Kjetil Kværnum

Incremental Update of Document Timestamping Models

Master's thesis in Computer Science

Supervisor: Kjetil Nørvåg

June 2020

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Kjetil Kværnum

Incremental Update of Document Timestamping Models

Master's thesis in Computer Science
Supervisor: Kjetil Nørvåg
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Estimating the creation time of documents is a task that requires an annotated dataset, and the performance of an estimation model is often closely tied to the size of the dataset used. The idea of an automatic way of increasing the size of the training data, could help improve the performance of existing document dating methods. It could also open up for new opportunities in the field of document timestamping, where there previously did not exist an annotated dataset of sufficient size. In this thesis, a new approach to automatically updating a dating model is suggested. The idea is to incrementally add previously unseen documents with an estimated label, if the confidence of the prediction meets a certain threshold. During the experiments in this thesis, an existing approach to document dating is implemented and modified to allow the model to be incrementally updated. Then the results are presented and evaluated in comparison to an ideal expansion of the training data. In the thesis, I also experiment with the usage of regression-based document timestamping techniques. These experiments include the usage of different feature groups and regression methods to find the best performing regression methods, and measure the impact of each feature group.

Sammendrag

Å estimere opprettelsestiden for dokumenter er en oppgave som krever et annotert datasett, og ytelsen til en estimeringsmodell er ofte tett knyttet til størrelsen på datasettet som brukes. Ideen om en automatisk måte å øke størrelsen på treningsdataene på, kan bidra til å forbedre ytelsen til eksisterende dokument-dateringsmetoder. Det kan også åpne for nye muligheter innen datering av dokumenter, der det tidligere ikke eksisterte et annotert datasett av tilstrekkelig størrelse. I denne oppgaven foreslås en ny tilnærming til automatisk oppdatering av dateringsmodeller. Tanken er å inkrementelt legge til tidligere usettede dokumenter med en estimert etikett, hvis tilliten til prediksjonen oppfyller et visst krav. Under eksperimentene i denne oppgaven implementeres og modifiseres en eksisterende tilnærming til dokumentdatering for å gjøre det mulig å oppdatere modellen inkrementelt. Deretter presenteres og evalueres resultatene i sammenligning med en ideell utvidelse av treningsdataene. I oppgaven eksperimenterer jeg også med bruk av regresjonsbaserte tidsstemplingsteknikker for dokumenter. Disse eksperimentene inkluderer bruk av forskjellige grupper med egenskaper og regresjonsmetoder for å finne regresjonsmetodene med best ytelse, og måle virkningen av hver gruppe med egenskaper.

Preface

This thesis is written by Kjetil Kværnum as the product of my final semester of the 5 year Master's degree programme in Computer Science with a specialization in Databases and Search, at the Department of Computer Science, at the Norwegian University of Science and Technology.

I would like to sincerely thank my supervisor, Professor Kjetil Nørvåg, for all the feedback, help and guidance he has provided during the process of writing this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter will give an introduction to the thesis, outlining the motivation behind the contributions in Section 1.1. A clear definition of the tasks at hand is described in Section 1.2, and the proposed research questions are shown in Section 1.3. The chapter will also summarize the contributions to the field in Section 1.4, and outline the structure of the remainder of the report in Section 1.5.

1.1 Motivation

Most of the models previously used for automatically dating documents require the use of an already annotated dataset used for training. A very common problem when using such datasets is that it often requires a lot of manual work to create them, and the quality of the resulting models are very dependent on the size of the training set. If there existed an approach to automatically increase the set of documents used to train the model, this could potentially improve the accuracy of the system, as generally more training data will lead to higher performance. Such an automated system could be used to improve the performance of current timestamping approaches, as well as open up new opportunities for document dating, where the dataset was previously of insufficient size.

Most of the recent approaches to document timestamping is based on a form of classification where the time period is split into partitions and the task is to predict the correct partition. The most intuitive way to approach the task is by regression as the timestamps exist on a continuous spectrum. While this has proven to be a harder task than the classification approach, it would be more precise as you get an exact timestamp and not a partition of a pre-determined size.

1.2 Problem Formulation

This thesis presents two problems: the limits of manually annotating datasets used to train models and the lack of recent regression-based approaches to document dating.

The first problem is important because the size of the training set can limit the performance of dating approaches significantly, and manually annotating data can be very tedious and time-consuming. This thesis will explore the possibility of incrementally updating the prediction model using the prediction of unseen examples with high confidence. The approach is inspired by pseudo-relevance feedback used in information retrieval to improve the performance of a document search.

The second problem about regression-based approaches to document timestamping could be useful because it could offer an intuitive and precise approach to timestamping. The approach can be especially useful for tasks where the required granularity is very fine and the traditional approach of finding a partition with the size ranging from months to years is not sufficient.

1.3 Research Questions

As described in the problem formulation, these are the research questions that will be answered in this thesis:

RQ 1. How will the performance of a document dating model be impacted by incrementally increasing training set size?

RQ 2. What will be the performance of a regression based document timestamping model?

To answer the first question, I have implemented a document dating method using Normalized Log-Likelihood Ratio, described by de Jong et al. (2005) [6], and modified it to allow the incremental update of the model.

The second question will be answered by exploring different combinations of features and regression techniques and compare the results using a variety of metrics.

1.4 Contributions

The contributions provided in this thesis can be summarized with the following:

- Implementation of NLLR document dating [6].
- Implementation of new technique for incrementally updating the document timestamping model.
- Extracting features for regression-based timestamping.
- Implementing different regression techniques.

1.5 Project Outline

Chapter 2 - Related Work: Describes some of the approaches made in research fields that are relevant to the work in this thesis.

Chapter 3 - Preliminaries: Presents a description of the techniques and concepts used for the implementations presented in this thesis.

Chapter 5 - Approach: Provides an explanation of the approach used to implement the systems described in the thesis.

Chapter 6 - Experimental Setup: Describes the dataset used for the experiments, and explains the setup of the experiments made in the thesis in detail.

Chapter 7 - Results & Evaluation: Presents and evaluates the results from the experiments.

Chapter 8 - Conclusion & Future Work: Presents a conclusion and a summary of the thesis, and provides some ideas for future work relevant to the thesis.

Chapter 2

Related Work

This chapter will cover previous work in the fields relevant to this project. The most relevant fields are temporal information extraction, covered in Section 2.1 and pseudo-relevance feedback, covered in Section 2.2.

