8

Table 2.2: Precision (%) for NYT10 Dataset [14].

a method that has no such requirements and can handle intervals of arbitrary length. The method considers the lexical similarity of a query document with documents in a reference corpus as well as the burstiness of terms over time. The motivation for using burstiness as a factor to decide a document's timestamp is that when an event occurs, terms that are characteristic for the given event appears more frequently in the media recording its progress. Term burstiness is computed for the overlapping terms between the query document and a document from the top-k most similar documents from the reference corpus and a weight is given to the reference document. All the weights from the top-k most similar documents are summed up for all the different time intervals possible and the one with the highest score is the one predicted as the publication date for the query document. In addition to providing a dating method that reports non-fixed periods of time, Kotsakos et al. [14] also provide an extensive experimental evaluation comparing their own method to the solutions proposed in [9] and [2] (Table 2.2).

Chapter 3

Preliminaries

This chapter provides general information on the techniques that we have used in our project. Section 3.1 explains different ways of representing documents while Section 3.2 describes preprocessing of documents. Sections 3.3 and 3.4 describe the Jaccard similarity measure and all pairs similarity search respectively. Section 3.5 explains the burstiness of terms and, finally, an overview of the all maximal scoring subsequence problem is given in Section 3.6.

3.1 Document Representation

The representation of a document is how the information is stored and used to represent its content. The representation should consider what tasks that are to be performed on the documents. In addition, the representation should also be able to compress the documents which makes computation easier.

3.1.1 Bag of Words Model

Due to its simplicity, efficiency, and often surprising accuracy, the bag of words model is one of the most common fixed-length vector representations for text. In a bag of words model each document is treated as an unordered set of words. With this model a document can be represented as a vector, where the vector size is equal to the size of the vocabulary in the corpus and each entry corresponds to a word. The entries in the vector could be binary (word present or not), the number of occurrences of the word or the term frequency. Consider the following three documents:

D_1 : *"President Donald Trump has very nice hair"*

D_2 : *"Donald Trump has small hands"*

D_3 : *"Small hands are very nice"*

From these three documents (after a simple stop word removal) we may have the vocabulary:

V = {*donald, hair, hands, nice, president, small, trump, very* }

and the bag of words binary vector representations are as follows:

$\vec{D_1} = \{1, 1, 0, 1, 1, 0, 1, 1\}$

$\vec{D_1} = \{1, 0, 1, 0, 0, 1, 1, 0\}$

$\vec{D_1} = \{0, 0, 1, 1, 0, 1, 0, 1\}$

The bag of words model has some critical drawbacks. One major drawback is the problem of sparsity in the vector representation of the documents. Consider a corpus consisting of hundred thousands of documents with over a million distinct terms. Using a naive bag of words approach to represent the collection would include generating hundred thousands of 10^6-dimensional vectors where only a small fraction of the entries are non-zero values. Another limitation to the bag of words approach is that it only considers single words. As pointed out by Le et al. [18] bag of words models does not take the word order into consideration and have very little sense about the semantics of the words.

3.1.2 K-Shingles

A *k-shingle* (or *k-gram*) for a document is a sequence of k tokens that appear in the document. Tokens can be characters, words etc. depending on the application. As an example one could have a look at the document:

D_1 : *"knowledge is important"*

If we set $k = 2$ we will get the set of *bigrams* which is:

$D_1(bigrams)$ = {*knowledge is, is important*}

This way you will get more information about the collocation and the context of the terms. The *k-shingles* can also be represented as vectors as shown for the bag of words model in Subsection 3.1.1.

3.2 Semantic-based Preprocessing and Stop Word Removal

Preprosessing is extremely important when it comes to text mining. One might even say it is *defined* by these elaborate preparatory techniques because it is so dependent on them. One well known technique is stop word removal. Stop words are common words of the language that usually do not contribute to the semantics of the document. They add little information to the document and removing these will lead to a reduction of the size [6].

3.2.1 TF-IDF

TF-IDF, short for term frequency-inverse document frequency, is a numerical value intended on reflecting the importance of a term within a given document as described in [6]. In information retrieval, TF-IDF is a widely used technique for identifying the index-terms in documents. The TF-IDF score of a term t within a document d is calculated as the product of the term frequency of t in d and the logarithmically scaled inverse document frequency of t in the entire document collection. Here the term frequency of t is the number of occurrences of term t in document d, and the inverse document frequency is the number of documents in the collection divided by the number of documents where term t occurs. The idea behind using the TF-IDF weighting scheme is to reward terms that occur frequent within a document, while penalizing words that occur in many documents. This way the TF-IDF score gives a measure as to how well a given term is suitable for distinguishing a given document from other documents (*idf*), while at the same time making sure that the term represents the content of the document (*tf*).

$$idf_t = \log(\frac{N}{df_t}) \tag{3.1}$$

$$tf{-}idf_{t,d} = tf_{t,d} \times idf_t \tag{3.2}$$

3.2.2 Part-Of-Speech Tagging

Part-of-speech (POS) tagging is the process of assigning a word with a label that corresponds to its syntactic class. By using POS tagging in the data preprocessing stage it is possible to eliminate a significant amount of stop words. The underlying assumption is that words of interest (words describing an event) for the most part belong to word classes such as nouns, verbs and adjectives. This way, using POS tagging could help with increasing the accuracy when it comes to determining a document's date [11]. To apply POS tagging, there exist a number of available tools such as the Stanford Core Nlp toolkit [22] which is the one we used in our implementation. This is a JVM based annotation pipeline framework, which provides most of the common core natural language processing steps. This includes pre-trained models for POS tagging and an API for using them.

3.2.3 Named Entity Recognition

Named Entity Recognition is the process of identifying named entities in text and assigning each entity with a label that corresponds to its type, such as ”Person”, ”Location”, ”Organization” etc. Identifying the terms that separates a given event from another is an important factor when detecting the topics from another. This is why it would seem like a good idea to look at the named entities in the documents, since a combination of these could be unique for different events. Although this intuitively seems like a good idea, observations made by Kumaran et al. [16] show that the utility of named entities can be a double edged sword. This is because the same named entities are often used in many different contexts. To overcome this issue, named entities can be used with a combination of keywords from the documents that you can find by using other methods, such as TF-IDF

(3.2.1). This way it is possible to capture both the entities the documents describe, as well as the the context.

3.3 Jaccard Similarity

The Jaccard Similarity is a measure that can be used to describe the similarity between two documents q and d. It is computed as the quotient of the overlap and the union of their respective vocabularies. [14]

$$Jaccard(q, d) = \frac{q \cap d}{q \cup d} \tag{3.3}$$

The pseudocode for computing the Jaccard similarity between documents is given in Algorithm 2.

Algorithm 2 Naive Method for Precomputing the Top-k Most Similar Documents

Input: Set of documents $\mathcal{D}$
Output: List of most similar documents $\mathcal{J}$

```
1: $\mathcal{J} \leftarrow \varnothing$
2: for $q \in \mathcal{D}$ do
3:     $S_q \leftarrow$ number of terms in $q$
4:     for $d$ not equal $q \in \mathcal{D}$ do
5:         $S_d \leftarrow$ number of terms in $d$
6:         $I \leftarrow$ number of intersecting terms in $q$ and $d$
7:         $JacSim \leftarrow \frac{I}{S_q + S_d - I}$
8:         $\mathcal{J} \leftarrow \mathcal{J} \cup JacSim$
9: Return $\mathcal{J}$
```

3.4 All Pairs Similarity Search

Solving a similarity search problem where one is interested in all pairs of objects whose similarity is above a specified threshold is required in many real-world applications. Such applications can be query refinement for web search by finding all pairs of similar queries based on the similarity of the search result of those queries [28], coalition detection for identifying coalitions of click fraudsters [23] and so on. This problem is a generalization of the well-known nearest neighbor problem and there has been done a lot of work on this subject, with many recent works considering various approximation techniques [3, 5, 7, 8]. On the other hand, in [1] Roberto J. Bayardo et al. propose a few methods which do not resort to approximation or discarding of frequent features. These includes building an inverted list index of the input vectors dynamically, exploiting the threshold during indexing and/or during matching, exploiting a specific sort order etc. The biggest challenge when performing all pair similarity search between documents in large corpora is the problem of high dimensional and sparse data.

3.4.1 LSH and MinHash

One of the problems of comparing documents is that they are simply too large. We need a representation of a document to be smaller so that it can be compared even faster. For this it is possible to use a set of *MinHash signatures*. Now that the documents are much smaller we are able to do the comparison a lot faster, but another problem that we might enter is that the document collection also is too large. To solve this problem one can use a function $f(S1, S2)$ that tells whether $S1$ and $S2$ is a pair of elements whose similarity must be evaluated, a *candidate pair*. The function aims to extract and return almost all the pairs that have a similarity measure that is higher than a given threshold and almost none that have a similarity measure that is below, resulting in a set of candidate pairs that is a lot smaller than the original set of pairs. To decide whether or not two documents is a candidate pair the signature matrix is split into b bands with r rows each. Then Each band of a document is hashed into a bucket. If there are two or more documents in the same bucket the combination of these are candidate pairs. J. Leskovec also shows the probability that signatures between $S1, S2$ agree in one row as well as further analysis in [19].

