NTNU
Norwegian University of
Science and Technology

Knowledge Base Acceleration Using Features Inspired by Collaborative Filtering

Benjamin Weggersen
Sam Mathias Weggersen

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer Science

Benjamin Weggersen
Sam Mathias Weggersen

Knowledge Base Acceleration Using Features Inspired by Collaborative Filtering

Master's Thesis in Computer Science, Spring 2017

Data and Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Most of us use knowledge bases every day, whether it is Wikipedia, Netflix, an online newspaper or a dictionary. These knowledge bases contain both timeless and up-to-date information and must be updated continuously. A knowledge base is often updated with content from channels with large amounts of information. When a human editor updates a knowledge base, it can be difficult to distinguish important signals from noise. To automatically identifying central documents from a stream of documents where the purpose is to expand a knowledge base is called Knowledge Base Acceleration (KBA). The task of separating the central documents is called Cumulative Citation Recommendation (CCR). Through feature extraction, document classification and ranking, we can find documents that will be central to a knowledge base. It will then be up to a human editor to select which documents are to be included in the knowledge base.

Our work in this thesis has two parts. First, we want to verify work done earlier in this field by applying similar principles to a new dataset. The second part of the thesis focuses on expanding the dataset with new features that are based on principles from Collaborative Filtering (CF). The model that solves the first part of the thesis achieves an F_1 value of 0.853. This is twice as good as the reference model which determines whether a Twitter message is central based only on the number of *likes*. Furthermore, we expand the first model with new features inspired by CF. The new model is significantly better and achieves an improvement of 1% points.

The thesis' main contribution to the KBA field is the insight that; taking into account the relationships between authors of incoming documents can improve a model's ability to identify central documents. Furthermore, we have verified that relevant principles from KBA work by applying these principles to a whole new dataset. The results of this work have been very satisfactory and are statistically tested.

Sammendrag

De fleste av oss bruker kunnskapsbaser hver dag, enten det er Wikipedia, Netflix, en nettavis eller en ordbok. Disse kunnskapsbasene inneholder både tidløs og aktuell informasjon og må oppdateres kontinuerlig. En kunnskapsbase blir gjerne oppdatert med innhold fra kanaler med store mengder med informasjon. Når en redaktør skal oppdatere en kunnskapsbase kan det være vanskelig å skille ut viktige signaler fra støy. Det å automatisk identifisere sentrale dokumenter fra en strøm med dokumenter hvor hensikten er å utvide en kunnskapsbase kalles Knowledge Base Accleration (KBA). Oppgaven å skille ut de sentrale dokumentene kalles Cumulative Citation Recommendation (CCR). Gjennom egenskapsutvinning, dokumentklassifisering og -rangering kan vi finne dokumenter som vil være sentrale for en kunnskapsbase. Det blir så opp til en redaktør å velge ut hvilke dokumenter som skal tas med videre inn i kunnskapsbasen.

Vårt arbeid i den oppgaven er delt i to. Først ønsker vi å verifisere arbeid som er gjort tidligere i dette feltet ved å anvende tilsvarende prinsipper på et nytt datasett. Den andre delen av oppgaven fokuserer på å utvide datasettet med nye egenskaper som er basert på prinsipper fra Collaborative Filtering (CF). Modellen som løser den første delen av oppgaven oppnår en F_1 verdi på 0.853. Det er dobblet så bra som referansemodellen som avgjør om en Twitter-melding er sentral kun basert på antall *likes*. Videre utvider vi den første modellen med nye egenskaper basert på CF. Den ny modellen er signifikant bedre og oppnår en forbedring på 1%-poeng.

Oppgavens hovedbidrag til KBA-feltet er innsikten om at det å ta hensyn til relasjoner mellom forfattere av innkomne dokumenter kan forbedre en modells evne til å identifisere sentrale dokumenter. Videre har vi verifisert at relevante prinsipper fra KBA virker ved å anvende disse prinsippene på et helt nytt datasett. Resultatene fra dette arbeidet har vært meget tilfredsstillende og er statistisk testet.

Preface

This master thesis is submitted to the Norwegian University of Science and Technology in Trondheim, as a final fulfilment of the requirements for a Master of Science in Computer Science with specialisation within Artificial Intelligence. This work was conducted at the Department of Computer and Information Science (IDI) during the spring semester of 2017.

Benjamin Weggersen

Sam Mathias Weggersen

Trondheim, 18th June 2017

Acknowledgments

We would like to thank our supervisor, Associate Professor Heri Ramampiaro at the Department of Computer and Information Science (IDI), for his guidance, insights, and enthusiasm in the work that we have been doing.

We are both married, so we would like to thank our wives, Ane Benedicte and Silje, for much patience and support this spring, as well as over the five years that we have studied at NTNU.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI Artificial Intelligence. 17

ANN Artificial Neural Network. 17, 37

CCR Cumulative Citation Recommendation. 3, 22, 24, 25, 27, 28, 37

CF Collaborative Filtering. 1–3, 5, 21, 24, 28, 33, 38, 43, 46, 47, 53, 56, 58, 71, 75, 78, 79, 90, 93, 95, 97–99

CV Cross Validation. 16, 68

GCLD Google Cross Lingual Dictionary. 23

KB Knowledge Base. 1, 2, 5, 22–24, 27–30, 32, 37, 43, 58, 61, 71, 72, 93

KBA Knowledge Base Acceleration. 1–4, 21–25, 27–29, 71, 76, 78, 83, 92, 93, 97, 98

KBP Knowledge Base Population. 21, 29

LTR Learning-to-Rank. 23–25

ML Machine Learning. 15, 18, 55

NLP Natural Language Processing. 9, 34, 54, 55

PR Precision-Recall. vii, 18, 76, 78, 83, 86–88

RDD Resilient Distributed Datasets. 56

RF Random Forest. 3, 16, 17, 37, 47, 68, 76–79, 82, 83, 93

TREC Text Retrival Conference. 22, 27

1 Introduction

We want to create a model capable of presenting documents that will be central to a Knowledge Base (KB). This falls under the topic of Knowledge Base Acceleration (KBA), which is the automatic process of identifying central documents from a stream of documents where the purpose is to expand a KB. We will use feature extraction, document classification and ranking, to be able to find documents that will be central to a knowledge base. Finally, it will be up to an editor to select documents that should be included in the KB.