2.1 Temporal Information Extraction

Temporal information Extraction (T-IE) is an area of research related to the field of Information retrieval, and specifically focuses on utilizing explicit and implicit temporal information to improve the performance of the retrieval [7].

The simplest form of T-IE is the extraction of *temporal expressions*. Temporal expressions can be classified as explicit, implicit or relative, as described by Schilder and Habel (2003) [16]. Explicit temporal extractions are expressions that can be directly mapped to a specific point or interval in time, for example ”June 10, 2020”. An implicit temporal expression is less precise, but still refers to an independent point or period in time, for example ”Christmas Eve 2019” or even expressions such as ”the Hong Kong protests” which carry inherent temporal properties and can be mapped to a specific period in time. A relative temporal expression is a temporal expression that can be mapped to a specific point or period in time, only with the help of another, explicit or implicit, temporal expression that it exists in reference to. An example of a relative temporal expression could be ”last month”, which only can be mapped to a specific time period if the time that the expression was made is known.

The most relevant form of T-IE for this thesis is content-based methods for extraction of the *document creation time*. One of the first approaches in this field was proposed by de Jong et al. (2005) [6], and was based on the usage of temporal language models. The goal of this approach was to use the difference in word use statistics over time to determine a likelihood of a document being generated by a certain temporal language model. This was done by partitioning the training corpus based on publication dates, for example one partition for each month or year. A language model was then built for each of these partitions, and for the query document. Then the approach uses a normalized log-

likelihood ratio (NLLR) [10] to calculate the similarity between the language model of the query document and the language models for each time partition.

This approach was later extended by Kanhabua and Nørvåg (2008) [8]. Their most significant addition was the introduction of *temporal entropy*. Temporal entropy is motivated by the way *term entropy* can be used to identify terms that are more suitable to distinguish between different *documents*. Instead of distinguishing between documents, the goal of temporal entropy is to find terms that are better suited to distinguish between different *time periods*. Kanhabua and Nørvåg also proposed other potential methods of improvement, including semantic-based preprocessing techniques and finding temporal trends using Google Zeitgeist.

Another, significantly different, approach to content-based document dating was suggested by Kotsakos et al. (2014) [9], where they used the burstiness of the terms to estimate the publication date. The approach is based on the idea that documents discussing similar events, often are created in the same time period, and that the burst intervals of the relevant terms in these documents overlap to a larger extent. The process of dating a document, d, in this approach starts with finding the k most similar documents to d. Then each of these documents are assigned a weight based on their burst interval overlap with d. Finally, the publication dates of each relevant document is assigned a weight equal to the sum of the relevant documents with that particular publication date. The predicted time interval for d is then chosen by finding the interval of an application-defined length having the maximum sum of weights.

2.2 Pseudo-Relevance Feedback

Pseudo-relevance feedback (PRF) is a method for query expansion used in traditional information retrieval to improve the performance of the retrieval. The general problem query expansion aims to counteract is that user generated queries often are too short to accurately describe the information needed. The idea of PRF is to make an assumption that the top-ranked results from the initial query are relevant, and contains terms that would help separate relevant and irrelevant documents. The idea is therefore to use these top-ranked documents to identify candidate terms for the query expansion.

The process of extracting terms for query expansion from these documents can be done in many different ways. The simplest approach is to use the term distribution of the feedback documents and extract the most frequently used terms. Cao, Guihong, et al. (2008) [3] proposed the use of a supervised learning method to select the best expansion terms.

Chapter 3

Preliminaries

This chapter will describe the techniques used in this thesis. Section 3.1 explains the different document representation models used, and Section 3.2 explains the how language models can be compared using a Normalized Log-likelihood Ratio.

3.1 Document Modelling

This section will provide a general overview of the methods used to represent textual documents in this thesis. First, the Bag of Words model is covered in Section 3.1.1. Then Paragraph Vectors are described in Section 3.1.2.

3.1.1 Bag of Words

The bag of words model for document representation is very simple and efficient, and can often achieve a surprising level of accuracy. This is why this model is the most common way to represent a document as a vector of fixed length. The model consists of a vocabulary containing all the terms in the corpus, and one document vector for each document in the corpus. The document vectors are of the same length as the vocabulary and the vector entries is linked to a word in the vocabulary. The entries can either be a binary representation of whether the word is present in the document or not, or they can represent the term frequency of the word in the document (i.e. the number of times the word occurs in the document).

Here is an example of how a bag of words model can be used to represent a corpus of documents:
Given these three documents:

D_1 : *"The weather is warm today"*

D_2 : *"Warm weather is nice"*

D_3 : *"The sun is shining today"*

After removing stopwords (i.e "the" and "is"), the resulting vocabulary may be:

$V = \{nice, shining, sun, today, warm, weather\}$

The bag of words representations of the documents would then be:

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

$\vec{D_2} = \{1,0,0,0,1,1\}$

$\vec{D_3} = \{0,1,1,1,0,0\}$

Even though the model is simple and efficient, it also has some considerable drawbacks. For large document collections with a large number of distinct terms, the vocabulary and thus the dimensionality of the document vectors will get very large and each vector will contain a lot of zero-values due to the vocabulary size being much larger than the number of distinct words in each document.

Another issue with the model is that it provides little information about the semantic meaning of the documents, as it disregards the order of the words [11]. Other document vector representation techniques has been made to combat these downsides, one of which is described in the next section.

3.1.2 Paragraph Vectors

Paragraph vectors were proposed by Le et al. (2014) [11] as a new way to represent documents by a fixed-length vector, with the goal to overcome the weaknesses of the bag of words representation mentioned earlier in Section 3.1.1.

Paragraph vectors are capable of handling input of varying length, and constructs vector representations where documents with similar semantic meanings also have similar vector representations. To achieve this the authors propose two methods: the distributed memory model and the distributed bag of words model.

When training the distributed memory model, the input paragraph, p, is mapped to a unique vector, which is then stored in a paragraph matrix D. Each word in the paragraph is also mapped to a unique vector, stored in a matrix W. The paragraph vector and the word vectors are then concatenated or averaged, and then used to predict the next word in the context. This new token can be thought of as a representation of the paragraphs topic. The process is shown in Figure 3.1.

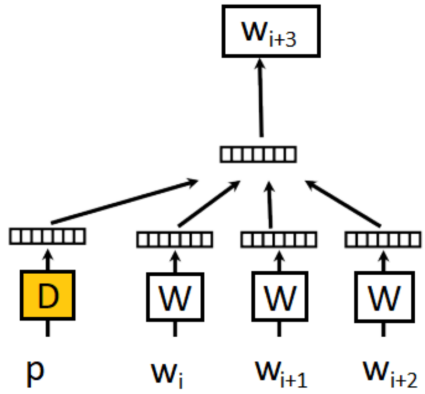

Figure 3.1: Distributed Memory version of paragraph vectors.