3.5 Term Burstiness

A term is defined as bursty in periods where it exhibits atypical high frequencies. A more formal definition of term burstiness is presented in the terms of numerical discrepancy by Lappas et al. in [17]. In order to define term burstiness we first need to present the general definition of numerical discrepancy. The numerical discrepancy is defined in Equation 3.4. Here we have that P is a set of points distributed over random locations in $[0, 1]^d$, where d is the number of dimensions on the plane. For any region R in $[0, 1]^d$ we have that $\mu(R)$ is the Euclidean measure of $R \cap [0, 1]^d$ and $\mu_p(R)$ is the discrete measure $|R \cap P|/|P|$.

$$D_p(R) = |\mu(R) - \mu_p(R)| \tag{3.4}$$

Since we are interested in finding bursty intervals of terms over the timeline, R is reduced to a one-dimensional interval I (e.g. $d = 1$). The discrepancy of I is calculated by taking the absolute value of the difference between its length and the ratio of points from P that fall within I (Equation 3.5). Here, $\mu(I)$ represents the baseline that is the expected fraction of points to fall within I, while $\mu_p(I)$ represents the observed fraction.

$$D_p(I) = |\mu(I) - \mu_p(I)| = \left| len(I) - \frac{|P \cap I|}{|P|} \right| \tag{3.5}$$

Regarding term burstiness, the set of points P is represented as the total frequency of a term over the entire document's sequence. When it comes to the baseline $\mu(I)$, it can either be pre-defined or based on the underlying distribution. Lappas et al. [17] point out out that you can not assume that the entire data set can be accurately described by a single distribution. Instead they define the baseline as follows:

$$\mu(I) = \frac{len(Y_t[l : r])}{m} \tag{3.6}$$

Here we have that $Y_t[l : r]$ is the frequency sequence for a term t and m is the length of the time series. Additionally, Lappas et al. [17] define $\mu_p(I)$ as the frequency of term t observed within interval $Y[l : r]$ divided by the total frequency of t throughout the sequence:

$$\mu_p(I) = \frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} \tag{3.7}$$

By putting Equations 3.6 and 3.7 into Equation 3.5, we get:

$$D_p(I) = \left| \frac{len(Y_t[l:r])}{m} - \frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} \right| \tag{3.8}$$

Equation 3.8 defines the numerical discrepancy of terms over the timeline, but is of little value for measuring term burstiness. This is because it gives positive values for observations that are either greater or less than the baseline. Instead, we would like a measure that gives positive values only for uncommonly high frequency observations. Thus, given a term t and an interval $[l : r]$ on the timeline, Lappas et al. [17] define the burstiness of t in [l:r] as:

$$B(t, [l:r]) = \left(\frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} - \frac{len(Y_t[l:r])}{m} \right) \tag{3.9}$$

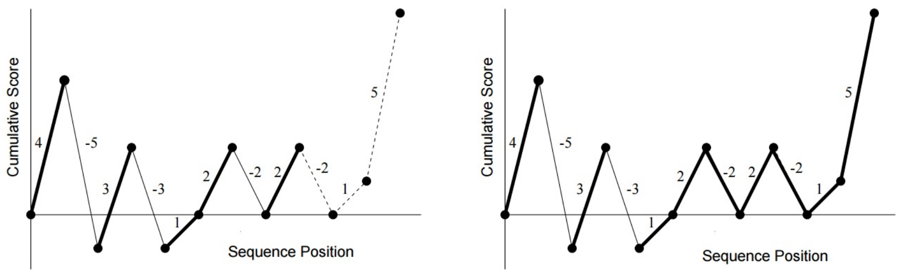
3.6 All Maximal Scoring Subsequences Problem

Finding all maximal scoring subsequences is often an important task in analyzing e.g. protein sequences or as we are going to be using it, finding burstiness of terms over a given period of time. When using it to find bursty intervals the subsequences can not overlap because either the union or the intersection of the two would have higher discrepancy than one of the two, creating a contradiction as discussed in [17]. Thus, the problem can be translated into the problem of finding all non-overlapping contiguous subsequences having the greatest total score given a sequence of real numbers. In [27] Ruzzo et al. presented a linear time algorithm that finds all maximal scoring subsequences. In the paper they also provide mathematical proofs for the correctness of their algorithm. The input for the algorithm is a list of real numbers read from left to right and the algorithm maintains both the cumulative total of the scores and a certain ordered list $I_1, I_2, \ldots, I_{k-1}$ of disjoint subsequences. For each subsequence I_j, it records the cumulative total L_j of all scores up to but not including the leftmost score of I_j, and the total R_j up to and including the rightmost score of I_j. A positive score is incorporated in its own subsequence and the algorithm then checks to see if extending the subsequence will make a new subsequence with higher score. As described in [27] this is done as follows:

1. The list is searched from right to left for the maximum value of *j* satisfying $L_j < L_k$.
2. If there is no such *j*, then add I_k to the end of the list.
3. If there is such a *j*, and $R_j \geq R_k$, then add I_k to the end of the list.

4. Otherwise (i.e., there is such a j, but $R_j < R_k$), extend the subsequence I_k to the left to encompass everything up to and including the leftmost score in I_j. Delete subsequences $I_j, I_{j+1}, \ldots, I_{k-1}$ from the list (none of them is maximal) and reconsider the newly extended subsequence I_k (now numbered I_j) as in step 1.

Figure 3.1 An example of the algorithm from [27]. Bold segments indicate score sequences currently in the algorithm's list. The left figure shows the state prior to adding the last three scores, and the right figure shows the state after.



This algorithm can be used to identify high-burstiness intervals for different terms where the input is the frequency sequence of the term over a given timeframe subtracted by the average frequency of the term. This average makes up the term's baseline which is the expected value of its occurences. An uncommonly high frequency observation will give a high score while a frequency observation lower than expected will give a low (negative) score.

An example of the algorithm can be seen in Figure 3.1. In this case the algorithm runs through a sample input with the scores $(4, -5, 3, -3, 1, 2, -2, 2, -2, 1, 5)$. After reading the last three scores, the last subsegments are extended to encompass these scores as well. In the process the algorithm creates a new segment, I_5 containing the value 1 (tenth iput) and then merge I_3, I_4 and I_5 into a new I_3 as the last score arrives.

Chapter 4

Document Dating

In this section we will provide a more detailed overview of previously proposed document dating techniques. In Section 4.1 we will explain the technicalities behind the NLLR unigram model proposed in [9]. Then, in Section 4.2, we will explain the *Temporal Entropy* weighting scheme described in [11]. In Section 4.3 we will describe the principles behind the *MaxEnt* document dating method proposed in [2]. Finally, in Section 4.4, we will explain the *burstySimDater* method given by Kotsakos et al. in [14].

4.1 Normalized Log Likelihood Ratio (NLLR)

When comparing a query to a document collection it is of great importance to use a good metric for comparison. De Jong et al. [9] propose to use the normalized log likelihood ratio presented by Kraaij et al. [15]. For a query Q, a document D and a background corpus C, the log likelihood ratio of Q given D is defined as follows:

$$LLR(Q|D) = \log \frac{P(Q|D)}{P(Q|C)} = \sum_{i=1}^{n} c(Q,\tau_i) \log \frac{(1-\lambda)P(\tau_i|D) + \lambda P(\tau_i|C)}{P(\tau_i|C)} \tag{4.1}$$

Here, n is the number of terms in Q, τ_i is a term in Q, $c(Q,\tau_i)$ is the number of times τ_i occurs in Q and λ is a smoothing parameter. For each term in the query, the LLR model gives a measure of the degree of surprise there is to see the term given the document model compared to the degree of surprise given the background model. The scores in equation 4.1 depend on the query length. This can easily be normalized by dividing the scores by the query length, which gives the normalized log likelihood (NLLR):

$$NLLR(Q|D) = \sum_{i=1}^{n} \frac{c(Q,\tau_i)}{\sum_{i=1}^{n} c(Q,\tau_i)} \log \frac{(1-\lambda)P(\tau_i|D) + \lambda P(\tau_i|C)}{P(\tau_i|C)} \tag{4.2}$$

By applying the NLLR formula (equation 4.2) to our problem we get the following formula, which finds the time partition that best suits a given document:

$$NLLR(D|p) = \sum_{w \in D} \frac{t_f(D, w_i)}{\sum_{i=1}^{n} c(D, w_i)} \log \frac{(1-\lambda)P(w_i|p) + \lambda P(w_i|C)}{P(w_i|C)} \tag{4.3}$$

In equation 4.3 we have that D is the query document, p is a given time partition, $t_f(D, w)$ is the term frequency of word w in document D, $\sum_{i=1}^{n} c(D, w_i)$ is the number of words/length of document D, $P(w|p)$ is the likelihood of term w appearing in partition p and $P(w|C)$ is the likelihood of term w appearing in the background corpus C.

4.2 Temporal Entropy

In order to achieve good results it is important to use a good term weighting scheme. A naive choice would be to simply use the raw term frequencies. However this would result in a lot of noise in the language model, since meaningless stop words would be overrepresented. Kanhabua et al. [11] propose a term weighting scheme concerning temporality called *temporal entropy* (TE), where terms are selected based on their entropy or noise measure. The idea is inspired by a term selection method presented by Lochbaum et al. [21]. The advantage of using a term weighting scheme such as entropy is that it gives a measure as to how well a term is suited for separating a document from other documents. Additionally, it also captures the importance of the term within the document. The entropy of a word w_i is defined as follows:

$$Entropy(w_i) = 1 + \frac{1}{\log N_D} \sum_{d \in D} P(d|w_i) \times \log P(d|w_i) \tag{4.4}$$

In equation 4.4 we have that: $P(d_j|w_i) = \frac{tf(w_i, d_j)}{\sum_{k=1}^{N_D} tf(w_i, d_k)}$, N_D is the total number of documents in a collection D and $tf(w_i, d_j)$ is the frequency of a word w_i in a document d_j. Kanhabua et al. [11] defines temporal entropy as a measure as to how well a term is suitable for separating a time partition from other time partitions as well as indicating how important a term is within in a specific partition. They define temporal entropy as follows:

$$TE(w_i) = 1 + \frac{1}{\log N_P} \sum_{p \in P} P(p|w_i) \times \log P(p|w_i) \tag{4.5}$$

Similarly to equation 4.4, we have in equation 4.5 that: $P(p_j|w_i) = \frac{tf(w_i, p_j)}{\sum_{k=1}^{N_P} tf(w_i, p_k)}$, N_P is the total number of partitions in a corpus P, and $tf(w_i, p_j)$ is the frequency of w_i in partition p_j. By modifying the score measure in [9], Kanhabua et al. [11] proposes the following score measure:

$$Score_{TE}(d_i, p_j) = \sum_{w \in d_i} TE(w) \times P(w|d_i) \times \log \frac{P(w|p_j)}{P(w|C)} \tag{4.6}$$

Using the score measure defined in Equation 4.6 for classifying documents could prove to be advantageous since it applies higher weights to terms that occur in few partitions, which in turn should result in a higher score for partitions where such terms appear.