Our work in this assignment has two parts. First, we want to verify work done earlier in this field by applying similar principles to a new dataset. Our knowledge base will be the Techmeme website which curates the most relevant technology news articles every day. For each news item they link to on the front page, they promote Twitter messages that are especially relevant to the news subject. Our task is to evaluate a large stream of Twitter messages and automatically identify the messages that are most important regarding different tech news articles.

The second part of the assignment focuses on expanding the dataset with new features that are based on principles from Collaborative Filtering (CF). This involves identifying the similarity between the various authors of Twitter messages and then awarding any writer a weighted contribution from his immediate neighbours. Our assumption is that if an author is similar to other influential authors, the author may be emphasised. And vice versa, if the author is similar to other writers with less meaningful contributions. In this way, we want to create a clearer distinction between key contributions and noise.

We have created three different models, the first model, Model A, solves the first part of the assignment and achieves a F_1 value of 0.853. This is twice as good as the reference model that determines whether a Twitter message is central based only on the number of *likes*. The second and third models, Model B and C, expand the Model A with new features inspired by CF. The new model is significantly better and achieves an improvement of 1% points.

1.1 Background and Motivation

With today's overwhelming amount of information, it is difficult to separate signal from noise. As time goes by, however, it often becomes evident which documents that were central. We want to help determine which documents are central early on. We also want to use influence in social media to help determine the centrality of a document.

If we have a Knowledge Base and a stream of documents, current state-of-the-art systems can determine relevant information from non-relevant, e.g. (Balog et al., 2013). Some relevant documents are also central. A central document is a document "you would cite [...] in the Wikipedia article for this entity, e.g. entity is a central figure in topics/events" [1]. Being able to identify central documents is very helpful when maintaining knowledge bases. Attempts have been made to identify central documents using feature extraction and classification (Balog et al., 2013). We want to recreate some of the features used in Balog et al. (2013) and verify their attempt on a new dataset and a new domain. Additionally, we want to identify central documents by extending the dataset with features based on principles from collaborative filtering. We believe social interactions captured through collaborative filtering can help explain and determine when we observe central documents.

1.2 Goals and Research Questions

We want to explore principles of the work around KBA and to identify central documents. Contributions to the KBA track work with a prepared dataset made available by the organisers of the track. We want to bring principles from KBA and apply them to a new dataset to investigate how effective the methods are. In addition, we want to expand these methods by quantifying relationships between authors of documents in the data stream. We explore how we can use similarity and weighted predictions from CF to quantify these relationships. Thus, our main research goal is to:

Goal *Identify in what ways principles from Collaborative Filtering help improve a classifiers' ability to identify central tweets from a stream of tweets?*

To do this, we will address the following research questions:

Research question 1 *How do we adapt principles from the area of Knowledge Base Acceleration to help identify central tweets from a newly created dataset?*

Research question 2 *How can we use Collaborative Filtering to generate new features?*

Research question 3 *How well do the features inspired by Collaborative Filtering perform compared to other features?*

1.3 Scope

In this section, we wish to define the scope of this thesis. We want to create a dataset that will be the foundation for training and testing our models. The KB

[1]http://trec-kba.org/
Retrieved on June 12, 2017.

will be scraped from Techmeme, while the stream of documents will be scraped from Twitter. The combination of these two data sources forms the foundation of the dataset. Further, we will use the raw data gathered from these two data sources and create features that best describe the data. We wish to recreate some of the features found in (Balog et al., 2013) and other articles from the KBA track, as well as create additional features.

We will try to limit the complexity of the task by looking at simple implementations of text analysis and classification. We are more interested in finding out about our feature engineering, to verify the work that has been done in the KBA track, and the creation of new models that use principles based on CF. We use a known classifier called Random Forest (RF). It is one of the algorithms that Balog et al. (2013) use in their paper.

1.4 Research Method

Based on Phillips and Pugh (1994) classification of research, this project may be classified as problem-solving research. We have identified a problem with the real world and used it as a starting point. We then adapt known methods to solve this particular problem.

The problem we are trying to solve is how to identify central tweets in relation to a knowledge base in a timely manner, exploiting data about the social influence of the tweet. Our knowledge base consists of articles that have appeared on the tech site Techmeme. The documents in the stream of data are tweets, limited to tweets that link to articles listed on Techmeme directly. We then extract and engineer features to be used for classification.

1.5 Contributions

Our contributions will be related to the KBA track and its task of identifying documents worth citing among documents in a stream. We create our own dataset as well good features that describe and discriminate a document in a vast pool of documents. This thesis will validate the work done with Cumulative Citation Recommendation (CCR) method in the past, such as Balog et al. (2013), as it is the essential task of KBA to identify documents as central or non-central. We also want to use principles from CF to improve performance. We want to use similarity between neighbouring users and predictions based on those similarity scores to improve a model's ability to identify central documents. To the best of our knowledge, this has not been done before. We propose an evaluation plan for how to measure the model's success in identifying central documents, as well a setup to compare different models. In addition, want to look at the importance of temporal features, since much of the data that we will be working with has temporal features attached to it.

1.6 Thesis Structure

This project consists of six main chapters, organised as follows: Chapter 2 provides background on topics such as Text Analysis, Methods in Recommender Systems, Data Analysis, and Evaluation methods. In the start of the chapter, we will go through the data sources used in the project. Chapter 3 gives an introduction to work that has been done in related fields. We do a review of KBA tracks, and look at methods and approaches similar to our tasks. Chapter 4 considers the research question and uses principles from recommender systems to create three different models to identify central documents. We also describe the scraping of data from Techmeme and Twitter, the various methods used for all models, including feature engineering. Chapter 5 gives a thorough review of the experiments we wish to perform, including statistics and assumptions made about the dataset and models. At the end of the chapter, we discuss how we will evaluate our models. Chapter 6 shows our results. We look at empirical methods and statistics and discuss our findings. Chapter 7 provides a conclusion to the research questions asked and discuss future work.

2 Background Theory

Background Theory will cover some key concepts that are relevant to recommender systems and to our task. In Section 2.1, we will look at *Data Sources*, that review the platforms that we will use to build our Knowledge Base (KB) and *Data Stream*. In Section 2.2, we will review text analysis methods. We will then review some *Methods in Recommender Systems* in Section 2.3. This part will lay a foundation for the Collaborative Filtering (CF) Methods that are used in some of our models. In Section 2.4, we will look at feature engineering and classification methods. At the end, in Section 2.5, we will look at evaluation methods that will be used to evaluate our models.