The distributed bag of words model ignores the word vectors in the input and instead make the model predict randomly sampled words from the paragraph, as shown in Figure 3.2. For a more detailed description of the two paragraph vector models, please refer to Le et al. (2014) [11].

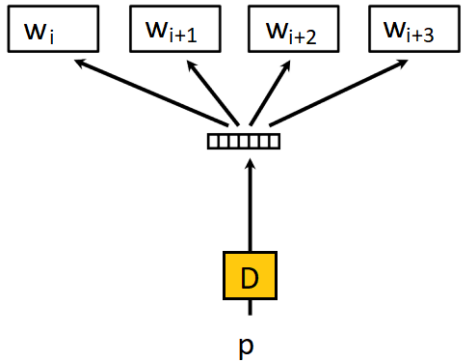

Figure 3.2: Distributed Bag of Words version of the paragraph vector model.

Dai et al. (2015) [5] built a paragraph vector model based on a Wikipedia dump and compared it to another widely used document modelling method called Latent Dirichlet Allocation (LDA) [1]. Their experiments showed that the paragraph vector model was better than LDA at measuring semantic similarity between the Wikipedia articles.

3.2 Normalized Log-likelihood Ratio

There are many ways to compare two language models, but the one that is most relevant to this thesis is a method proposed by Kraaij (2005) [10]. This method is called the normalized log-likelihood ratio (NLLR), and its formula is as follows:

$$NLLR(Q, D) = \sum_{w \in Q} P(w|Q) * log(\frac{P(w|D)}{P(w|C)})$$

where Q and D are the models to be compared, and C is the model of the entire background corpus.

As can be seen from the formula, the comparison method runs into some problems if one of the terms present in Q, is not present in P (i.e if $P(w|D) = 0$). This scenario would lead to an undefined logarithm. To avoid this problem, a smoothing method must be applied. Some of the possible smoothing approaches are linear interpolation smoothing, which was used by Kraaj (2005) [10], and Dirichlet smoothing [20], which was experimented with by de Jong et al. (2005) [6].

Chapter 4

Approach

This chapter will cover a detailed description of the approaches made in this thesis. First, in Section 4.1, the approach to the first research question about incremental updates is described. Then, in Section 4.2, the approach to the second research question about regression-based document dating is explained.

4.1 Incremental Update

This section covers the approach used in the first research question. The goal of this task was to implement the NLLR-based timestamping approach suggested by de Jong et al. (2005) [6], and implement a technique for adding new documents to the model based on a confidence threshold. The following sections explain the specific steps taken in this approach in more detail.

Section 4.1.1 covers the preprocessing of the documents. Section 4.1.2 describes how each document and the corpus as a whole were modelled. Section 4.1.3 covers the computation of the Normalized Log-Likelihood Ratio used to compare document models to the models of each time partition. Section 4.1.4 will cover how the system made a prediction of a time partition for an unseen document and how the confidence of the estimation was calculated. Lastly, Section 4.1.5 will cover the procedure of updating the prediction model with examples with a high confidence score.

4.1.1 Preprocessing

In this approach, the documents were preprocessed using a Part-Of-Speech(POS) tagger. This tagger was implemented using the Stanford CoreNLP toolkit [12], and was configured to only keep verbs, adjectives and nouns, just as in [9]. This was done to remove stopwords and unnecessary words with little to no relevance to the task of dating the documents.

The dataset was also split into three parts: P1, P2 and P3. P1 was used as an initial training set for the prediction model, where the correct publication date was known by the model. For P2 and P3, the correct publication dates were not known by the model. P2

was used as an update set, where documents with predictions of high confidence would be used to update the model. P3 was used as a test set, where a prediction was made for each document and compared to the correct publication date.

4.1.2 Document and Corpus Modelling

The original dataset contains a variety of metadata fields that were unnecessary to the experiments in this thesis. Therefore, it was natural to use a different form of data representation for the documents. The relevant data fields from the dataset was the document body and the publication date of each document. After these fields were extracted from the original dataset, it was time to create the corpus model. This was done by calculating the term frequencies of all the terms for each document and adding them up, resulting in a bag of words model with all the terms present in the corpus, and their corresponding frequencies. The same was done for each time partition, but only including the documents with a publication date within the boundaries of the time partition.

4.1.3 Computing Normalized Log-Likelihood Ratio

Now that the documents and the corpus were modeled, it was time to calculate the Normalized Log-Likelihood Ratio (NLLR) between the test documents and each time partition. This was done using the same formula as de Jong et al. (2005) [6]. This formula is described in Section 3.2, but is presented again here, with more descriptive variable names:

$$NLLR(d_i, p_j) = \sum_{w \in d_i} P(w|d_i) * log(\frac{P(w|p_j)}{P(w|C)})$$

where d_i is the model of a test document, p_j is the model of a time partition and C is the model of the entire background corpus. To prevent undefined logarithms due to zero-values for $P(w|p_j)$ or $P(w|C)$, only the intersecting terms between d_i and p_j were considered.

4.1.4 Estimating Time Partition and Confidence

After the NLLR-score is calculated for each time partition, the document's estimated partition is chosen as the one with the highest NLLR-score. At this point the confidence measure is also calculated. This is done by dividing the chosen partition's NLLR-score by the sum of the NLLR-scores of all the partitions. This results in a higher confidence score the more significant the difference is between the chosen partition's NLLR, and the NLLR-scores of the other partitions.

4.1.5 Updating the Model

During the incremental update of the model, the goal is to add new examples to the model to make future predictions more accurate. This is done after the confidence score of a prediction is calculated. If the confidence of the prediction is found to be higher than

some pre-determined confidence threshold, the model representations of the corpus and the relevant time partition are updated with the term frequencies of the document.

This procedure was performed on the P2 partition of the dataset, the update set. As new documents were added to the model, the accuracy was periodically tested using the test set to track the impact of the updates.