4.3 MaxEnt Document Dater

The two techniques discussed so far can be viewed as token-based classifiers where the learners solely rely on the observed frequencies of unigrams in order to assign a query document with a time partition. Although they manage to capture and utilize some of the temporal information in a document collection, Chambers et al. [2] point out that they do not consider richer time-based features such as *typed dependency*, *verb tense*, *verb path* or *named entities*. Instead Chambers et al. propose to use such features in order to learn and enforce time constraints on the query document. The key idea is to identify features such as "until February 2000", which in this case is a year mention that excludes documents written in year 2000 or later, and use these to narrow down the possible candidate time partitions. The authors define three **core relations**:

1. **Before** Timestamp: $M < T$
2. **After** Timestamp: $M > T$
3. **Same** as Timestamp: $M == T$

Here T is the document timestamp's year and M is the year mention. In order to classify year mentions Chambers et al. [2] use a MaxEnt classifier that labels mentions with relations. Regarding the document classifier, they introduce the following technique for mapping the relations to individual year predictions:

$$P_{year}(y|d) = \frac{1}{Z(T_d)} \sum_{t \in T_d} P_Y(rel(val(t) - y)|t) \tag{4.7}$$

$$rel(x) = \begin{cases} before & \text{if } x < 0 \\ after & \text{if } x > 0 \\ simultaneous & \text{otherwise} \end{cases} \tag{4.8}$$

Here we have that T_d is the set of mentions in document d. A MaxEnt classifier is represented by $P_Y(R|t)$ for a time mention $t \in T_d$ and possible relations R. The mapping of this distribution over relations to a distribution over years is defined by $P_{year}(y|d)$ where $val(t)$ is the integer representation of the year mention and $Z(T_d)$ is the partition function. The $rel(val(t) - y)$ function determines if the year mention t overlaps the predicted year for the document's "unknown" timestamp y. Finally Chambers et al. [2] define a joint model by combining the unigram model from Section 4.1 with their own constraint classifier with the following linear interpolation:

$$P(y|d) = \lambda P_{doc}(y|d) + (1 - \lambda) P_{year}(y|d) \tag{4.9}$$

Here we have that y is a year, d is the query document and λ is a real number between 0 and 1 that sets the level of impact between the two summands in the equation.

4.4 BurstySimDater

In [14] Kotsakos et al. propose a method for dating documents called *burstySimDater*. This is a content-based document dating algorithm which considers both lexical similarity as well as burstiness of the terms throughout the dataset. To achieve this it studies the terms in the **New York Times Annotated Corpus** to compute the burstiness of the different terms. The intervals of when each term is bursty are stored and used later for deciding the publishing date for a given query document. The Jaccard similarity for every pair of documents is also precomputed and stored for later use. The paper studied other similarity measures as well such as cosine similarity, but concluded that the Jaccard similarity measure was the best suited. When a query document is to be dated a K-fold cross-validation process is started and a given number of documents from the training set is selected for comparison. To find the best number of documents selected the authors ran a few different tests and decided to go for 10, which means that the 10 most lexical similar documents according to the Jaccard similarity measure when compared to the query document are used. For each of these top 10 documents the intersecting words with the query document are extracted for burstiness evaluation. Now, for each of the intersecting words, if the training document is dated in the same interval the given term is bursty a weight of 1 is added to the document. After all the terms are evaluated the weight is normalized by dividing it by the number of intersecting terms. Finally the weighted documents are analyzed and the interval of a given length containing the highest sum from the weight of the documents dated in this interval is used for dating the query document. The accuracy for the method is computed and a correct answer is given for each document where the actual publishing date is within the estimated interval. The pseudocode for the *burstySimDater* is given in Algorithm 3.

Algorithm 3 BurstySimDater

```
Input: reference corpus $\mathcal{D}$, bursty intervals $\mathcal{B}$, query document $q$, max time frame length $l$
Output: time frame of q
1:  $\mathcal{S} \leftarrow$ top-k most similar documents to $q$ from $\mathcal{D}$
2:  $W_\mathcal{S} \leftarrow \varnothing$
3:  for $d \in \mathcal{S}$ do
4:      $w_d \leftarrow 0$
5:      $\mathcal{Y} \leftarrow d \cap q$
6:      for $x \in \mathcal{Y}$ do
7:          $w_d \leftarrow w_d + |\{I \in B(x) : t(d) \in I\}|$
8:      $w_d \leftarrow w_d / |Y|$
9:      $W_\mathcal{S} \leftarrow W_\mathcal{S} \cup \{w_d\}$
10: $A_\mathcal{S} \leftarrow (d \in \mathcal{S}, W_\mathcal{S})$
11: $\mathcal{I} \leftarrow$ getMaxSumInterval$(A, l)$
12: Return $\mathcal{I}$
```

Chapter 5

Approach

This chapter provides a detailed overview of our approach to realizing a content based document dating system. This includes a description of how we selected index terms, given in Section 5.1, and how we precomputed the top-k most similar documents and the bursty intervals, given in sections 5.2 and 5.3 respectively. A description of the database used to store this information is given in Section 5.4. The different methods for weighting the documents are explained in Section 5.5 and finally Section 5.6 describes how the final interval for the documents is estimated.

5.1 Finding Index Terms

A large portion of the text in documents consists of stop words. These are ”meaningless” words, such as ”he”, ”is”, ”a”, ”to” etc, that do not incorporate the topic of the document and can not be used efficiently for distinguishing documents from another. Not only do stop words act as noise when comparing documents, they also make the comparison more computationally heavy. This is because they constitute a large portion of both the vocabulary and the total volume, which results in higher dimensional document representations. Therefore we had to implement some means of stop word removal. There exist several well known techniques for stop word removal, or keyword filtering. The techniques we chose to explore were TF-IDF, POS tagging and Named Entity Recognition. TF-IDF is proven to be an efficient and simple algorithm for identifying relevant words in a document [26]. The key idea behind TF-IDF is to reward terms that occur frequent within a documents while at the same time penalizing terms that occur frequent throughout the corpus, as described in Section 3.2.1. For each term in each document we calculated the TF-IDF score and then reduced the documents to only containing a fraction of its highest scoring terms. This fraction is called the *keyword density*. To calculate the TF-IDF scores we had to index each term in the corpus with the number of documents in which the term appeared and generate indices for each document with its terms and the occurrence of the term within the document. Then we could simply join the two indices and apply the TF-IDF formula. Regarding POS tagging, we found index terms by filtering out specific parts of speech. This was done by using the Stanford Core NLP toolkit [22]. In addition to using POS tagging to find index terms, we also used Named Entity Recognition to identify meaningful terms. In order to recognize the named entities in the corpus we used the Stanford Core NLP's Conditional Random Field Classifier [22]. The model we used was a pre-trained 4-class model that classifies char sequences as either person, organization, location or misc. Since the named entities are char sequences, not single terms, we tried an approach where we removed all white-spaces from the named entities so that the system would regard a named entity as a single term. However this turned out to be a poor idea, since char sequences such as ”Barack Obama” and ”President Obama” would be considered as two different entities. Hence, we ended up using a simple BOW-model for the named entities assuming that most of the co-occurences could be captured when looking for simultaneous bursts.

5.2 Precomputing the Top-k Most Similar Documents

To be able to run the program several times with different parameters we needed it to terminate within reasonable time. Computing the Jaccard similarity between all documents, one pair at a time, is an all pairs similarity method which, for a dataset of more than 800,000 documents, would take too long. Therefore, we decided to precompute this value for all of the documents. This way we could simply look up the k most similar documents in a map stored on disk, instead of having to compute the similarities from scratch for each query document. For our dataset the naive double for-loop Jaccard similarity algorithm was not able to scale. Hence, we needed a more sophisticated approach. First we tried to estimate the similarities by using MinHash-LSH, but unfortunately this did not give precise enough estimates. We suspect that the reason behind this is that the syntactic similarity between the documents in the collection we used are typically low. This means that in order to find the topmost similar documents we need to look at the lower decimals in the Jaccard similarity values. Therefore we need very precise, if not exact, estimates for similarities. However, if we were working on a different collection, such as a twitter stream or query logs where the documents are very short which in turn gives a higher probability of high similarity between best matches, it would be likely that MinHash-LSH would give satisfying estimates. To be able to do the all pairs similarity computation we had to find a different way of representing the documents. We ended up creating a document model for all the different documents containing a set of all the terms from the document as well as a field holding the GUID of the document. We then proceeded by creating a list with all the document models, *docModels*, and a hashmap of each of the documents' size according to its termcount, with the document's GUID as key, *docSizes*. From the document models we created a new hashmap with all the distinct terms from the dataset as keys holding the GUIDs for all the documents containing the given term, *termCatalogue*. By using a single for-loop we can iterate through the documents and by scanning *termCatalogue* we are able to compute the number of intersecting words between the query documents and all others. In addition we discard all the documents that do not have any intersecting terms with the query documents to make the set on which the Jaccard similarity is to be computed smaller. Now we could compute the Jaccard similarity for each document in reasonable time with some simple calculations and then sort them by values to find the most similar ones. This information was stored in our database to be used later.

5.3 Precomputing Bursty Intervals

To save more time when running the program we decided to precompute the bursty intervals for each distinct term in the document collection as well. For this task we created an overview for all of the terms and all their respective timestamps (i.e. publication dates and occurences of all the documents containing the given word) and gave each term a score based on how many times it was used for any given day throughout the time span of the document collection. To be able to find when a term has uncommonly high frequency observations we defined a baseline which we subtracted from the scores. The baseline we used was the average frequency of the term.

The algorithm for calculating the bursty intervals would read the scores of each term from left to right (i.e. from the earliest publication date to the last) and keep track of the maximal scoring subsequences in an ordered list $I_1, I_2, ..., I_{k-1}$ calculated so far. Further the algorithm would generate the bursty intervals as described earlier and in [27].

The output from the method was also stored in our database as a key-value pair with the term as key and its bursty intervals as its value. In addition to producing the intervals we also used a plotting tool to be able to visualize the frequency of different words throughout the time span. We plotted both the frequency of the word as well as the scores after the baseline was subtracted. The plots for the word ”earthquake” and ”richter” in a collection of New York Times articles from 1987 are shown in Figure 5.1. These are two words that are often used in the same context and as you can see from the figure, they are spiking around the same time frames. The large spike between day 250 and 300 is most likely related to the *Whittier Narrows Earthquake*, which took place in Southern California October 1st 1987 and left 8 casualties as well as major material damage. This example visualized how a collection of words can have a simultaneous ”burst” of activity and then fade out over time.

After all the bursty intervals for the distinct words in the dataset were computed we wanted to create the bursty intervals from the **MAX-2** algorithm to capture the second level of bursty intervals. When computing these we would run the algorithm for finding bursty intervals within each of the intervals already found for a given term and then concatenate all the results into a final list of bursty intervals. This second burstiness level pertains to smaller intervals when the event was particularly popular and extensively covered in the news. This would typically be the first time an event made the headlines, or, in the context of a newspaper, when new developments bring it back to the front page. The different outputs from both the **MAX-1** and **MAX-2** algorithm is compared in Table 5.1 where the first column marks the timestamp of when Christmas Eve (i.e. December 24th) occurs over the timespan of the NYT10 dataset. The second and third column represent an interval that spans this timestamp from its respective burstiness algorithm. As we can see, the results from the **MAX-2** algorithm is narrower and more concentrated around the actual time of the event. For the **MAX-1** algorithm we see that there is even an interval spanning over the three first christmases, probably because the term frequency for *christmas* was lower at the end of the timespan. We also included a plot for the term frequency of the word *christmas* subtracted by the baseline to be able to visualize the different intervals better.