2.1 Data Sources

In this Section, we will review the differen data soures that we have used in order to create a KB and a Data Stream.

Knowledge Base

A *Knowledge Base* is the information source for a system. The information can be structural or nonstructural and often relies on a database. Wikipedia is a form of modern Knowledge Base system. In our system, we will use stories from Techmeme as our Knowledge Base.

Techmeme

Techmeme [1] is a semi-automated news curation service that lists the most important tech news at any point in time. The service was founded in 2005 by Gabe Rivera, and was in the start fully automatic. In 2008, human editors were introduced to complete the editorial process. The service stated that "[o]ur experience leads us to believe that a thoughtful combination of both algorithmic and human editing offers the best means for curating in a space as broad as technology." [2]

[1] `https://www.techmeme.com/`
Retrieved on December 2, 2016.

[2] `https://www.techmeme.com/about`
Retrieved on December 2, 2016.

Techmeme compiles a list of links to the most popular technology-related news of the day, by scraping central news websites and blogs [3].

Techmeme uses an algorithm to order stories by centrality or importance. This ordering depends on several factors that include the number of links to the story's web page and how old the story is. A story contains one or multiple news articles, where one is selected to represent the story. A story is, therefore, a cluster of news articles that talk about the same news. Most stories have tweets that are relevant to it; these often contain a link to one of the articles relevant to the story, but can also be important people connected to a story. A story can also have sub-stories that are relevant. These are shown as indented stories below its main story.

Twitter

Twitter is an online social microblogging service that allows its users to broadcast short posts called tweets. Twitter users can choose to follow other users, thus creating a personalized feed of the people that you find most interesting. Some users tweet of what is going on in their lives, links to content they find interesting, political comments, and much more. "Twitter is an experience. The more you use it, the more enjoyable and resourceful it will become." [4]. Twitter has more than 300 million users worldwide, celebrities and politicians often use it to announce important events or news. Unlike Facebook, Twitter allows aliases, fake names, and spoof accounts, thus it provides a verified account checkmark for those accounts that are legitimate [5]. Retweeting is an important aspect of Twitter. It is a common way to share something interesting you've seen on Twitter. A retweet can be thought of as quoting someone or citing a source [6].

Noise Tweets are subject to noise. We are not dealing with traditional documents, and it is not obvious that the same methods used in Information Retrieval will work for Tweets. Twitter users are allowed only to write short messages, restricted to 140-characters in length. This rule stems from the early idea that Twitter was an online SMS service. Due to this restriction, people use acronyms and emoticons very frequently. Spelling mistakes are normal for tweets. Capitalisation is often non-existent or used in unconventional ways, where entire sentences might be capitalised to express some emotion. Punctuation or the use of commas are commonly misused or non-existent. Abbreviations, slang and random single words are other characteristics of microblogging services. Often the mistakes are

[3] `http://archive.wired.com/techbiz/media/news/2007/05/techmeme`
Retrieved on December 4, 2016.

[4] `http://mashable.com/2012/06/05/twitter-for-beginners/#LN63jCGlPkq1`
Retrieved on November 25, 2016.

[5] `http://www.theaustralian.com.au/business/business-spectator/commentary/twitter-has-become-a-utility/news-story/0851a1b1c7e56de8807ea574ffa13ca2`
Retrieved on November 25, 2016.

[6] `http://mashable.com/2012/06/05/twitter-for-beginners/#LN63jCGlPkq1`
Retrieved on June 17, 2017.

intentional to be able to express thoughts in a limited context. Retweets can be another source of noise. Retweets do not bring any new information to the table other than that the original tweet will have increased statistics, showing that it got much traction. The actual retweet can be viewed as noise. Bots continue to flourish in the Twitter environment, even though Twitter has rather strict anti-spam policies[7] (Haustein et al., 2016). According to Zhang and Paxson (2011), it has been found that 16% of Twitter accounts "exhibit a high degree of automation". Bots can have many positive uses, like automated tweets based on fresh news articles from specific news sites. It can be seen as noise in other contexts.

Sparsity Sparisity is another problem when dealing with microlblogs. Tweets have had 10.7 words on average [8], much because of its character limit. Therefore, when comparing two users based on the term vectors from tweets are rather difficult.

Data availability Twitter makes some of its data accessible through their Streaming and Search API, though there are some clear limitations. We will be using the Search API in this thesis. The limits regarding this API is that one can only access tweets that are seven days old. Also, Twitter has their usage limits as well, with a *time out*-window of 15 min if the usage limits are exceeded. It used to be more open than it is today, but that has changed over the years. It is possible to pay for more data, from services like GNIP[9].

2.2 Text Analysis

There are a variety of domains that use content-based methods, and they all need to involve different steps to perform better in their respective domain. In other words, this step is highly domain specific. Domains such as movies, news, music, web pages, etc. Features are extracted respective to their domain and converted into a keyword-based vector-space representation. It is similar to methods used in information retrieval. This step is essential for a system to be effective, as it lays the groundwork for similarity measures, models, and prediction.

The following methods are applied to clean and prepare a document:

- **Stop-word removal** – Words such as a, an, and the, are removed. These does not give any added value to the document.
- **Stemming** – Variations of the same word are consolidated. This means that a stemming algorithm would reduce the words *fisher*, *fishing*, and *fished* to the root word, *fish*.

[7] `https://support.twitter.com/articles/20170469`
Retrieved on June 17, 2017.

[8] `http://blog.oxforddictionaries.com/2011/05/short-and-tweet/`
Retrieved on June 16, 2017.

[9] `https://gnip.com/sources/twitter/`
Retrieved on June 17, 2017.

- **Phrase extraction** – Detect words that occur together, so that the meaning is not lost. An example could be *"Computer science"*

Sentiment Analysis

Sentiment can be feelings, attitudes, emotions, a view, or opinions. Sentiment Analysis is the process of identifying a view or opinion that is expressed in a text. We will focus the sentiment analysis performed in this paper as only having the following classes, neutral, negative, and positive. Some sentences could have underlying negativity that is difficult to retrieve, such as "How could anyone sit through this movie?" (Pang et al., 2002). This makes sentiment analysis more difficult than traditional text classification. [inline]i forhold til tweets. agarwal2011sentiment

N-gram

N-gram model is a simple model that assigns probabilities to sequences of words. Models that assign probabilities to sequences of words are called language models, and thus N-gram is a language model. An N-gram is a sequence of N words, and when N=1 it is called a 1-gram or unigram and is a one-word sequence like "foo", or "bar". The same applies for when N=2, then it is called a 2-gram or bigram and is a two-word sequence like "foo bar", and "bar baz". The model can find terms that are likely to occur next to each other and terms that are longer than only one word.