4.2 Regression-based Document Timestamping

This section will discuss the approach used for the second research question, where the task was to explore the use of regression-based document dating methods. The methods used for preprocessing and the document representation is covered in Section 4.2.1. The feature engineering process is covered in Section 4.2.2. Finally, the regression models used are covered in Section 4.2.3

4.2.1 Preprocessing and Document Representation

Similarly to the incremental update approach, the dataset provided more metadata fields than necessary for this approach. Therefore, the first task was to parse the dataset and extract only the body and the publication date from the dataset. After this was done, the documents were tokenized, and stopwords were removed using a list of English stopwords from the Natural Language Toolkit (NLTK) [1]. The publication dates were then converted from the format provided in the dataset[2] to a Unix timestamp. Each document is now represented as a list of all the non-stopword tokens in the original body and a timestamp corresponding to the publication date of the article.

4.2.2 Feature Engineering

The approach implements a variety of different feature types. Each feature group will be explained in this section.

Simple Lexical Features

The first feature group consists of two very simple lexical features: the length of a document (total number of tokens) and the average token length. While these features are simple, the aim is to capture some shallow temporal patterns. The assumption is that these features can represent a shift in writing style and vocabulary along the years, as an article written in 1990 might use a significantly different vocabulary of words than an article written in 2010. Zampieri et al. (2015)[19] also found that the document length in sentences was a surprisingly predictive feature, despite its simplicity.

[1] `https://www.nltk.org/`

[2] A variation of the ISO-8601 was used, specifically the format was yyyyMMddTHHmmss.

Latent Semantic Analysis

Another feature group is produced using Latent Semantic Analysis (LSA). The idea behind this feature group is that there might be relevant information about the publication date of the document in the general topic of the article. The assumption is that articles with similar topics are written in the same time period. It is easy to imagine a case where this is true, for instance topics about specific events. For example, most articles involving the dissolution of the Soviet Union are likely to be written in 1991. The assumption that articles with similar topics are more likely to be written in the same time period might not always be the case though, especially when dealing with periodically reoccurring events. For instance, articles about the Summer Olympics are likely to spike in frequency once every four years, which might result in a high similarity between these articles, even though they have vastly different publication dates.

To extract the LSA features from the documents in the dataset, I first had to build a LSA model. To make the model as accurate as possible, a Wikipedia dump[3] was used as the training data. The LSA model is made by first making a TF-IDF model from the Wikipedia articles, using Gensim[4]. The TF-IDF model is then used to create a LSA model with 200 topics. The LSA features of a document can then be inferred from the model as a vector with 200 elements.

Paragraph Vectors

Paragraph vectors aim to capture the semantic meaning of the documents. The idea behind this feature group is similar to the LSA feature group, as the assumption is that documents with high semantic similarity are more likely to have similar publication dates.

The paragraph vectors were built using the same Wikipedia dump as the LSA model. Similarly to the model used by Dai et al. (2015) [5], the model was built using the Distributed Bag of Words version of the paragraph vectors, with a vocabulary size of around 900,000 unique tokens. To test the model, I extracted the headline of the 10 Wikipedia articles most similar to the article with the headline *"Donald Trump"*. The resulting articles and their cosine similarity to the query-article is shown in table 4.1. The results seem promising, as the resulting articles seem to have a close semantic similarity to Donald Trump. The dimensionality of the model was set to 200, similarly to the LSA model.

[3]Downloaded the latest dump of all the articles from `https://dumps.wikimedia.org/enwiki/latest/`.

[4]`https://radimrehurek.com/gensim/`

Article Headline	Similarity
Marco Rubio	0.742
George W. Bush	0.739
Jared Kushner	0.729
Foreign policy of the Donald Trump administration	0.723
Presidency of Donald Trump	0.7178
First 100 days of Donald Trump's presidency	0.715
Rand Paul	0.708
Senate career of John McCain, 2001–2014	0.697
Political positions of Marco Rubio	0.697
Tom Cotton	0.696

Table 4.1: Results from querying the paragraph vectors for the 10 most similar articles to the article on Donald Trump.

Part-Of-Speech Tags

The last feature group used in this approach was Part-Of-Speech (POS) Tags. The goal of this feature group is to capture information about the grammar of the documents, making the assumption that documents with a high degree of grammatical similarity, are more likely to have similar publication dates as well. The assumption here is similar to the simple lexical features (Section 4.2.2), but the feature group itself is a bit more complex, and might capture some of the deeper grammatical differences between the time periods.

The POS features was made using a trained POS-tagger provided by the NLTK[5] library. The tagger uses a tag-set containing 35 different tags, describing different types of words (for example adjectives, verbs, nouns and personal pronouns). When generating this feature group, each token in the document is tagged, and the number of occurrences for each of the 35 tags are counted. The final representation of the document is a vector containing the number of occurrences for each of the tags.

4.2.3 Timestamp Estimation

There exists a wide variety of regression techniques that could be suitable for this task, and I have therefore chosen a few of them with varying level of complexity. The goal is to compare the results from the different regression methods, and find the most suitable to this task. The regression methods used were all implemented using scikit-learn [14], and they are described in this section.

Ordinary Least Squares

Ordinary least squares (OLS) is a simple linear regression model that minimizes the sum of squares between the true labels and the predictions. The method fits a linear regression model with coefficients $w = (w_1, ..., w_n)$, to minimize the function:

[5] `https://www.nltk.org/`

$$||Xw - y||_2^2$$

Where X and y are arrays containing training inputs and targets, respectively.

Because the model is simple, it is relatively fast to fit and it can still be relatively accurate for a lot of tasks. For example, Nguyen et al. (2011) [13] used linear regression to predict author age from text, achieving a mean absolute errors between 4.1 and 6.8 years.

One of the issues with using this model is that the coefficient estimates rely on the independence of the features, making it sensitive to random errors in the labels. This makes the model vulnerable to situations with high collinearity between multiple explanatory variables.

Ridge Regression

Another interesting regression model to test is Ridge regression. This method addresses the collinearity issue of the OLS model by imposing a penalty on the coefficient size. This makes the model more robust towards the features used. The model's coefficients minimize the following function:

$$||Xw - y||_2^2 + \alpha * ||w||_2^2$$

The penalty the model assigns can be controlled by the parameter α. A higher α value makes the model more robust to collinearity. The α value chosen for this approach was 1.

Decision Trees

The next regression model that will be included is a Decision Tree. This method infers simple decision rules based on the features of the input vectors. These rules are then used to predict the timestamp of new documents.

While this method is conceptually simple, a drawback is that it can be somewhat unstable making small variations of the input vectors might lead to a completely different result. This might lead to a higher variance in the results.

K Nearest Neighbors

Another regression method used in this approach is the k nearest neighbor (K-NN) regressor. This method uses the average of the labels of the k most similar documents to calculate the label of new documents.