Event timestamp	MAX-1	MAX-2
358	297 - 1103	318 - 374
724	297 - 1103	682 - 738
1089	297 - 1103	1060 - 1102
1454	1406 - 1468	1431 - 1456
1819	1767 - 1838	1795 - 1820
2185	2148 - 2195	2171 - 2187
2550	2505 - 2561	2530 - 2551
2915	2869 - 2930	2906 - 2917
3280	3223 - 3300	3258 - 3281
3646	3601 - 3650	3626 - 3647

Table 5.1: The different intervals from the **MAX-1** and **MAX-2** algorithms spanning the timestamp of christmas each year.

This plot can be seen in Figure 5.2

5.4 Oracle Berkeley DB

For storing the precomputed data we wanted to use a simple database which offered key-value storage and ended up choosing Oracle's Berkeley DB. It uses simple function-call APIs for data access and management and the Oracle Berkeley JE version is implemented in Java [25]. In this way we got to put all the information from the Jaccard similarity results as well as the information of the bursty intervals into one file each instead of, for instance, having over a million files with the burstiness of each term.

5.5 Weighting the Documents

When we use the most similar documents from the test set to decide the time frame of the query document we want the ones that are considered more important to have more impact on the final decision. To be able to do this we give each document a weight and for each possible time frame we add all the weights of the documents within the given time frame. The time frame with the highest score will be our estimate of the time frame where we find the timestamp of the query document. The way we compute the document weights have a lot of impact on the end result and we chose to try a few different models.

5.5.1 Burstiness

The model that is similar to the one that is used in [14] is the one we call *burstiness*. For each document in the top-k most similar documents the intersection of words with the query document is extracted. For each of these words we analyze its burstiness and if the timestamp of the test-document is within a range where the word is bursty we add 1 to the

Figure 5.1 The top picture shows the term frequency of the word *earthquake* throughout the timespan of the NYT1987 dataset, while the bottom picture shows the term frequency of the word *richter*.

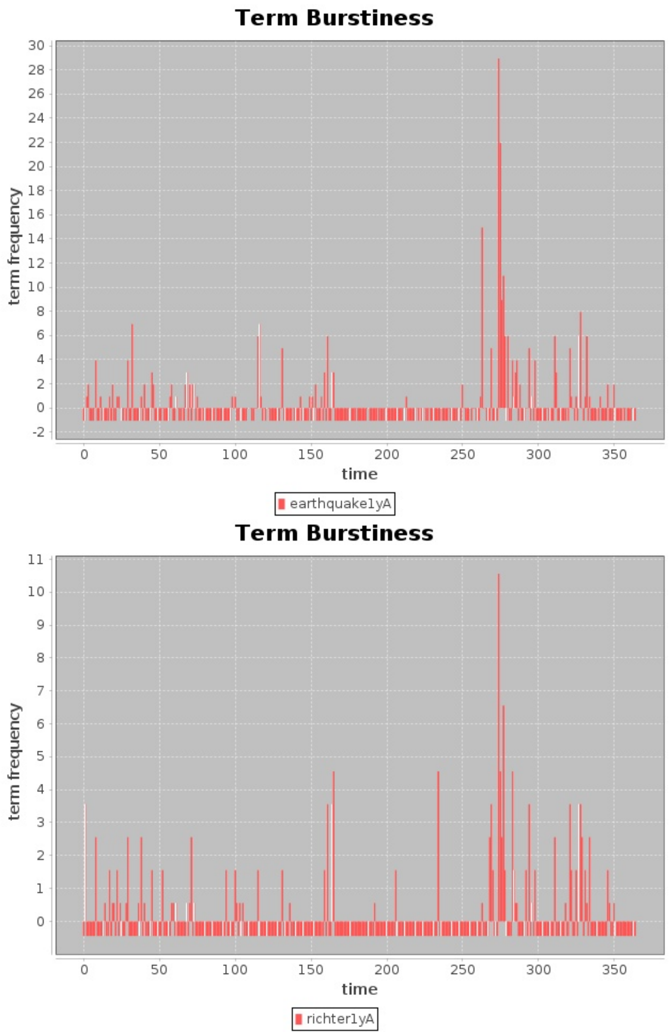


Figure 5.2 Plot of term frequency for the word *christmas* subtracted by the baseline.

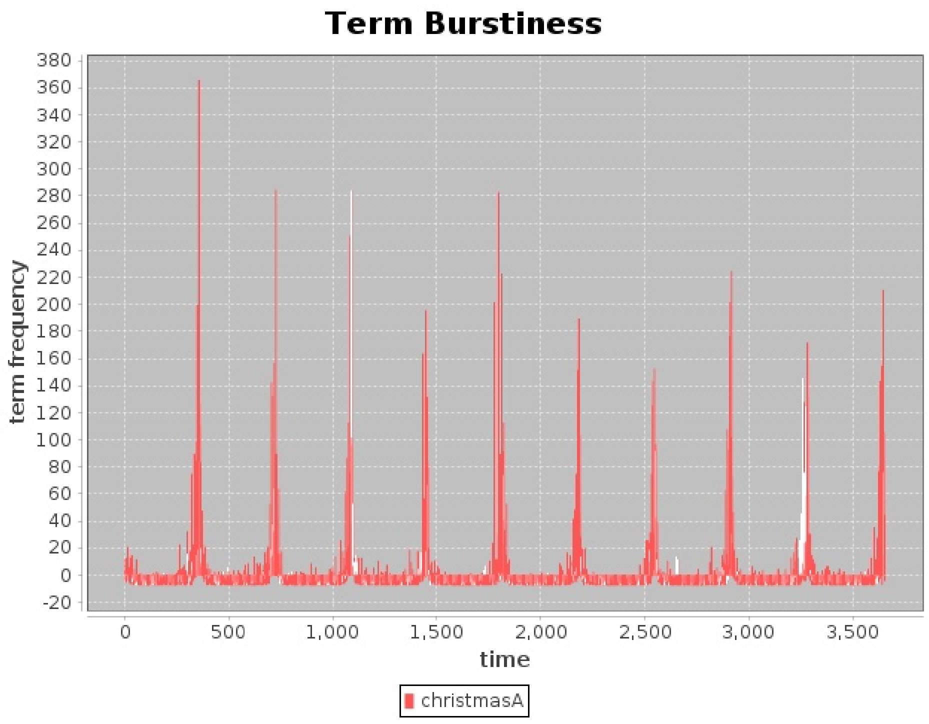
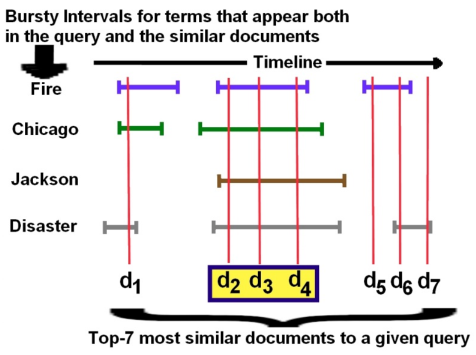


Figure 5.3 An example of how the burstiness is computed for the test-documents. In this case, the three documents d_2, d_3, d_4 will be given the highest score since they overlap with the most bursty intervals of the considered terms [14].



Bursty Intervals for terms that appear both in the query and the similar documents
Timeline
Fire
Chicago
Jackson
Disaster
d1
d2 d3 d4
d5 d6 d7
Top-7 most similar documents to a given query

weight of the document. After all the words in the intersection is analyzed we normalize the weight by dividing it by the number of terms in the intersection.

To visualize how the burstiness affects the weights of the different documents, the example in Figure 5.3 gives a conceptual view of the approach. Say that a given query document q discusses a fire in the Jackson theater in Chicago. The seven most lexical similar documents to q: $d_1, d_2, d_3, d_4, d_5, d_6$ and d_7 and a subset of the intersecting words are shown in the figure. Further the bursty intervals of each of these words are also shown.

The weight for any of the documents in the test set will be the normalized number of bursty intervals from the intersecting words with the query document that is bursty in its date of creation. The pseudocode of this computation can be seen in Algorithm 3.

5.5.2 Top-k

A different model, and perhaps the easiest one is giving each of the top-k most similar documents the weight 1. The result from using this method will be the time frame where the highest amount of the most similar test-documents are written. In other words, the algorithm choose the time frame of the query document to be around the same time the largest portion of the top-k most similar documents are written.

5.5.3 Jaccard Scores

The results from the top-k model proved to be promising and we decided to do a variation of this. Instead of giving all the k most similar documents the weight 1 we gave them their Jaccard score as weight so that the most similar document will have more impact than, say, the twentieth most similar document.

5.6 Estimating the Interval

The output of the program is an interval, or time frame, in which the program finds it most likely that the query document is published. The interval is of length l and will be the interval throughout the timeline with the highest sum of document weights. To compute this we create a list where the index corresponds to a timestamp in the timeline of the dataset and the value is the sum of the weights of all documents published on the corresponding date (index). Using this list we go through every possible time frame throughout the list and add the values of all the corresponding timestamps. The start date of the interval and the sum of weights are stored and compared to the next time frame. If the new sum of weights is higher, then store the new sum and the start date of the corresponding time frame. Otherwise compare the next time frame. Finally the suggested time frame will be the start and end date of the interval of length l with the highest sum of document weights. Pseudocode for this is given in Algorithm 4.

Algorithm 4 Estimating the timeframe

Input: List with the scores for every timestamp $\mathcal{W}$, max timeframe length l
Output: Start date of estimated interval d

```
1:  $max \leftarrow 0$
2:  $currentStart \leftarrow 0$
3:  $d \leftarrow 0$
4:  **for** $v \in \mathcal{W}$ **do**
5:      $Score \leftarrow \sum_{i=currentStart}^{currentStart+l} v_i$
6:      **if** $Score > max$ **then**
7:          $max = Score$
8:          $finalStart = currentStart$
9:      $currentStart + +$
10: **Return** $d$
```

Chapter 6

Experimental Setup

In this chapter we will describe our experimental setup. In Section 6.1 we will describe the different datasets we derived using different preprocessing techniques. Then, in Section 6.2 we will describe the different parameters that can influence the results. In Section 6.3 we define our evaluation metrics. Finally, in Section 6.4 we will describe how we conducted our experiments.

6.1 Datasets

For our experiments, we used **The New York Times Annotated Corpus**. This is a document collection which contains over 1.8 million articles written and published by the New York times between 1/1/1987 to 6/19/2007 and its corresponding metadata. The text in the corpus is formatted in the *News Industry Text Format* (NITF) developed by the International Press Telecommunications Council. NITF is an XML specification that provides a standarized representation for the content and structure of news articles. The New York Times Annotated corpus also provides Java code for extracting information from the news articles.