We cannot just estimate the probabilities by counting the number of times every word occurs following er every long string. This is because text can change and some sentences might never be found. That is why we need N-gram. N-gram computes the probability of a word given its entire history, and it can approximate the history by just the last words. Instead of computing the probability

$$P(dog|The\ quick\ brown\ fox\ jumps\ over\ the\ lazy)$$

we can then use bigram and find the probability using

$$P(dog|lazy)$$

N-gram can be used to extract terms from a text document. Let's say we have the following text *"The quick brown fox jumps over the lazy dog"*. After removing stop words, the sentence is shortened to *"quick brown fox jumps lazy dog"*. When using a combination of unigram and bigram we end up with the following terms: *"quick"*, *"brown"*, *"fox"*, *"jumps"*, *"lazy"*, *"dog"*, *"quick brown"*, *"brown fox"*, *"fox jumps"*, *"jumps lazy"*, *"lazy dog"*. In this way the model is able to extract terms that are more than one word. When using this model on a large corpus of text documents, only the most relevant words and word combinations will be shown in the result. The result will also include the probability that the word combinations will occur together in the future. If we use a unigram, the probability score will be the same as the amount of times we have seen that word.

Word2vec

Word2vec was presented by Mikolov et al. (2013). Word2vec is not a single monolithic algorithm, but rather a tool that contains models used for Neural Language Models. The two models are CBOW and skip-gram. The tool takes as input a large text corpus and produces the word vectors as output. Vocabulary is then constructed from the training data, and then the model learns vector representation of words. The result is a word vector file, that can be used as features in both machine learning and natural language processing[10].

Shingles Similarity

Shingles Similarity, also known as W-Shingling, is a tool used in Natural Language Processing (NLP), which can be used to calculate the similarity between two text documents (Biegel et al., 2011). The method takes a text and divides it into *"shingles"* or tokens of W length. These tokens are similar to N-grams, where $N = W$, and stored in a set so that it only keep instances of the same token only once. After retrieving all the tokens of all W for each text, a Jaccard similarity is performed on the two sets of tokens. This value is Shingles Similarity Score.

$$r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} \tag{2.1}$$

2.3 Methods in Recommender System

Collaborative filtering is based on the idea that similar users act in similar ways. Ratings from peers (collaborative) can be an excellent predictor (filtering) of the target user's ratings. There are two types of collaborative filtering approaches, (1) memory-based and (2) model-based. We will consider the former in this section. We draw from Aggarwal's excellent book *Recommender Systems: The Textbook* (Aggarwal, 2016), when discussing this section.

Introduction

The memory-based approach computes the similarity between neighbouring users or items. It is known as *Neighbourhood-Based Collaborative Filtering.* This approach to recommender systems is among the earliest attempts and also the simplest.

[10]`https://code.google.com/archive/p/word2vec/`
Retrieved on June 17, 2017.

There are two types of Neighbourhood-Based algorithms:

1. *User-based collaborative filtering*
   Rating predictions are given by first identifying similar or neighbouring users to the target user, and making a recommendation for the target item based on the weighted average of those users.
2. *Item-based collaborative filtering*
   Instead of looking at similar users as the basis for the prediction, we look at similar or neighbouring items to the target item from the target user. We use the weighted average rating from these items to predict the rating for the target item.

We can express this problem of collaborative filtering as an incomplete $m \times n$ matrix $R = [r_{uj}]$ containing m users and n items. This matrix will almost always be a sparse matrix since most users will only have reviewed a small subset of all items available.

	n_1	n_2	n_3	n_4	n_5
m_1	1			5	
m_2		5			4
m_3	5	3		1	
m_4			3		
m_5				3	5

Table 2.1: Incomplete user-item ratings matrix

When we apply *user-based collaborative filtering*, we look at similarities row-wise, between different users. When we apply *item-based collaborative filtering*, we look at similarities column-wise, between the various items.

As stated earlier, recommender systems are trying to solve two problems: prediction and ranking. Neighbourhood-Based Collaborative Filtering solves each of these problems by:

1. *Predicting rating values*
   This is simply predicting the missing rating r_{uj} of user u for item j.
2. *Determining the top-k items*
   This is to identify the top-k items for a given user, as determined by ratings. These ratings are either given by the target user or predicted by the system.

It is interesting to note that to solve the second problem; we need to solve the first issue. This is how individual ratings are predicted.

Key Properties of Ratings Matrices

We have a rating matrix denoted by R. It has m users $\times$ n items. The specified ratings in this matrix are referred to as training data, while unspecified entries are referred to as test data. This is analogous to classification, regression, and semi-supervised learning algorithms. The recommender problem can be considered a generalisation of classification and regression.

The ratings can be of different types

1. *Continuous ratings*
   Ratings can be any real number. The lower the number, the more the dislike. The issue with this approach is that the user needs to think of a real number from a set of infinite possibilities. This adds an unnecessary cognitive load on the user.
2. *Interval-based ratings*
   Ratings can be any integer in an interval of integers. We typically see 5-point or 7-point scales. Examples of integer values in a 5-point scale could be values from 1 to 5, from -2 to 2, or from 1 to 7.
3. *Ordinal ratings*
   Similar to interval-based ratings, except that ordered categorical values may be used. An example of this could be "Strongly Disagree", "Disagree", "Neutral", "Agree", and "Strongly Agree". One has to be careful when using this type of rating not to assume that the distance between any pair of adjacent ratings is the same. If only an even number of values are provided, we have a *forced choice* situation, as there is no neutral element the middle.
4. *Binary ratings*
   Ratings can be one of two options. This type of rating can be considered a special case of both interval-based and ordinal ratings. It is also a forced-choice rating as the user has to choose between two options.
5. *Unary ratings also known as implicit feedback*
   Ratings can only be positive. The user can either provide a rating, which indicates a real preference or do nothing, which indicates a neutral preference. These ratings are often made indirectly through user actions, like viewing a post or watching a movie. A user can also rate directly using a "like" button or something similar.

Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The similarity is based on the orientation of the vector and not the magnitude. This means that two vectors with the same orientation have a cosine similarity of 1, while two vectors at 90° have a similarity of 0. Equation 4.1 show cosine similarity, where A_i and B_i are components of vector A and B respectively.

$$\text{CosineSimilarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}} \tag{2.2}$$

The resulting value is between -1 and 1. The value -1 indicates that the vectors are exactly opposite, 0 indicates decorrelation and the value 1 indicate that the vectors are identical. In IR we use tf-idf values for terms with this method, and since tf-idf values cannot be negative, the cosine similarity will range between 0 and 1.

We are going to derive the User-Based Neighbourhood Model and the Item-Based Neighbourhood Model.

User-Based Neighbourhood Models

The User-Based Neighbourhood model predicts ratings for a target user by looking at average ratings from similar users to the target user (Aggarwal, 2016).

The first step in this model is to calculate the similarity to all other users. This is not a trivial task as users might have very different rating patterns. For instance, one user might only use the upper end of the rating scale. We address this by normalising the ratings, as we will see later.

Let's consider the incomplete rating matrix $R = [r_{uj}]$ with m users and n items. u represents the user, and j represents the item. We want to compute the similarity between a target user u and neighbouring users v. Let I_u represent the index positions of items that user u has rated. The set of items that both user u and v have rated is given by $I_u \cap I_v$. This set is often empty, as most users do not rate the same items.

When we calculate the similarity between two users we often use the Pearson correlation coefficient. We only look at the intersection of ratings between the two users we compare, and normalise the ratings by removing the mean rating. First, let us calculate the mean rating μ_u for each user u:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \dots m\} \tag{2.3}$$

We then define the Pearson correlation coefficient between user u and v in this manner:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.4}$$

This is the definition of the similarity between two users.

$$\text{Sim}(u, v) = \text{Pearson}(u, v) \tag{2.5}$$

We want to compute the similarity between the target user and all other users who have rated the target item. We are often only interested in the top-k users, those users with the highest Pearson coefficient with the target user. Let $P_u(j)$ be the set of k closest users to target user u, who have specified ratings for item j. Each of these users will contribute to the predicted rating with their specified rating weighted by the Pearson coefficient. Also, to account for different rating scales, we will mean-centre each peer rating. We define the mean-centred rating of user u for item j as follows:

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \dots m\} \tag{2.6}$$

The raw, mean-centred rating above defines each user's contribution to the overall prediction. We will create a weighted average of these contributions. Then, we will add the target user's mean so it fits their rating scale. The overall neighbourhood-based prediction function is as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.7}$$

This function allows for much customisation by changing the similarity function or how items are filtered out.

Item-Based Neighbourhood Models

In this model we use the other items rated by target user u to predict the rating of a target item t. We weigh those ratings by their similarity to other items using collaborative filtering.

To determine the similarity between two ratings we use the *adjusted* cosine function. We use this instead of the Pearson correlation when looking at item similarities, as the adjusted cosine generally provides better results. The adjusted cosine will only consider ratings by users that has rated both items compared. Let U_i be the indices of the users that has rated item i, and U_j be the indices of the users that has rated item j. $U_i \cap U_j$ is then the set of indices of users that has rated both item i and item j. As with the User-Based model, we will use mean-centred ratings. Let s_{uj} be the mean-centred rating for item j for user u. The mean is calculated by subtracting the average rating of each item. The adjusted cosine similarity is defined as follows:

$$\text{AdjustedCosine}(i, j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}} \tag{2.8}$$

Similar to the User-Based model, we will determine the top-k most similar items to the target item t. These items, denoted by $Q_t(u)$, will help determine the prediction for item t. We calculate the adjusted cosine for each item $j \in Q_t(u)$ with the target item t. This similarity determines how influential the rating by target user u for item j will be in predicting the rating for t. The prediction function is as follows:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j,t)|} \tag{2.9}$$

We have predicted the rating by looking at the user's ratings on similar items.

Cold-Start problem

Guo (1997) says that a recommender system only can produce good recommendations after it has accumulated a large set of ratings. Moreover, since one of the problems in recommender systems is that the number of available ratings, to begin with, is often little, it becomes more difficult to apply traditional collaborative filtering models (Aggarwal, 2016). Other recommender system methods based on content and knowledge are more robust against cold start, but might not be readily available. Thus, when the system has none or limited data to use for predicting ratings, it is a cold start problem. Both the problems of making recommendations for new users and new-item are regarded as cold-start problems (Schein et al., 2002).

Clustering

Clustering is the grouping of similar objects (Hartigan and Hartigan, 1975). The similarity between examples are chosen by a proximity measure, and a number of examples in each group are often denoted k. There are many ways to cluster data, but we will look at the most primitive method. Lets say we have a sample set S and a proximity function called $d(e, v)$, where e and v are examples found in S. d returns the distance between e and v. We pick out a random example e. If we want to get the closet k examples, we can run d on all examples in the sample set S and pick out the k closest example. Clustering can reduce the size of examples that needs to evaluated together by a large factor. Let us say there are 1 000 000 users in a dataset. To perform extensive analysis on all users could take some time. If we first perform clustering, we can greatly reduce the dimensionality of the task. One problem can be that when users are removed, or not compared against, this can degrade recommendation quality (Linden et al., 2003).

2.4 Data Analysis

Feature engineering

A *feature* is a piece of information that is potentially useful for prediction. Feature engineering is the tasks related to designing feature sets for Machine Learning (ML) applications. Domingos (2012) writes about the importance of feature engineering in his excellent paper *A Few Useful Things to Know about Machine Learning.*

> At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used. ... Often, the raw data is not in a form that is amenable to learning, but you can construct features from it that are. **This is typically where most of the effort in a machine learning project goes.** ... So there is ultimately no replacement for the smarts you put into feature engineering. (Emphasis added)

Features can either be handcrafted, which is this topic, or gathered through automatic feature selection [11]. To be able to effectively perform feature engineering, one has to understand the properties of the task at hand, also called domain knowledge. Feature engineering is more difficult because it is domain-specific. Having domain knowledge will help to create the features that will interact with the strengths and limitations of the model. This choice can drastically affect performance.