The primary parameter for this method is k, determining how many training examples will be used to calculate the label of the new document. For this approach, a k-value of 5 was used.

Random Forest

The last regression method included in the tests is the Random Forest regressor, proposed by Breiman (2001) [2]. This is an ensemble method, combining the use of multiple randomized decision trees. Each tree in the ensemble is built from a random sample from

the training set. This randomness is meant to reduce the variance of the estimator by combining multiple, diverse trees.

The primary parameters to adjust for this method are the number of estimators, and the number of features to consider for each split. For this approach, the number of estimators were set to 100, and method was set to consider all the features each split.

Chapter 5

Experimental Setup

This chapter will describe the setup used for the experiments made during the project. Section 5.1 covers the experiments related to the first research question about incremental updates, and Section 5.2 describes the experiments related to regression-based document timestamping.

5.1 Incremental Update

This section will cover the experimental setup for the approach answering the first research question tied to incrementally updating a document dating model. Section 5.1.1 describes the dataset used in the experiments. Section 5.1.2 describes the different parameters that could impact the performance. Section 5.1.3 covers the metrics used to present the results, and lastly, Section 5.1.4 describes the goals of each experiment.

5.1.1 Dataset

The dataset used in this thesis is the **New York Times Annotated Corpus** [15]. This corpus contains over 1.8 million articles published by The New York Times in the time period between January 1, 1987 and June 19, 2007. While the dataset contains multiple forms of metadata, this thesis will only use the body of the articles and the publication date.

The earlier approach by Kotsakos et al. (2014) [9] also used the New York Times dataset. They do, however, only use subsets of the dataset, ranging from one to ten years. The task of incrementally updating the prediction model requires a larger dataset. This is because the regular temporal classification task typically partitions the dataset in two parts: one set used for training and one for testing. The incremental update approach, however, requires a third partition: the update set. The update set will also need to be quite large, due to the fact that only some of the examples in this set will actually be used to update the prediction model. Because of this, I chose to use the entire dataset for this approach, and not experiment with smaller subsets.

5.1.2 Parameters

There are a few parameters that can be tuned to measure the performance of the incremental update algorithm. These will be covered in this section.

Time Partition Length

The length of each time partition is an important parameter as it decides the granularity of the model. This parameter is set in number of days, and is set to 365 for this approach. This is due to the large size of the dataset, and a partition length of a year seems reasonable.

Confidence Threshold

The confidence threshold decides which documents are used to update the model, and is a very important parameter. A low threshold will increase the size of the model more rapidly, but the label of the added documents also have a higher chance of being incorrect, misleading future predictions.

Initial Training Set Size

The initial size of the training set is also and interesting parameter to consider. It seems intuitive that the incremental update will have a larger effect on a smaller initial training set, as the documents added during the update will make up for a bigger portion of the total examples.

5.1.3 Metrics

The main metric used to measure the performance of the model is accuracy. The accuracy is calculated by taking the number of correctly predicted test examples divided by the total number of test examples. The accuracy of the model is measured periodically during the incremental update procedure and represented as a plot with the total number of documents the model is built on for the x-axis.

Another interesting metric to track is the fraction of the documents added during the incremental update that have the correct label, and how many that have the wrong label and is therefore expected to have a negative impact on the performance of the model.

5.1.4 Experiments

The main experiment's goal is to measure the impact the incremental update has on the performance of a model, and to measure the impact each of the parameters has on the results.

The baseline used for this experiment is the performance of the model at different training set sizes, without the incremental update. This can be thought of as an ideal version of the incremental update methodology because its results are equal to that of a perfect incremental update where every added example is correctly labeled. This baseline therefore represents the theoretical maximum performance of the incrementally updated model.

5.2 Regression-based Document Timestamping

This section covers the setup used for the experiments tied to the second research question, about regression-based methods for document dating. Section 5.2.1 covers the dataset used for the experiments, while Section 5.2.2 covers the parameters that can be tuned to measure a difference in performance. Section 5.2.3 describes the metrics used to measure the performance, and finally, Section 5.2.4 describes each experiment's goal and setup in more detail.

5.2.1 Dataset

The dataset used for this task is the same as for the previous experiment, the **New York Times Annotated Corpus** [15]. The dataset is described in Section 5.1.1.

Although the original dataset spans over 20 years, it would require vast amounts of memory to extract features from the entire 20 years. It would also be time consuming. Because of this, it was decided to use subsets of the dataset, varying in size.

5.2.2 Parameters

The method has a few different parameters that can be adjusted to optimize performance. These parameters are described in this section.

Dataset size

As mentioned in Section 5.2.1, it was decided to experiment with different subsets of the dataset. The subsets used in this experiment will be one where the first year is used, and one where the first three years are used. This will keep the memory requirements manageable, while still allowing the measuring of the impact the dataset size has on the results.

Feature Groups

Another parameter that can be tuned is which feature groups to include in the experiment. This will be useful when trying to measure the impact of each feature group. All the feature groups used are described in Section 4.2.2.

5.2.3 Metrics

It is useful to keep track of a few different metrics to be able to thoroughly assess the performance of the models. The metrics used in this experiment are described below.

Mean Squared Error

The first metric used is the Mean Squared Error (MSE). This metric is calculated by measuring the numeric difference between all the predictions and the actual labels in the test set. These errors are then squared and averaged to provide a metric for the entire test set.

Average Error

Another metric to keep track of is the average error. This is similar to MSE, but instead of squaring the error, the absolute value is used instead. This is more intuitive to humans, as it represents exactly how much the model is off by on average. To make this feature group even more readable to humans, the average errors have been converted from seconds to days.

Accuracy Within 30 Days

The next metric used is the accuracy of the prediction model, provided some pre-determined acceptable error. The error I chose as acceptable was 30 days.

This metric is calculated by measuring the fraction of the test set where the estimated timestamp is within 30 days of the true target. This metric is useful for comparing the results to classification-based methods, as the metric is somewhat analogous to the fraction of test examples that would be labeled correctly with a certain time partition length.

Coefficient of Determination

The last metric that was calculated is the coefficient of determination (R^2). This metric is a measure of how well the independent variables of the model accounts for the variance of the results. The best possible R^2 score is 1.00, and a model can have a negative score. A model that consistently predicts the expected value of the timestamps, not regarding the difference in input vectors, would get a R^2 score of 0.