From the **The New York Times Annotated Corpus** we derived seven different data sets, spanning either one or ten years, generated using three different preprocessing techniques. An overview of the datasets before preprocessing are listed in Table 6.1 and our resulting datasets after preprocessing are listed in Table 6.2. Here, the NYT data sets was generated using POS tagging for stop word removal. Just as in Lappas et al. [14] we chose to only keep verbs, nouns and adjectives. A comparison of the NYT10 used in [14] and our NYT10 dataset can be seen in Table 6.3. Regarding the ”Keywords”-data sets, we used TF-IDF to identify and filter out stop words. Here we chose to only keep the 20% (keyword density of 0.2) most significant terms in each document. This fraction was set by looking at the occurrences of obvious stop words, such as ”a”, ”to”, ”are”, ”this” etc., versus various levels of keyword density. The NYT10-06d dataset was created by filtering out the 60% most significant terms in the NYT10 dataset by using TF-IDF. The last preprocessing technique we experimented with was Named Entity Recognition. Intuitively,

Year(s)	#Docs	# Distinct Terms	# avg Distinct Terms per Document
1987-1996	887,556	8,033,464	272
1987	106,103	1,544,019	245

Table 6.1: Dataset before preprocessing.

Dataset	#Docs	#Days	# Distinct Terms	# avg Distinct Terms	Stopword Removal
NYT1987	106,103	365	224,338	168	POS tagging
NYT10	887,553	3653	692,090	168	POS tagging
NYT10-0.6d	887,553	3653	692,008	174	POS tagging & TF-IDF
Keywords1987	106,103	365	242,018	105	TF-IDF
Keywords10	887,553	3653	750,429	110	TF-IDF
NamedEntities1987	106,103	365	152,828	33	NER-tagging
NER-Keywords1987	106,103	365	238,937	76	NER-tagging

Table 6.2: Description of datasets used.

named entities, such as persons, locations and organizations, are imperative when describing events. Thus using named entities to approximate a document's time of origin could prove to be beneficial. In addition to make a data set only consisting of named entities, we also made a data set (NER-Keywords1987) by combining named entities and keywords. In this data set we extracted all named entities from the documents and then filtered out the top 10% most significant terms from the remaining text by using TF-IDF.

6.2 Parameters

When running the algorithm and trying to find the optimal setup there are a few parameters to be tweaked and tested. To be able to receive the best results possible these parameters should be tuned and optimized.

6.2.1 Top-k Value

The number of documents selected by the Jaccard similarity function is a parameter that may have a lot of impact on the result. We have chosen to call the parameter k, where k documents are selected from the test set which are considered to be the most similar compared to the query document. Our assumptions are that the k should not be too large and values between 5-100 should be tested.

Dataset	#Docs	#Days	# Distinct Terms
NYT10 from [14]	665,741	3650	1,036,204
NYT10 in our implementation	887,553	3653	692,090

Table 6.3: Comparison of the NYT10 dataset used in [14] and NYT10 used in our implementation.

6.2.2 Time Frame Length

The time frame length is the length of the interval in which we estimate the query document to have been published in. Naturally a longer time frame gives a higher probability for the algorithm to produce correct results. As a default the time frame for a dataset over one year is one month and the time frame for a dataset over 10 years is one year. This means that randomly guessing time intervals should produce respectively $100 * (1/12) = 8.33\%$ and $100 * (1/10) = 10\%$ accuracy. Other time frames should also be tested for evaluating the results of the algorithm e.g. three months over ten years.

6.2.3 Document Weighting

How the different weights for each document is computed is described in Section 5.5. Choosing between these may have a lot of impact on the end result and can be interesting to compare. From [14] the burstiness weighting is used and we want to see how the other proposed weighting methods compare to this one.

6.2.4 Threshold

The threshold is a limit as to how certain the system needs to be before assigning a time frame to a document. The value of the threshold is defined as a real number between 0 and 1. The way that we use the threshold is given in Equation 6.1. Here the the fraction of the scores within the estimated interval over all the scores assigned over the timeline needs to be larger than the given threshold value.

$$\frac{scores(interval)}{scores(timeline)} > Threshold \tag{6.1}$$

6.3 Evaluation Metrics

In this section we will define the metrics that our evaluation is based upon. These are respectively the *accuracy*, *mean error* and *fraction of dataset estimated*.

6.3.1 Accuracy

The program gives a time frame in which it estimates the query document to have been published within. This is either correct, if the query document is in fact published in the estimated time frame, or wrong if it is published any time other than within the time frame. The number of times the program estimates correct divided by the total number of documents to be dated gives the accuracy shown in equation 6.2.

$$Accuracy = \frac{correct}{correct + wrong} \tag{6.2}$$

Here *correct* is the number of documents where the time frame is correctly estimated and *wrong* is the number of wrongly estimated documents.

6.3.2 Mean Error

The accuracy gives a kind of black and white image of how the algorithm performs. It is only giving us information about when the program either does something right or wrong. When the program is not able to estimate the right time frame for a query document it is interesting to know how much off it is. Does the program miss by just a few days, or is it totally off? To get some insight to this we calculate the *mean error* metric which says something about how many days before or after the estimated time frame the query document is actually written. The results of all the wrongly estimated documents are added up by how far they are off and divided by the number of wrong estimates as given in Equation 6.3.

$$MeanError = \frac{\sum_{i=0}^{n} |publicationDate_i - (intervalStart_i + \frac{l}{2})| - \frac{l}{2}}{n} \tag{6.3}$$

Here i is a document which publication date is not in the estimated time frame and n is the total number of wrongly estimated documents. The *publicationDate*$_i$ is the query document's actual publication date, *intervalStart*$_i$ is the start date of the estimated time frame and l is the length of the interval.

6.3.3 Fraction of Dataset Estimated

This evaluation metric is only used when the tests for a given threshold is run. When setting a threshold for the level of certainty the algorithm requires before estimating an interval it is interesting to know the number of documents that the algorithm renders inconclusive. This is why we decided to keep track of the fraction documents that are being dated when using the threshold. This also gives an idea of the trade off for a higher accuracy.

6.4 Experiments

To our understanding there are four main factors that can influence the accuracy of the system. These are the value of the k-parameter, the preprocessing technique, the weighting scheme used and the threshold value. In order to test the impact that different values for the k-parameter has on the system, we simply ran the method using different values for k, ranging from low values to high values. In addition we also ran experiments to see the k-parameter's degree of impact on the different data sets, as well as k's impact when using different weighting schemes. Since completing a run takes a lot of time, we decided that testing enough values for k to find a decisive break point (the optimal value) was not feasible regarding the time-constraints of our thesis. Although we did not run experiments for finding an optimal k it is interesting to see how large values for k performs compared to smaller values. Therefore, we chose to do runs with k varying from 5 to 30 with intervals of 5 and then one run for $k = 100$. Regarding the threshold, we ran tests with threshold values ranging from 0 to 1. Here we reported the accuracy and the fraction of the dataset that was estimated. Finally we ran tests with four different time frame lengths in order to evaluate how our system performs on different levels of granularity. The time frame lengths we ran tests on was respectively 1 month, 3 months, 6 months and 1 year. In all of our runs we used 10-fold cross validation, where 10% of the documents served as documents with unknown timestamps and the remaining 90% of the documents served as the reference corpus. We also ran these tests on the different datasets constructed by using different preprocessing techniques as well as with the different approaches to computing the weights as described in Section 5.5.

Chapter 7

Results and Evaluation

In this chapter we will present the results from our experiments. These include test runs with the different datasets, weighting methods, time frame lengths, threshold and different numbers for k when choosing the k most similar documents. In Section 7.1 we will present an overview of our results using the evaluation metrics defined in the previous chapter. Then, in Section 7.2 we will evaluate and reason about how different parameters affect the results. Finally, in Section 7.3 we will discuss the influence of different preprocessing techniques.

7.1 Overview

time frame Length	NLLR [9]	MaxEnt [2]	BurstySimDater [14]	Burstiness	Jaccard Scores
1 month	18	-	23.4	18.1	23.2
3 months	24	-	32	26.3	31.7
6 months	25	36	40	33.5	39.3
1 year	38.4	48.6	49.8	42.0	49.2

Table 7.1: Accuracy (%) for NYT10 dataset.

As we see from Table 7.1 the results from the previous *burstySimDater* is noticeably better than our method for estimating the publication date of a query document. One possible explanation for this is the difference in the datasets shown in Table 6.3. The fact that our dataset is 33.3% larger may have a lot of impact on the final result. Given that we do not know which documents are removed we can not say this for sure. For some documents it is more difficult to estimate the publication date, using a content-based algorithm, than for others due to their lack of content and relevant temporal features. If the majority of the removed documents fall into this category a result with higher accuracy is not surprising. Other factors that may apply is the preprocessing of the datasets and/or different restrictions for finding the bursty intervals e.g. the baseline. When running through the tests of

Dataset	Weighting	Accuracy
NYT1987	Burstiness	31.9%
NYT1987	JaccScores	34.9%
Keywords1987	Burstiness	31.8%
Keywords1987	JaccScores	34.0%
Keywords1987	Top-k	31.0%
NamedEntities1987	Burstiness	21.6 %
NamedEntities1987	JaccScore	23.6 %
NER-Keywords1987	Burstiness	25.9 %
NER-Keywords1987	JaccScores	27.4 %

Table 7.2: Results from running the different dating algorithms on different datasets with $k = 10$ and $l = 30$.

Dataset	Weighting	Accuracy
NYT10	JaccScores	49.2%
NYT10	Burstiness	42.0%
Keywords10	JaccScores	49.5%
Keywords10	Burstiness	45.3%
NYT10-0.6d	Burstiness	42.9%

Table 7.3: Results from running the different dating algorithms on different datasets with $k = 10$ and $l = 365$.

different datasets, preprocessing methods and weighting techniques we discovered that, in our case, the Jaccard weighting score gave better results than burstiness. The comparison can be seen in Table 7.4. When using the Jaccard weighting score on the *Keywords10* dataset with an interval of 1 year and $k = 20$ we received the best results, and it also slightly outperformed the *burstySimDater* from [14].

7.1.1 Accuracy

The accuracy is computed as the fraction of correct estimations over the total number of estimated documents, as described in section 6.3.1. Table 7.2 and 7.3 list the results that we achieved for the different datasets, described in Table 6.2, with $k = 10$ using two different weighting techniques. The complete list of results for different values of k can be found in Table 8.2. In addition to running experiments for different values of k, we also wanted to see how our implementation performed regarding different values for l, e.g. the estimated time frame length. Figure 7.1 shows the results that we achieved for this experiment. The percentage points is computed as the achieved accuracy subtracted by the expected accuracy of a random guess. The red column in the figure represents the results achieved by Kotsakos et al. [14], which we chose to include for comparison.