Training and testing

Training and testing are essential in ML. A model needs a training set to train. A model will try to predict a variable based on all the others. To verify how well the model works, we need a new dataset, also called a test set. If a model does very well in training, but get poor results during testing, it is because the model learned a very specific way of predicting the training data. This is called overfitting. We want the model to be as general as possible. Also, we want to avoid bias. Data is biased when the expected value is different from the true value of the population (Mandel, 2012). To avoid bias, an algorithm can not exclude true classifications from the hypothesis space (Mitchell et al., 1997). For machine learning tasks, it is then common to divide the dataset into two random sets. One is used for training and the other for testing. The training set tends to be larger or similar to the test set. If one have different algorithms and want to find out which one works best, one can train the models on the same training set and evaluate which model that works best with the test set.

[11]`https://people.eecs.berkeley.edu/~jordan/courses/294-fall09/lectures/feature/slides.pdf`
Retrieved on June 16, 2017.

Cross Validation

K-fold Cross Validation (CV) is the process that partitions a sample set of size N into a training set of size $N-1$ examples and a test set of size 1, and is repeated N times. By using this method, one can train and test on the entire dataset. The balance between bias and variance is changed with the choice of k.

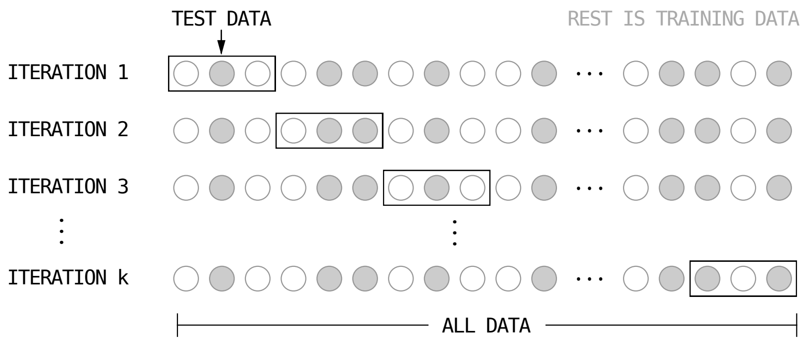


Figure 2.1: A graphical explanation of k-fold cross validation

Stratified sampling

To perform stratified sampling on a population means that the parts are sampled randomly but the class distribution from the selected column is maintained. It can also help with variance reduction.

Classifiers

Random Forest

Random Forest (RF) was proposed (Breiman, 2002), and adds an additional layer of randomness to bagging. In RF each node is split using the best split among a subset of predictors that are randomly chosen at that node. This is different from the classic tree split methods that split at the best split among all variables for each node. Even though the overall best variable is chosen for each split, it performs very well compared to other classifiers, including support vector machines and neural networks (Liaw and Wiener, 2002). It is also robust against overfitting (Breiman, 2002). RF is part of ensemble based methods. Ensembles are a divide-and-conquer approach used to improve performance. The main principle is to use a set of what is called *weak learners* and combine them into a *strong learner*. A *weak learners* is defined as a classifier that can label examples better than a random guess, which would be 1/2 in a binary classification problem. A *strong learner* is a classifier that can label examples close to that of the true classification. We can then look at the trees as *weak learners* and the RF as a *strong learner*. The runtimes for RF are very fast, and the classifier can deal with missing and unbalanced data.

Artificial Neural Network

Our brains use extremely large interconnected networks of neurons to process information and model the world we live in. This network of neurons is called a neural network. In the field of Artificial Intelligence (AI), we use machine learning algorithms to model this network and call it Artificial Neural Network (ANN). ANNs have received much traction over the last years. In the past, it was impractical or computationally impossible to perform these networks over multiple layers. In later years this has been possible, and we refer to multi-layered neural networks as Deep Neural Networks. Because this group of classification methods are so good, they have won numerous contests in pattern recognition and machine learning (Schmidhuber, 2015). We will not go into more detail on the subject, but a thorough overview of the field can be found in Schmidhuber (2015).

2.5 Evaluation Methods

Accuracy Accuracy is the number of examples classified as positive divided by the number of examples from a test set. It can be a good parameter to evaluate classifiers. One downside with this parameter is with imbalanced datasets. Imbalanced datasets mean that the class distribution is not uniform among the classes, such that there exists a majority class and a minority class. Say that we work with a binary classifier and the data is imbalanced. If most of the examples are *negative*, and the model classifies each example as *negative*, it will have a very high accuracy. If we just rely on the accuracy of the model, we can be misguided. This did not help us to understand how good the classifier performed with *positive* examples. For a better evaluation parameter on imbalanced data, we can use the following confusion table as a basis and introduce Precision and Recall.

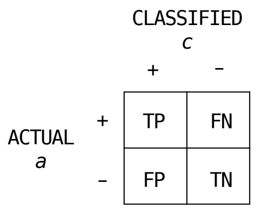


Figure 2.2: Confusion table for Precision and Recall example

Precision Precision measures the fraction of examples classified as *positive* that are *positive*, shown in the equation below.

$$precision = \frac{TP}{TP + FP} \tag{2.10}$$

Recall Recall measures the fraction of examples classified as *positive* from the set of all positives. This is shown in the equation below:

$$recall = \frac{TP}{TP + FN} \tag{2.11}$$

Precision-Recall Curve

Precision-Recall (PR) curves are often used in Information Retrieval (Manning et al., 1999; Raghavan et al., 1989). In PR space, one plots Recall on the x-axis and Precision on the y-axis.

In PR space the goal is to be in the upper-right-hand corner

F-measure

F-measure is the weighted harmonic average between precision and recall (Goutte and Gaussier, 2005). We refer to the balanced F-measure, where $\beta = 1$, as the F_1-measure (See Equation 2.12). The best score for the F_1-measure is 1, while 0 is the worst score.

$$F_1(t) = \frac{2 \cdot Precision(t) \cdot Recall(t)}{Precision(t) + Recall(t)} \tag{2.12}$$

McNemar's test

McNemar's test is a statistical test used on paired labelled data. In the perspective of ML, these labels are *actual* and *predicted.* The test is based on χ^2. If we have two classifiers C_A and C_B and enough data for a separate test set, we can determine which classier will be more accurate on new test examples. This can be done by measuring the accuracy of each classier on the separate test set and applying McNemar's test (Dietterich, 1998).

Let us say we have a dataset. We divide the dataset into a training set R and a test set T. We then train both algorithms A and B on the training set, which output the classifiers C_A and C_B. To apply McNemar's test (Everitt, 1977), we test the two classifiers C_A and C_B on the test set. Each example $e \in T$ is recorded, and its results create a contingency table, that will be the basis for McNemar's analysis.

number of examples misclassied by both C_A and C_B	number of examples misclassied by C_A but not by C_B
number of examples misclassied by C_B but not by C_A	number of examples misclassied by neither C_A nor C_B .