5.2.4 Experiments

The main goal of the experiments is to compare the performance of different regression models and combinations of feature groups. First the regression models will be tested using all available feature groups. Then, using the best performing regression model, the impact of the individual feature groups will be measured. I will run these tests using both the 1-year and the 3-year datasets to measure any potential difference in performance with an increase in the number of documents and the span of their publication dates.

Chapter 6

Results And Evaluation

This chapter will present and evaluate the results from the experiments described in the previous section. Section 6.1 presents the results from the first experiment exploring the incremental update approach, while Section 6.2 showcases the results from the second experiment regarding regression-based document timestamping.

6.1 Incremental Update

This section covers the experimental results and evaluation of the task provided in the first research question. First, the baseline is presented in Section 6.1.1, before the different parameters are tuned, and their impact is measured in Section 6.1.2.

6.1.1 Baseline

The baseline used for this experiment is shown in Figure 6.1. This curve is, as mentioned in Section 5.1.4, a representation of the ideal performance of the incremental update algorithm, as it models the performance of an incremental update where every added example is correctly labeled.

The curve shows a form of diminishing returns, where adding new documents has less of an impact the more documents are already used to train the model. This is expected as the accuracy of the model will converge towards a given value and not infinitely increase with an increase in the training set size.

It is also worth noting that the absolute change in accuracy is quite small. As seen in the figure, a doubling of the training set size from 400,000 to 800,000 documents only leads to an accuracy increase from 22.6% to 24.6%.

Figure 6.1: Accuracy of NLLR-model based on the number of documents used to train the model. This is based on all 20 years of the dataset and a time partition length of 365 days. Note that the beginning values for the axes are not zero.

6.1.2 Parameter Tuning

Because of the diminishing returns seen in the baseline (Figure 6.1), the size of the training set (partition P1), was chosen to be relatively small initially. The aim is to have the training set small enough to allow the incremental update to have a meaningful impact. The initial training set must not be too small, however, as this might make the initial model too inaccurate and therefore increase the number of incorrectly labeled additions during the update. The training set size parameter was initially set to 20% of the dataset (60% update set and 20% test set). The incremental update experiment was then ran with different values for the confidence threshold, ranging from 0.5 to 0.9.

Figure 6.2 shows the change in accuracy every 1,000 added document during incremental update over the full 20 year dataset, with a partition length of 1 year, a confidence threshold of 0.7 and an initial training set size of 20%. As can be seen from the x-axis of the graph, only 4,503 new documents were added during the incremental update. As the update set has a total size of around 1,098,600 documents, this means only 0.41% of the documents processed during the incremental update had a confidence score above 0.7. Additionally, from the second metric, only 18.11% of the added documents were correctly labeled, so even though the change in accuracy was insignificant due to the small change in total documents, the metrics seem to suggest that the incremental update would lead to worse performance, even if the update set was larger and more documents could be added in the same manner.

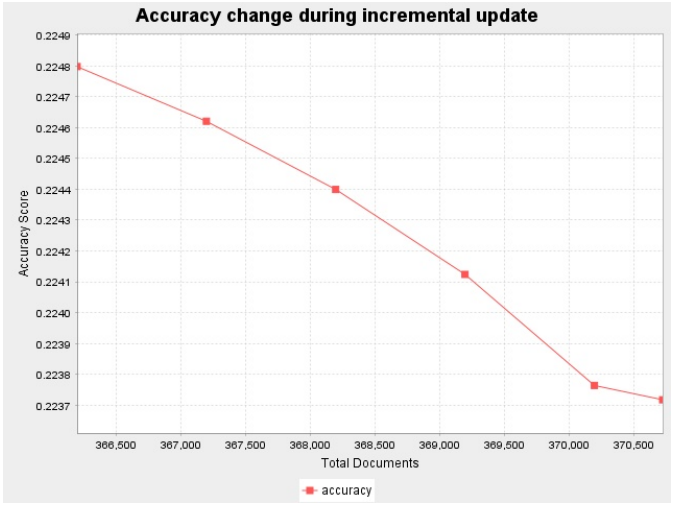

Figure 6.2: Accuracy plot for incremental update with a confidence threshold of 0.7 and a training set size of 20%. Note that the axes start at non-zero values.

When testing different confidence thresholds, there were similar results. Neither of the thresholds tested lead to a significant change in the accuracy, again due to the low amount of new documents added from the update set. There was, however, some differences in the number of added documents, and the percentage of the added documents that were correctly labeled. These differences can be seen in Table 6.1. As one might expect, decreasing the confidence threshold lead to an increase in the amount of new documents added, but also a decrease in the accuracy when estimating the labels of these documents. The table also shows the opposite effect when increasing the confidence threshold: fewer documents were added, but the accuracy of the estimated labels were slightly higher.

CT	Added Docs	Correct
0.5	5021	17.61%
0.7	4503	18.11%
0.9	4022	18.75%

Table 6.1: Total number of docs added and the percentage of added docs that are correctly labeled for different confidence thresholds. Initial training set size is 20%, CT = confidence threshold.

One issue with the model seems to be that very few documents in the update set met the required confidence score, and that the ones who did had a low chance of being correctly labeled. This might be because of the relatively small training set size, making the initial model too inaccurate. Therefore it was decided to increase the training set size to 40%. This would hopefully lead to a higher fraction of the update set being used and a higher accuracy among the added documents' labels. The issue with increasing the training set size, however, is that it leads to a smaller update set, and a lower absolute number of documents added. Therefore it is expected to see an even lower change in the actual

accuracy of the model.

The results from the test with a larger training set size is shown in Table 6.2. The change in accuracy was statistically insignificant due to the small number of added documents. The change in the percentage of correctly labeled added documents, is significant, increasing from 18.11% using a training set size of 20% (Table 6.1), to 22.39% with a training set size of 40% (Table 6.2).

CT	Added Docs	Correct
0.5	2718	21.60%
0.7	2407	22.39%
0.9	2148	23.23%

Table 6.2: Total number of docs added and the percentage of added docs that are correctly labeled for different confidence thresholds. Initial training set size is 40%, CT = confidence threshold.

This was the expected result as more documents were used when training the initial model, making it more accurate. It is still relatively low however, and the incremental update would likely not improve the model's performance if the same trend continued with a larger update set. The table also shows that 2407 documents were added when the training set size was 40% and the confidence threshold was 0.7. With this setup, the update set contains in total 732,388 documents, which means only 0.38% of the update set met the required confidence level to be added to the model. This is a slightly smaller fraction than with a training set size of 20%, which means that the model's confidence has not increased, despite more of the predictions being correct.