Figure 7.1 Percentage points achieved for $l = \{1 \text{ month}, 3 \text{ months}, 6 \text{ months}, 12 \text{ months}\}$ over 10 years.

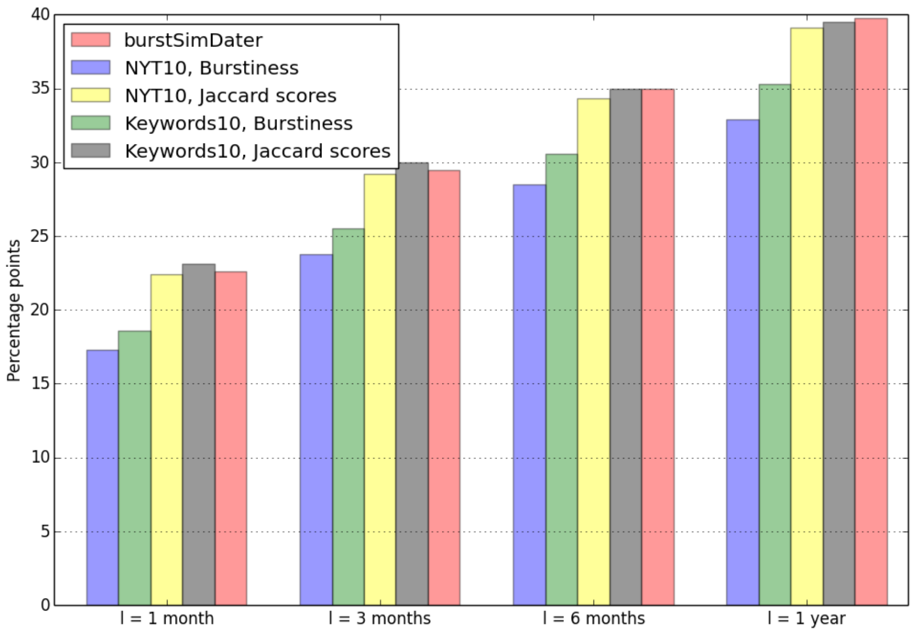
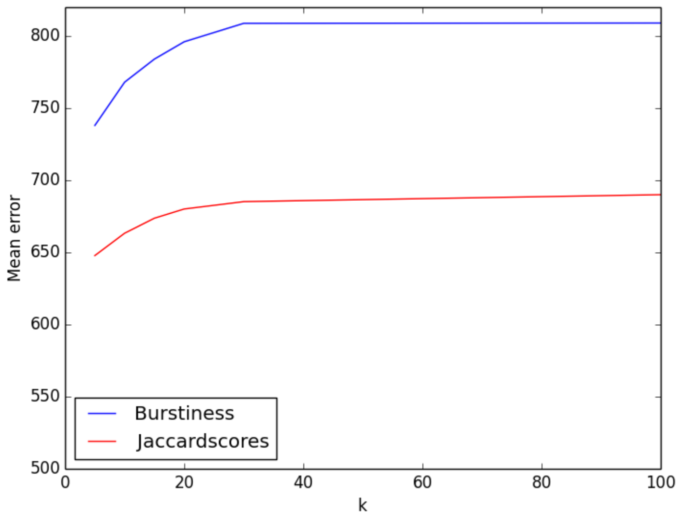


Figure 7.2 Mean error using different weighting schemes.



7.1.2 Mean Error

To be able to see the development for the mean error over the different values of k for the burstiness and Jaccard score weighting methods a visualization is given in figure 7.2. The plots correspond to the runs that were done on the *Keywords10* dataset with $l = 365$. The mean error is calculated for the documents which the algorithm fails to estimate an interval in which it is published and show the average number of days the algorithm misses by.

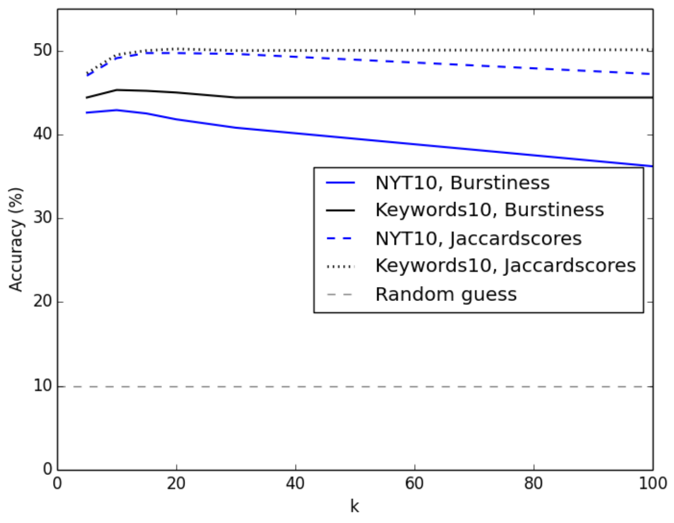
7.2 Parameters

In this section we will discuss and reason about how the parameters explained in Section 6.2 affected the results.

7.2.1 Top-k Value

As described in Section 6.2.1 the k-parameter is the number of the topmost similar documents from the reference corpus that are used for estimating the time interval. Figure 7.3 shows the plot of the reported accuracy using different values for k. Here we ran tests with $k = 5$, $k = 10$, $k = 15$, $k = 20$, $k = 30$ and $k = 100$. As you can see in the plot, the k-value had a larger impact for the NYT10-dataset than for the Keywords10-dataset. As seen in Table 6.2, from Section 6.1, the NYT10-dataset contains 692,090 distinct terms and each document contains an average of 168 distinct terms, while the Keywords10-dataset contains 750,429 distinct terms with an average of 110 distinct terms in each document. This means that each document in the Keywords10-dataset on average contains a smaller

Figure 7.3 Comparison of the accuracy for different values of k (number of most similar documents selected).



fraction of the vocabulary than in the NYT10-dataset. Additionally, the terms used to represent each document in the Keywords10-dataset is selected using TF-IDF and thus the documents in this dataset are bound to be more unique than in the NYT10-dataset, where the documents are filtered by POS tagging. Hence, it makes sense that the similarity value for the k-most similar documents for larger values of k are typically lower in the Keywords10-dataset and will therefore have a lower impact on the final result. Figure 7.3 consitutes this by showing that the accuracy for the NYT10-dataset drops as the value for k grows, while the accuracy for the Keywords10-dataset stays the same.

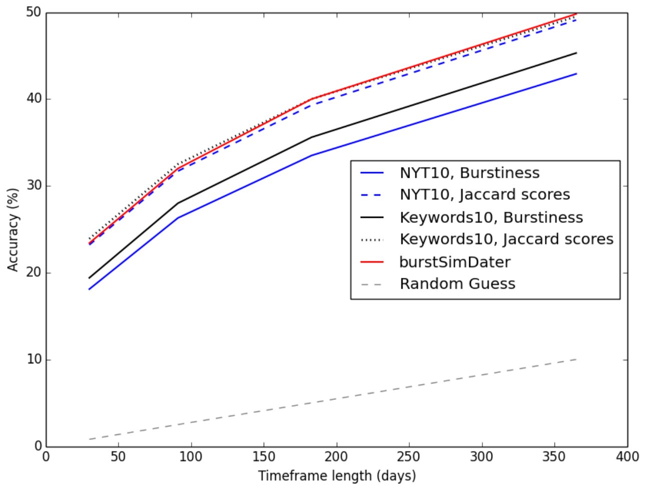
7.2.2 Time Frame Length

When running the program, the length of the time frame can vary depending on how precise the results need to be. The trade off for adjusting the interval length to span over a smaller time frame is the decrease in accuracy as we can see in Table 7.4. Figure 7.4 shows the plot of our reported accuracy from runs with different time frame lengths using different preprocessing techniques and weighing schemes. Here the red line is the results achieved by Kotsakos et al. [14]. As seen in the plot, using TF-IDF preprocessing and Jaccard similarity as document-weights produces nearly identical results as the ones presented in [14]. In addition, one can see that all the lines have very similar slopes. This means that the differences in accuracy for different time frames between the runs are typically low. Following these observations we find it unlikely that the deviation in our results and the results in [14] lies in the way we implemented the algorithm. We find it more likely that the differences could lie in the data preprocessing or in the way we computed the bursty

time frame Length	BurstySimDater [14]	Jaccard Scores
1 month	23.4	24.1
3 months	32	33.0
6 months	40	40.9
1 year	49.8	50.2

Table 7.4: Results from Kotsakos' *burstySimDater* compared with the results from out method using the *Keywords10* dataset, $k = 20$ and Jaccard score weighting scheme.

Figure 7.4 Accuracy for different values of l (time frame-length).



intervals or maybe in both. Regarding the reported accuracy for different granularities of time one can see that the accuracy grows as the time frame length increases. Obviously, this is because it is more likely that the topmost document originate from the same time frame for larger time frame lengths. In addition to increased accuracy, the achieved percentage points also grows with the time frame length. This shows that the methods perform relatively better for larger time intervals.

7.2.3 Document Weighting

As we see from Table 8.2 in the Appendix the best result was received when using the Jaccard score weighting scheme on the *Keywords10* dataset with $k = 20$. Overall our Jaccard score method received a better accuracy for all the different datasets and values for k. We wanted to compare this weighting scheme using the dataset and k that gave the best results in our experiments with the *burstySimDater* from [14] to see how our results compared to it. From Table 7.4 we see that for all the different time frame lengths, our method received a higher accuracy than the *burstySimDater*. What is interesting is that

Threshold	Accuracy	Fraction of dataset estimated
0.3	52.0 %	93.1 %
0.4	60.2 %	71.7 %
0.5	70.7 %	50.1 %
0.6	80.2 %	34.3 %
0.7	87.0 %	23.1 %
0.8	91.6 %	14.8 %
0.9	94.4 %	8.5 %

Table 7.5: Result from running the algorithm with different threshold. The parameters used are $k = 10$ and $l = 365$ and are run on the *Keywords10* dataset with the Jaccard score weighting method.

the *burstySimDater* already uses the Jaccard similarity measure to compute the top-k most similar documents but does not use this measure further. Instead the top-k documents, 10 in this case, are analyzed to find the burstiness of their terms and calculate a weight based on this calculation. In our case, we achieved better results by using the Jaccard similarity score directly to give the documents their weights. Additionally, as seen in Figure 7.2, using the summed Jaccard weights gives a lower *mean error* than the burstiness weighing scheme.