Table 2.2: Contingency table for McNemar example

n is the total number of examples from the test set T. According to the contingency table above, $n = n_{00} + n_{01} + n_{10} + n_{11}$ is the total number of examples, where n_{00} corresponds to the number of examples misclassied by both C_A and C_B etc. This is shown in the the following table:

n_{00}	n_{01}
n_{10}	n_{11}

Under the null hypothesis, the two algorithms should have the same error rate, which means that $n_{01} = n_{10}$ (Dietterich, 1998). The following equation show the McNemar's test.

$$\frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} - n_{10}} \tag{2.13}$$

If the null hypothesis is correct, then the probability is less than 0.05. We can reject the null hypothesis in favour of the hypothesis that the two algorithms have different performance when trained on the training set R. See Dietterich (1998) for a more complete review of McNemar's test.

In this Chapter, we have outlined key theories and methods that will be used in the rest of the thesis.

3 Related Work

3.1 An introduction to our problem

We want to verify work was done earlier in the field of Knowledge Base Acceleration (KBA) by applying similar principles to a new dataset. We needed a new dataset to create similarity scores for neighbouring users. This leads us to the next focus area of our research. That is, we want to use principles and features inspired by Collaborative Filtering (CF) to improve the model's ability to identify central documents. In addition to the model that use the content based and temporal features of the new dataset, Model A, we propose two additional models. These models have different approaches on how to use principles from CF to generate meaningful features.

In this Chapter, we want to review related fields, and get an overview of *state-of-the-art* methods that are closest to our task.

3.2 Knowledge Base Population (KBP)

Knowledge Base Population (KBP) is the effort of constructing a knowledge base, based on information from a text collection (Balog and Ramampiaro, 2013). Two research efforts in this area are Open Information Extraction (Etzioni et al., 2008) and The Never-Ending Language Learner (Carlson et al., 2010). Both systems build a knowledge based on extracted structured information from unstructured web pages. YAGO (Hoffart J and G, 2013; Suchanek et al., 2007) is another related project. It builds a knowledge base by aggregating information from multiple sources, such as Wikipedia, Wordnet[1], and Geonames[2]. The Text Analysis Conference (TAC) introduced a dedicated Knowledge Base Population track in 2009 (Ji and Grishman, 2011). Three key components of KBP are addressed: entity linking, slot-filling, and cold start KBP.

[1]`http://wordnet.princeton.edu/`
Retrieved on June 12, 2017.

[2]`http://www.geonames.org/`
Retrieved on June 12, 2017.

3.3 Knowledge Base Acceleration (KBA)

KBA attempts to address this fundamental question: "Given a rich dossier on a subject, filter a stream of documents to accelerate users filling in knowledge gaps."[3]. Let's consider KBA through an example. Let our Knowledge Base (KB) be a set of articles from Wikipedia. We call these articles entities. We then have a stream of incoming documents. They can be news articles, blogs posts, tweets, forum posts or other content. We would like to identify documents from the stream that would add knowledge to the entities in our KB. In KBA, a human editor would typically make the final call about which documents to cite in the Wikipedia articles. Thus, KBA only suggest which documents are most important. The task of identifying these documents is called Cumulative Citation Recommendation (CCR). An incoming document can be classified as Garbage, Neutral, Useful or Central/Vital. Below is a short description of each state:[3]

- **Garbage** The document provides no information about the target entity.
- **Neutral** The document mentions the entity, but provides very little information about the entity. The information could simply be a vauge reference to the entity.
- **Useful** The document could be citable, but the information contained is not timely and contains little detail or no new information about the entity.
- **Central/Vital** The document contains information that would motivate an update to the entity. The information contributes with new information or could be very timely.

The main part of the CCR task is to distinguish between Useful and Central/Vital documents. The Text Retrival Conference (TREC) track established in 2012 was dedicated to KBA and the problem of CCR[4]. Participants of the track had access to a large dataset and could use any method or approach to solve the problem. The dataset contained a pre-recorded data stream with content from three main sources: social media (blogs and forums), news (public news wire), and links (content from URLs shortened at `bitly.com`)[5]. Each document in the data stream had been annotated as either Garbage, Neutral, Useful, or Central/Vital. Most participants of the track approached the problem using classification or learning-to-rank, with an emphasis on feature creation.

In this section we will look at some concepts and methods that have been used to solve the CCR task in the KBA track. The track was active between 2012 and 2014.

[3]`http://trec-kba.org/trec-kba-2014/vital-filtering.shtml`
Retrieved on May 3, 2017.
[4]`http://trec-kba.org/kba-ccr-2012.shtml`
Retrieved on May 18, 2017.
[5]`http://trec-kba.org/kba-stream-corpus-2012.shtml`
Retrieved on May 18, 2017.

Entity-based retrieval models

A common, first step to the KBA problem is to use an entity-based retrieval method (Liu and Fang (2012) and Bellogín et al. (2013)). The purpose of the method is to filter a set of documents based on an entity[6] or query, and then rank the results. Two examples of filtering functions that has been used in KBA are:

- Comparing the titles of the articles in the KB with the content of incoming documents using a string match (Liu and Fang (2012) and Efron et al. (2012))
- Scoring incoming documents using various scoring methods and then filter by using a cut off level (Abbes et al., 2013)

Entity-based retrieval models are frequently used in the KBA track.

Many of the KBA submissions look for related entities and use these to create a richer filtering. One model (Liu and Fang, 2012) derives related entities from links to other entities found in the text. Specifically, if the KB is a set of articles from Wikipedia, and one of the articles is about *Aharon Barak*, then all the links found in that article makes up the related entities. If we later want to determine whether an incoming document is relevant, we see if the entity's title or any of the related entities are found in the content of the incoming document. Another model uses Google Cross Lingual Dictionary (GCLD) to create realted entities (Bellogín et al., 2013). GCLD is a type of thesaurus specific for Wikipedia. Each entity in Wikipedia is given a weighted connection to other Wikipedia entities (Spitkovsky and Chang, 2012). These connections make up an entity's related entities.

The role of entity-based retrieval models varies among the submissions. Some models use entity-based retrieval models to pre-filter incoming documents, and then use use other methods such as the classification or Learning-to-Rank (LTR) to determine centrality (Efron et al. (2012) and Balog et al. (2013)). Other models use entity-based retrieval models alone to determine centrality (Liu and Fang, 2012).