6.2 Regression-based Document Timestamping

This section will cover the results from the second experiment, about regression-based document timestamping. First, there will be an overview of the result from all the regression models, using all the feature groups in Section 6.2.1. Then, the impact of each feature group will be measured in Section 6.2.2.

6.2.1 Overview

Table 6.3 shows the metrics described in Section 5.2.3 for all the regression models tested, using all feature groups described in Section 4.2.2, and a dataset size of 1 year. Table 6.4 shows the same metrics when increasing the dataset size to 3 years.

Model	avg_diff	acc_30d	mse	r2_score
OLS	89.29	16.36%	8.05e+13	0.03827
Ridge	89.29	16.37%	8.054e+13	0.03777
Decision Tree	113.5	**20.12%**	1.526e+14	-0.8236
K-NN	94.86	17.76%	9.608e+13	-0.1479
Random Forest	**86.03**	18.24%	**7.582e+13**	**0.0942**

Table 6.3: Resulting metrics for all the regression methods tested on the 1-year dataset. All feature groups were used. Note that avg_diff is listed in days for readability.

Model	avg_diff	acc_30d	mse	r2_score
OLS	268.4	6.02%	7.312e+14	0.02878
Ridge	268.4	6.02%	7.312e+14	0.02878
Tree	348.8	**7.60%**	1.409e+15	-0.8714
K-NN	284.4	5.81%	8.638e+14	-0.1474
Random Forest	**262.1**	6.07%	**7.004e+14**	**0.06968**

Table 6.4: Resulting metrics for all the regression methods tested on the 3-year dataset. All feature groups were used. Note that avg_diff is listed in days for readability.

These results show that Random Forest seems to be the best performing model in terms of most metrics, with the exception being that the Decision Tree regressor has a higher percentage of the estimations having less than 30 days error. Since the Decision tree regressor also has the highest average difference and the highest mean squared error (MSE), this model seems to have a very high variance, which is consistent with the characteristics of decision trees, as mentioned in Section 4.2.3.

The results also show a decrease in accuracy, and an increase in average error when increasing the dataset size. The reason for this might be that the dataset spans over a larger time frame, making the labels range over a 3 year long period compared to only 1 year. This makes each day of error a smaller fraction of the time frame covered by the span of the labels. To account for this, it might be interesting to take a look at the average absolute error (*avg_diff* in the tables), as a fraction of the time frame of the dataset. The average error is then similar between the two datasets, ranging from 24% to 31% for the 1-year dataset, and 24% to 32% for the 3-year dataset.

Because the Random Forest regressor showed the best overall performance, it was chosen as the preferred regressor to use when measuring the impact of each feature group.

6.2.2 Impact of the Feature Groups

Now that Random Forest was determined to be the most accurate regression model, the next interesting question was which feature groups had the most impact on the model's performance, and whether or not each feature group made a positive or negative impact on the performance. To measure the impact of each feature group, they were removed from the input vectors one by one, and the performance was measured. Table 6.5 shows the performance when removing each feature group.

Feature Group	avg_diff	acc_30d	mse	r2_score
Lexical	86.08	18.13%	7.595e+13	0.09265
LSA	86.60	18.01%	7.668e+13	0.08391
POS-Tags	86.41	**17.80%**	7.616e+13	0.09014
Paragraph Vectors	**86.89**	18.01%	**7.715e+13**	**0.0783**

Table 6.5: Results when each feature group is removed. The regression model used is Random Forest and dataset size is 1 year. The largest change for each metric is marked with bold text.

The feature group that has the biggest impact on most metrics seems to be the paragraph vectors, with the exception that the POS-tags has the biggest impact on the percentage of test documents being labeled with an error of 30 days or less. This is not surprising, as the paragraph vectors is the most complex and comprehensive feature group used, and aims to capture some of the deeper patterns of the documents' topics and semantic profiles.

Chapter 7

Conclusion and Future Work

In this chapter, the thesis will first be summarized and some concluding thoughts will be expressed In Section 7.1. There will also be a few ideas for future work relevant to the thesis presented in Section 7.2.

7.1 Conclusion

The project described in this thesis, mainly has two goals: To increase the performance of existing document dating methods by automatically increasing the training set size, and to measure the performance of some regression methods when applied to the document dating task. These goals were formulated as research questions in Section 1.3, and were defined as follows:

RQ 1. How will the performance of a document dating model be impacted by incrementally increasing training set size?

RQ 2. What will be the performance of a regression based document timestamping model?

7.1.1 Incremental Update

After running the first experiment, about incrementally updating a NLLR-based document dating model, a few problems became clear. While the idea works on an intuitive level, it was very difficult to find a good balance for the parameters.

For example the size of the training set. A smaller initial training set will leave more room for the update procedure to improve the model before the diminishing returns of adding more documents to the model becomes too prevalent. The problem is that a smaller initial training set will also reduce the performance of the initial model, affecting the accuracy when predicting the labels of the documents that are added during the update. Another problem with a smaller initial training set is that it leaves less of the dataset to be used for updating the model, resulting in fewer documents being added during the update, and thus less of an impact on the overall performance of the model.

It seems that to make the incremental update strategy viable, one would need an initial training set of sufficient size to make relatively accurate predictions, at least among the predictions with a high confidence score, but also small enough to leave room for improvement and to have enough examples in the update set. I was not able to achieve this balance with this dating method and dataset.

The other parameter used, the confidence threshold, was also difficult to balance. A higher confidence threshold is generally good to increase the chance that the documents that are added to the model is labeled correctly and will have a positive impact on the overall performance. The negative aspect of a high confidence threshold is that fewer documents will meet the requirements, so even though the impact is more likely to be positive, it will also be a smaller impact than if a lower confidence threshold was used. One can imagine a scenario where lowering the confidence threshold would improve the model's performance, if it would lead to more correctly labeled examples to be added than inaccurate ones.

It is difficult to find the correctly balanced confidence threshold, as it must be tuned according to the initial training set size. A lower training set size might require a higher confidence threshold because the initial model is less accurate, increasing the likelihood of adding incorrectly labeled examples. Similarly, a larger initial training set size would likely require a lower confidence threshold, because the update set would be smaller, making it necessary to add a larger fraction of the documents to have a measurable impact on the model's performance.