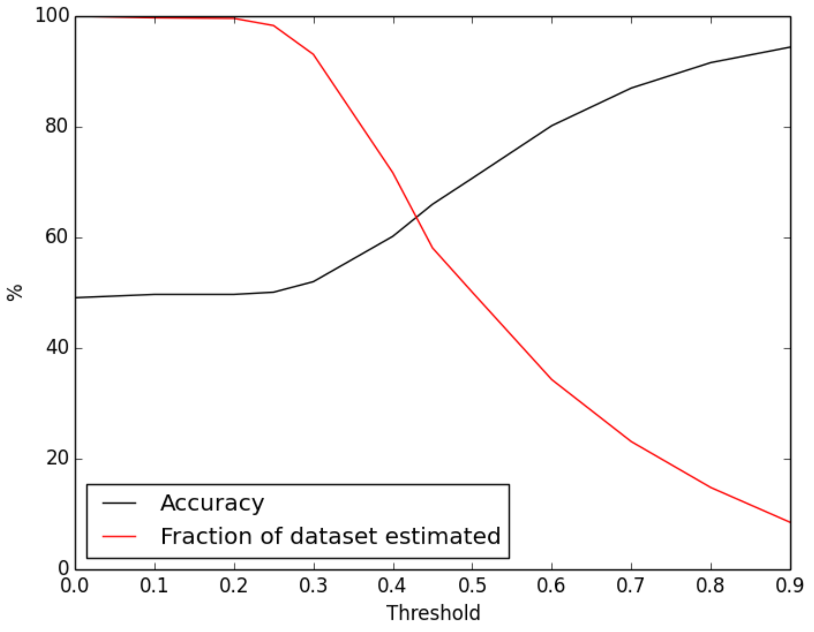
7.2.4 Threshold

When running the tests with different threshold values we decided to keep track of how many documents the algorithm discarded and how many documents the publishing date was estimated for. We represent this information as a fraction of how many documents, out of the ones to be dated, that the program decided to actually date. This means that if a document has a lower score for its estimated interval than the given threshold it is discarded and nothing happens. If the sum is above the threshold then the algorithm will return its estimated time frame for the document. The results from these tests can be seen in table 7.5. In Figure 7.5 we see how the accuracy and the fraction of the dataset for which an interval is estimated change when the threshold increases. The number of documents that are being discarded increases drastically as the criteria gets stricter. At the cost of discarding half of the results the accuracy is increased to 70.7 % and if one needs the accuracy to be over 90 % approximately 85 % of the dataset can not be dated. Finding a desired threshold really depends on the application and how important it is that the publishing date is correct. The fact that the system achieves over 90 % accuracy for large threshold values substantiates the assumption that there often is a strong correlation between the lexical similarity of documents and their origin of time.

7.3 Data Preprocessing

The different preprocessing techniques had some impact on the results. From the results presented in Table 8.2 in the Appendix we see that the dataset that scored the best among

Figure 7.5 Accuracy and fraction of dataset estimated when using different values for threshold.



the datasets spanning over 10 years was the *Keywords10* dataset. A brief sample of the results can also be seen in Table 7.3 as well as figures 7.1 and 7.3. The *Keywords10* dataset used only simple TF-IDF for stop word removal keeping 20% of the terms for each document. As comparison the *NYT10* dataset which was used in the *burstySimDater* in [14] used POS tagging to remove stop words. For the runs of the datasets spanning 1 year (1987) we hade a few more datasets to compare. As well as the *NYT1987* and *Keywords1987* we also wanted to analyze two datasets utilizing the named entity tagger, namely *NamedEntities1987* and *NER-Keywords1987*. When comparing the different runs on these datasets both the *NYT1987* and the *Keywords1987* dataset received pretty similar results.The latter of these seems to perform a bit better with the increasing of k. The scores from the *NamedEntities1987* dataset showed a much lower accuracy. The intuition behind using named entities was to capture the essence of the events described in the documents. However, as pointed out in [16], using the utility of named entities isolated can prove to be a poor idea. This is because there is a high chance that the same named entities such as locations, persons and organizations are mentioned all across the the corpus. Although two documents talk about the same entities they could very well be describing different events. Thus named entities should be used in a combination with words that describe their context. This was the idea behind testing on the NER-Keywords1987 dataset, were we used a combination between NER-tagging and TF-IDF. This showed some improvement over the purely NER-tagged dataset. However it did not perform nearly as well as the datasets where we solely used TF-IDF or POS tagging.

In short the tendency is that TF-IDF scores better than POS tagging when used as a preprocessing technique for datasets to be dated by a content-based dating algorithm. We find it likely that the reason behind this is that the POS tagged documents probably contain more stop words that act as noise and can have a negative impact when running the dating algorithm. From this, one can say that the documents where we used TF-IDF probably contained more relevant information than the POS tagged documents.

Chapter 8

Conclusion and Future Work

In this chapter we will summarize our project with some concluding remarks. In Section 8.1 we will provide the conclusion for our project. Then, in Section 8.2 we will provide some suggestions regarding future work.

8.1 Conclusion

In this project we have worked towards solving the problem of incorrectness and ambiguity regarding temporal information in meta data describing documents. Our approach to this problem was to implement content based dating methods that look at the terms used to describe documents over the timeline in a reference corpus in order to estimate the dates for undated documents. This is similar to the *burstySimDater* method described in [14]. In addition we implemented our own approach only utilizing lexical similarities between documents. In order to evaluate the plausability of these methods we have performed extensive experiments testing how different factors around the algorithms itself can affect the results. In the beginning of this report we defined the following four research questions:

RQ1 : What impact does the preprocessing of the dataset have on the final result when timestamping different documents?

RQ2 : When deciding the timestamp for a document by comparing it to similar documents, how will using different weighting methods for the reference documents alter the results?

RQ3 : What will be the optimal number of documents to use for comparing when deciding a timestamp for the query document?

RQ4 : Can we achieve a fair trade off between accuracy and the fraction of the dataset estimated by introducing a threshold?

After writing the program and running the experiments it became clear to us that preprocessing the documents was extremely important. A good representation of the documents is essential both for compressing the documents to be able to run an all pairs similarity search as well as representing enough of the temporal information that is essential for the dating algorithm to do its calculations and estimate a publishing date. An optimal preprocessing technique here would be a technique that keeps a combination of terms that are "typical" for the document's time and discards the rest. As discussed earlier in the paper the all pairs similarity search is computationally heavy for large datasets. There exists a number of algorithms to make this computation faster, some utilizing approximation methods and others exploiting sort order, threshold, indexing and so on to be able to compute the exact similarity measure within reasonable time. One such method is using the MinHash-LSH which, as mentioned earlier, could be useful for other datasets where the similarities between documents are higher. In our case, when using the **New York Times Annotated Corpus** it was sufficient to adjust the representation of the dataset, using a hashmap with each distinct term as keys. To be able to use a representation like this it was important to reduce the size of the dataset first. We tried a few different methods, namely TF-IDF, POS tagging, Named Entity tagging and some combinations of these methods. Our experience is that the TF-IDF method received the better results on the whole dataset, but when running test with documents from only 1987, the POS tagging method scored slightly better. After compressing the documents and running a Jaccard similarity search on the dataset this information was stored in a database. This made it easy to extract the top-k most similar documents compared to a given query document. Having this information stored also meant that the dating algorithm could run tests on a large amount of query documents much faster and was, in our case, even necessary to receive enough results to evaluate the different dating methods in reasonable time.

As well as precomputing and storing the Jaccard similarities to save time when running the algorithm we decided to do the same for the bursty intervals of the terms. Even though the terms which bursty intervals need to be computed is a small number for each query document the whole dataset needs to be considered for being able to do so. This adds up over time and eventually makes the program a lot slower when computing this on the fly, since the mapping between terms and timestamps needs to be kept in memory. After computing the bursty intervals for every term in the corpus we stored each bursty interval with timestamps representing the start and the end of all the intervals in which the term was bursty. This made it easy to check if a document had a publishing date inside one of these intervals, thus reducing the running time even more.

Naturally, the different weighting schemes had a lot of impact on the results we received. The one coming out with the highest scores both regarding accuracy and mean error was the Jaccard scores weighting scheme. This method outperformed all of our other implementations as well as the *burtsySimDater* from [14]. When given the optimal k and preprocessing of the corpus it dated more than 50% of the documents correct for time intervals of one year with a dataset spanning 10 years. When decreasing the time interval to achieve a more precise timestamp the accuracy, naturally, decreases. Still the Jaccard scores weighting scheme received the highest accuracy for all the different intervals tested.

Regarding the third research question, we did not find a decisive optimal value for k (i.e. the number of documents used for comparing). However we did discover that it was, in our case, slightly better to use small values. To our understanding it is intuitive that as the number of documents used for comparing grows there is a higher chance that intervals outside the desired date gets higher scores, which in turn results in a higher probability for estimating the wrong time frame. However, the contribution of the most lexical similar documents will be of a much higher degree than the ones for documents that are ranked lower. Hence, one would probably need to do tests for even higher values for k to see a drastic reduction in accuracy.

By introducing a confidence threshold when estimating a timestamp the test runs show that it is possible to receive a higher accuracy if the algorithm discards documents which are typically more ”unsure”. If high accuracy is more important than timestamping the whole dataset this is a useful function. The trade off between discarding documents for higher accuracy can be analyzed to find an optimal threshold value for a given usage.

8.2 Future Work

When we started this project we had little practical knowledge on working with document streams and large volumes of sparse high-dimensional data. As a result, a lot of time was spent trying and failing when implementing the system. If we were to start the project from scratch with the knowledge we have gained through the process there are some things that we definitely would have done in a different way. In this section we will give some propositions regarding future work on document dating. Hopefully these suggestions will be helpful for anyone who wants to do further work in this field.

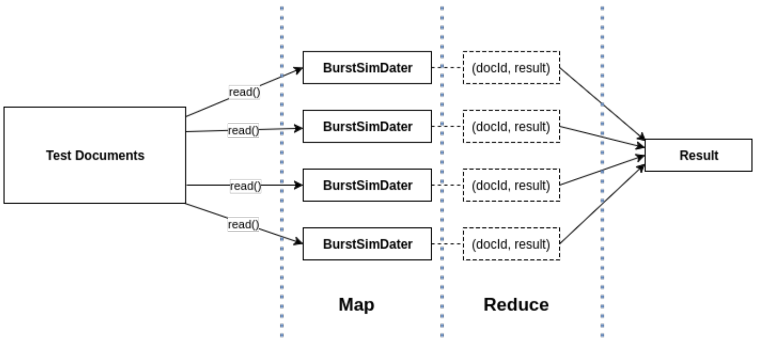
8.2.1 Parallelization

The 10-fold cross validation on our implementation was done using a for-loop dating one document at a time. However, the validation process has no loop-carried dependencies regarding the dating process of the different documents. Thus, it could be beneficial to date the test-documents in parallel. One way to go about this problem could be to implement the solution using the Apache Spark Framework [33] on top of the Hadoop Distributed File System [30]. The benefits of using spark is that it allows you to work with distributed collections as you would with with local ones. From the programmer’s point of view, you simply write a sequential pipeline program. However, when the program is executed, the computations are distributed among several nodes. Without going into detail, one can say that Spark allows coders that know little about parallel programming to write parallel programs. Figure 8.1 shows conceptually how a Spark implementation could improve the performance.