Another problem that was encountered during the incremental update experiments was that the confidence measure was somewhat misleading. The confidence of a prediction is only measured within the context of that prediction, meaning that a prediction can get a high confidence if the NLLR-score between the document being evaluated and a particular time partition is high, relative to the NLLR-score of the other time partition. It does not necessarily mean that the time partition has a high similarity with the document in general, only that it is more similar than the other time partitions. This leads to scenarios where all time partitions have a low NLLR-score with a document, and the prediction still gets a high confidence score, due to one of the time partitions having a slightly higher NLLR-score, even if it is still low.

7.1.2 Regression-based Document Timestamping

The performance of the models tested during the second experiment was hard to compare to other results, as all the previous approaches related to document timestamping using the **New York Times Annotated Corpus** [15], were based on temporal classification, not regression. The best way to compare the results, seem to be using the percentage of the samples that had an error of less than 30 days. It is worth mentioning, however, that this metric is not completely analogous to the temporal classification results as even though any correctly classified example in the classification task is guaranteed to have an error lesser than the time partition length, p, all examples with an error less than p in the regression model is not guaranteed to be within the boundaries of the correct time partition. Still, the approach suggested by Kotsakos et al. (2014) [9] has an accuracy of 23.4% when using their *BurstySimDater* on the first year from the **New York Times Annotated Corpus** [15], which seems to outperform the regression-based approach presented in this thesis.

When comparing the different regression methods, the Random Forest regressor was found to have the best performance, although the linear models (OLS and Ridge) was not very far behind. Considering the linear models have a lower time complexity than Random Forest has, they might also be viable options to consider.

When examining the impact each individual feature group had on the models' overall performance, the paragraph vectors seemed to be the biggest contributor. The best performance was achieved when using all feature groups, however, so none of the feature groups examined in this approach seems to have had an negative impact on the accuracy of the models.

7.2 Future Work

At the beginning of the project, I had very limited experience in the field of content-based document timestamping. Because of this, there was a lot of trial and error and a lot of things that could have been done differently. This section will cover some of the things I would have done differently, as well as some other potential ideas for future work based on this thesis.

7.2.1 Incrementally Updating Other Timestamping Models

Even though the incremental update method implemented in this approach did not have much success in increasing the accuracy of de Jong et al. (2005)'s dating method [6], there might be other dating methods that are more suitable for the idea.

One could, for instance, try to implement a similar idea for the *BurstySimDater* proposed by Kotsakos et al. (2014) [9], where examples from the update set are added to the reference corpus and is then potentially used to predict the time partition of future test documents. One significant difference between these dating methods is that the *BurstySimDater* only uses the top-k most similar documents to the query document. This means that documents added during the incremental update would have a relatively small chance of being selected and used to date a new query document, but in the scenario where it is selected, it would have more of an impact on the result.

Another potential way to improve the performance is to use a different dataset, or combine data from different sources. The primary aim with this is to maximize the number of documents per unit of time. This could make it possible to make a relatively accurate initial model, and still have a large enough update set to be able to measure a significant change in accuracy.

7.2.2 Incremental Updates for Other Supervised Learning Tasks

Another interesting approach would be to use the idea behind incremental updates to implement a similar approach for other prediction tasks. This would be interesting since limited dataset size is a prevalent issue within all supervised learning tasks, not only content-based document dating. Finding a way to automatically expand a model with correctly labeled examples could be very helpful in multiple areas.

7.2.3 Potential Feature Groups for Regression

A way the second experiment can be further explored, is to experiment with more feature groups. Some potential feature groups could be for example involve the use of temporal expressions within the document body. There are plenty of tools available to extract these temporal expressions, for instance the Stanford Temporal Tagger (SUTime) [4], TARSQI [18] and HeidelTime [17].

These temporal expressions can be used to make different kinds of features. One could for example have a feature representing the first explicit temporal expression mentioned in the body, or calculate a time period of a pre-determined granularity (i.e. month or year) that has the highest number of occurrences in the document.

Another idea for new feature groups is to introduce features based on the metadata fields provided in the **New York Times Annotated Corpus** [15]. The approaches made in this thesis are purely content-based, but, as mentioned in Section 5.1.1, the dataset used includes multiple forms of metadata. This metadata includes tags from a controlled vocabulary of persons, organizations, places, topics and titles mentioned in the documents.

Bibliography

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] Guihong Cao, Jian-Yun Nie, Jianfeng Gao, and Stephen Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 243–250, 2008.

[4] Angel X Chang and Christopher D Manning. Sutime: A library for recognizing and normalizing time expressions. In *Lrec*, volume 2012, pages 3735–3740, 2012.

[5] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[6] Franciska De Jong, Henning Rode, and Djoerd Hiemstra. Temporal language models for the disclosure of historical text. In *Humanities, computers and cultural heritage: Proceedings of the XVIth International Conference of the Association for History and Computing (AHC 2005)*, pages 161–168. Koninklijke Nederlandse Academie van Wetenschappen, 2005.

[7] Nattiya Kanhabua, Roi Blanco, Kjetil Nørvåg, et al. Temporal information retrieval. *Foundations and Trends® in Information Retrieval*, 9(2):91–208, 2015.

[8] Nattiya Kanhabua and Kjetil 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. Springer, 2008.

[9] Dimitrios Kotsakos, Theodoros Lappas, Dimitrios Kotzias, Dimitrios Gunopulos, Nattiya Kanhabua, and Kjetil 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.

[10] Wessel Kraaij. Variations on language modeling for information retrieval. In *ACM SIGIR Forum*, volume 39, pages 61–61. ACM New York, NY, USA, 2005.

[11] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[12] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[13] Dong Nguyen, Noah A Smith, and Carolyn P Rosé. Author age prediction from text using linear regression. In *Proceedings of the 5th ACL-HLT workshop on language technology for cultural heritage, social sciences, and humanities*, pages 115–123. Association for Computational Linguistics, 2011.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Evan Sandhaus. The new york times annotated corpus, 2008.

[16] Frank Schilder and Christopher Habel. Temporal information extraction for temporal question answering. In *New Directions in Question Answering*, pages 35–44, 2003.

[17] Jannik Strötgen and Michael Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 321–324. Association for Computational Linguistics, 2010.

[18] Marc Verhagen, Inderjeet Mani, Roser Sauri, Jessica Littman, Robert Knippen, Seok Bae Jang, Anna Rumshisky, Jon Phillips, and James Pustejovsky. Automating temporal annotation with tarsqi. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 81–84, 2005.

[19] Marcos Zampieri, Alina Maria Ciobanu, Vlad Niculae, and Liviu P Dinu. Ambra: A ranking approach to temporal text classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 851–855, 2015.

[20] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

NTNU

Kunnskap for en bedre verden