8.2.2 Single Document BurstDater

The *burstySimDater* method [14] takes the top-k most similar documents into consideration when dating documents. More specifically, as mentioned earlier, it takes the intersecting terms with the k most similar documents and then finds the interval with the highest number of terms are simultaneously bursty. As a concept we propose the *SingleDocument-BurstDater* method. Unlike the *burstySimDater* method, this method does not directly

Figure 8.1 Spark could improve the performance by parallelizing the cross-validation.



compare documents. Instead, it only looks at terms within single documents and then finds the time frame of a given length with the highest number of simultaneously bursty terms. If this method where to produce as good results as the *burstySimDater* method, it would be a more efficient approach. This is because this method would solely rely on finding bursty intervals, which can be done in linear time [27], opposed to solving the all-pairs similarity problem, which can be challenging to do precisely and efficiently for large corpora. A suggestion to how we would have solved this task is given with pseudocode in Algorithm 5.

Algorithm 5 Single Document BurstDater

```
Input: bursty intervals $\mathcal{B}$, query document $q$, max time frame length $l$
Output: time frame of q
1: $W_{\mathcal{S}} \leftarrow \varnothing$
2: for $t \in timeline$ do
3:     $w_I \leftarrow 0$
4:     for $x \in q$ do
5:         $w_t \leftarrow w_t + |\{I \in B(x) : t \in I\}|$
6:     $W_{\mathcal{S}} \leftarrow W_{\mathcal{S}} \cup \{w_t\}$
7: $A_{\mathcal{S}} \leftarrow (t \in timeline, W_{\mathcal{S}})$
8: $\mathcal{I} \leftarrow$ getMaxSumInterval$(A, l)$
9: Return $\mathcal{I}$
```

8.2.3 Document Clustering

In our solution, we chose to represent documents using the BOW (Bag of Words) approach. An obvious disadvantage of using this approach is that it cannot accurately represent the meaning of documents since it ignores the semantic relationship among words. Additionally the BOW approach will result in high dimensionality and data sparsity when dealing with large corpora. Wei et al. [32] proposes a method that uses a modified similarity measure based on WordNet for word sense disambiguation and lexical chains to capture the main theme of texts. Experiments show that this method can estimate the true number of clusters in document collections as well as generating topic labels that are good indicators for recognizing and analyzing the content of clusters. Such cluster techniques could prove to be highly relevant regarding document timestamping. One obvious application regarding our problem would be to group the documents in the reference corpus in their respective clusters. Then, when you are to find the date of an ”unknown” document, you can first find the cluster it belongs to and then find its top-k most similar documents within the cluster. This way the test document only needs to be compared with a small portion of the reference corpus. However, one should keep in mind that when using these kind of methods, which uses lexicons such as WordNet, it is important that the relationship between words are thoroughly represented in the lexicon.

Bibliography

[1] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140, 2007.

[2] N. Chambers. Labeling documents with timestamps: Learning from their time expressions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 98–106, 2012.

[3] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[4] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 18–24, 2004.

[5] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 301–312, 2003.

[6] R. Feldman and J. Sanger. *THE TEXT MINING BOOK Advanced Approaches in Analyzing Unstructured Data*. Cambridge, New York: Cambridge University Press, 2007.

[7] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[9] d. F. Jong, H. Rode, and D. Hiemstra. Temporal language models for the disclosure of historical text. 2005.

[10] N. Kanhabua, R. Blanco, K. Nørvåg, et al. Temporal information retrieval. *Foundations and Trends® in Information Retrieval*, 9(2):91–208, 2015.

[11] N. Kanhabua and K. Nørvåg. Improving temporal language models for determining time of non-timestamped documents. In *International Conference on Theory and Practice of Digital Libraries*, pages 358–370, 2008.

[12] N. Kanhabua and K. Nørvåg. Using temporal language models for document dating. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 738–741, 2009.

[13] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 91–101, 2002.

[14] D. Kotsakos, T. Lappas, D. Kotzias, D. Gunopulos, N. Kanhabua, and K. Nørvåg. A burstiness-aware approach for document dating. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1003–1006, 2014.

[15] W. Kraaij. Variations on language modeling for information retrieval. 2004.

[16] G. Kumaran and J. Allan. Text classification and named entities for new event detection. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 297–304, 2004.

[17] T. Lappas, B. Arai, M. Platakis, D. Kotsakos, and D. Gunopulos. On burstiness-aware search for document sequences. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 477–486, 2009.

[18] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[19] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*, chapter 3. Cambridge University Press, 2014.

[20] X. Li and W. B. Croft. Time-based language models. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 469–475, 2003.

[21] K. E. Lochbaum and L. A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing & Management*, 25(6):665–676, 1989.

[22] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

[23] A. Metwally, D. Agrawal, and A. El Abbadi. Detectives: detecting coalition hit inflation attacks in advertising networks streams. In *Proceedings of the 16th international conference on World Wide Web*, pages 241–250, 2007.

[24] M. J. Metzger. Making sense of credibility on the web: Models for evaluating online information and recommendations for future research. *Journal of the American Society for Information Science and Technology*, 58(13):2078–2091, 2007.

[25] M. A. Olson, K. Bostic, and M. I. Seltzer. Berkeley DB. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191, 1999.

[26] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, 2003.

[27] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *ISMB*, volume 99, pages 234–241, 1999.

[28] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, pages 377–386, 2006.

[29] B. Shaparenko, R. Caruana, J. Gehrke, and T. Joachims. Identifying temporal patterns and key players in document collections. In *Proceedings of the IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications (TDM-05)*, pages 165–174, 2005.

[30] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10, 2010.

[31] R. Swan and D. Jensen. Timemines: Constructing timelines with statistical models of word usage. In *KDD-2000 Workshop on Text Mining*, pages 73–80, 2000.

[32] T. Wei, Y. Lu, H. Chang, Q. Zhou, and X. Bao. A semantic approach for text clustering using wordnet and lexical chains. *Expert Systems with Applications*, 42(4):2264–2275, 2015.

[33] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

Appendix

Extended Results

Dataset	Timeframe Length	k	Weighting	Accuracy
NYT1987	1 month	5	Burstiness	31.7%
NYT1987	1 month	10	Burstiness	31.9%
NYT1987	1 month	15	Burstiness	31.4%
NYT1987	1 month	20	Burstiness	30.6%
NYT1987	1 month	30	Burstiness	29.1%
NYT1987	1 month	100	Burstiness	23.6%
NYT1987	1 month	5	JaccScores	33.6%
NYT1987	1 month	10	JaccScores	34.9%
NYT1987	1 month	15	JaccScores	34.7%
NYT1987	1 month	20	JaccScores	34.4%
NYT1987	1 month	30	JaccScores	33.9%
NYT1987	1 month	100	JaccScores	31.8%
Keywords1987	1 month	5	Burstiness	31.5%
Keywords1987	1 month	10	Burstiness	31.8%
Keywords1987	1 month	15	Burstiness	31.9%
Keywords1987	1 month	20	Burstiness	31.6%
Keywords1987	1 month	30	Burstiness	31.0%
Keywords1987	1 month	100	Burstiness	28.5%
Keywords1987	1 month	5	JaccScores	32.8%
Keywords1987	1 month	10	JaccScores	34.0%
Keywords1987	1 month	15	JaccScores	34.2%
Keywords1987	1 month	20	JaccScores	34.2%
Keywords1987	1 month	30	JaccScores	34.0%
Keywords1987	1 month	100	JaccScores	32.2%
Keywords1987	1 month	30	Top-k	31.1%
NamedEntities1987	1 month	10	Burstiness	21.6 %
NamedEntities1987	1 month	10	JaccScores	23.6 %
NER-Keywords1987	1 month	10	Burstiness	25.9 %
NER-Keywords1987	1 month	10	JaccScores	27.4 %

Table 8.1: Results from running the different dating algorithms on different datasets

Dataset	Timeframe Length	k	Weighting	Accuracy
NYT10	1 year	5	Burstiness	41.9%
NYT10	1 year	10	Burstiness	42.0%
NYT10	1 year	15	Burstiness	41.3%
NYT10	1 year	20	Burstiness	40.6%
NYT10	1 year	30	Burstiness	39.5%
NYT10	1 year	100	Burstiness	35.0%
NYT10	1 year	5	JaccScores	47.0%
NYT10	1 year	10	JaccScores	49.2%
NYT10	1 year	15	JaccScores	49.7%
NYT10	1 year	20	JaccScores	49.8%
NYT10	1 year	30	JaccScores	49.7%
NYT10	1 year	100	JaccScores	47.3%
Keywords10	1 year	5	Burstiness	44.4%
Keywords10	1 year	15	Burstiness	45.2%
Keywords10	1 year	10	Burstiness	45.3%
Keywords10	1 year	20	Burstiness	45.0%
Keywords10	1 year	30	Burstiness	44.4%
Keywords10	1 year	100	Burstiness	44.4%
Keywords10	1 year	5	JaccScores	47.3%
Keywords10	1 year	10	JaccScores	49.5%
Keywords10	1 year	15	JaccScores	50.0%
Keywords10	1 year	20	JaccScores	50.2%
Keywords10	1 year	30	JaccScores	50.11%
Keywords10	1 year	100	JaccScores	50.1%
NYT10-0.6d	1 year	5	Burstiness	42.6%
NYT10-0.6d	1 year	10	Burstiness	42.9%
NYT10-0.6d	1 year	15	Burstiness	42.5%
NYT10-0.6d	1 year	20	Burstiness	41.8%
NYT10-0.6d	1 year	30	Burstiness	40.8 %
NYT10-0.6d	1 year	100	Burstiness	36.2%
NYT10-0.6d	1 year	5	JaccScores	47.0 %
NYT10-0.6d	1 year	10	JaccScores	49.1 %
NYT10-0.6d	1 year	15	JaccScores	49.7 %
NYT10-0.6d	1 year	20	JaccScores	49.7 %
NYT10-0.6d	1 year	30	JaccScores	49.6 %
NYT10-0.6d	1 year	100	JaccScores	47.2 %

Table 8.2: Results from running the different dating algorithms on different datasets