When the KBA track was summarized after the first year, Frank et al. (2012) found that:

> the highest scoring systems in KBA 2012 were split between rich feature engineering from the KB versus focusing on machine learning tools, such as SVMs. In the future a combination of these approaches might score even higher.

Most submissions stopped using entity-based retrieval models alone to determine centrality after the first year. Many adopted a multi-step approach (Balog et al. (2013) and Abbes et al. (2013)) where the first step filters documents based on mentions.

[6]not to be confused with entities from the KB

Feature engineering

As introduced in the previous section, Frank et al. (2012) found that the best KBA systems were based on good feature engineering and the use of machine learning methods. These two concepts are related and linked. In order for machine learning to produce good results, the data needs to have well defined features. Good features enables machine learning methods to more clearly seperate classes of data. Feature engineering is therefore very interesting for KBA and CCR.

Balog et al. (2013) are among the first on the KBA track to emphasize feature engineering. They create four types of features: document features, entity features, document-entity features and temporal features. Document features look at the characteristics of the document such as length (term count) and source (news, social, or linking). Entitiy features look solely at the number of related entities. Document-entity features look at statistics between documents and entities, such as the number of instances of an entity in the document, the position of the first and last instance of the entity in the document, proliferation and equality. Finally, temporal features look at how often articles related to an entity had been viewed.

An analysis of those features based on information gain show that entity-based features work very well (Balog et al., 2013), including those features that measure stream volume and Wikipedia page views. This observation is interesting since those features are not taking into account the actual content of the incoming documents, but rather look at data about the data. This is something we want to investigate further in our work. We want to create new features based on principles from CF. These features will focus on the relationship between users, rather than focusing on the data directly. Further, entity-document features performed very well, especially features that captured the spread between the first and last mention of entities in incoming documents and features that found similarities between the document and entity article from the KB.

Several papers report that temporal features do not contribute much towards identifying central or vital documents. Sherman et al. (2014) says that "[o]ur investigation of [...] temporal features [...] demonstrated little or no effect on vital filtering performance". Balog et al. (2013) reports that "[t]here is no temporal feature [...] that would stand out as universally beneficial". And, lastly, Abbes et al. (2013) observed that the inclusion of temporal features, in their case "degrades the system performance".

We also observe several papers that use results from an entity-based retrieval method as features, such as in document ranking (Abbes et al., 2013).

Classification

There are primarily two approaches to solving a CCR task, namely classification and LTR (Balog and Ramampiaro, 2013). Here we will look at how to go about solving CCR using classification.

The CCR task can be seen as a binary classification problem which seeks to distinguish between relevant (central/vital and useful) and irrelevant (neutral and garbage) documents, and even more specifically between central/vital and useful documents (Wang et al. (2013) and Balog et al. (2013)). It is also possible to put these two together and create a multi-step classifier. Such a model would first classify relevant from irrelevant documents, and then, as the second step, seperate central/vital from useful documents.

Many of the submitted approaches to the CCR task uses classic classification methods such as Random Forest (Wang et al. (2013) and Balog et al. (2013)), J48 (Balog et al., 2013) and linear SVM (Kjersten and McNamee, 2012). Most of these implementations have had little or no experimentation with hyper parameters. Further, we haven't observe any optimization of estimators using k-fold cross validation. The lack of these kinds of optimization will in most instances lead to suboptimal results since tuning parameters often improves a model's ability to fit the data (Duan et al., 2003). We want to perform basic optimization on our model to better illustrate our models ability to classify the data. Finally, we cannot see that anyone has attempted to classify using deep learning.

Learning-to-Rank

Instead of classifying documents using a classifier, we can look at the CCR task as a LTR problem. The system will then try to rank all incoming documents in accordance with a learned ranker. In the KBA track the different classes a natural ranking: central/vital > relevant > neutral > garbage (Wang et al., 2013). These classes are mapped to scores ranging from 0 to 1000. The advantage of the LTR method is that it fits intuitively with CCR task. Incoming documents have varying degrees of relevance and their ranking will indicate this. An editor will first look at the top ranked documents.

LTR can use several different scoring methods, including machine learning methods, in order to rank. In the KBA track we observe the use of Markov Random Fields (Dietz and Dalton, 2013), Jaccard similarity (Li et al., 2012), Random Forests (Wang et al. (2013) and Balog and Ramampiaro (2013)), Rank Boost and LambdaMART (Balog and Ramampiaro, 2013), to name a few.

After evaluating KBA runs for 2012 Balog and Ramampiaro (2013) concluded that, on a general basis, LTR methods performed better than the Classification approaches.

Conclusion

In the current landscape of related fields, we are not able to find something that can answer our research question or solve our task. Therefore, we must solve this ourselves. In the next Chapter, we will outline our approach starting with the theoretical solution.

4 Methods and Implementation

In this Chapter, we will review our approach. We will start by looking at the theoretical solution in Section 4.1. As part of the theoretical solution we will review the three Models that we wish to present, Model A 4.2, Model B 4.3, and 4.4. In Section 4.5 we will give the description of our implementation, the system architecture, algorithms and libraries used to complete the implementation.

4.1 Theoretical Solution

We want to help the editor of a Knowledge Base (KB) find the most relevant and central tweets related to a set of news articles. A relevant tweet might mention a novel aspect about a news article or provide a decent summary. A central or vital tweet, however, should provide new insights or add value to the news article in such a way that the editor would want to cite the tweet. The task of separating central documents from relevant documents is, in the context of Knowledge Base Acceleration (KBA), known as Cumulative Citation Recommendation (CCR). The focus of the task is to help human editors maintain a KB despite high volumes of incoming documents. Centrality is defined as requiring "that the document relates directly to the target entity such that one would cite it in the Wikipedia article of that entity" (Balog et al., 2013).

Separating central tweets from relevant tweets is a hard task. That's why the Text Retrival Conference (TREC) explored this challenge through the KBA track. If we were to classify tweets manually, we can imagine that we would want to know things such as: which news article the tweet is about, the content of the tweet, who tweeted it, if and how that person relates to the news article and whether the tweet adds value or new information to the article. A person with domain knowledge about tech news would look for these and many other signals as they try to determine if a tweet is central to a particular news article. An important part of the CCR task is to know which signals or features are important to help distinguish relevant and central tweets. In this chapter, we will look closer at which features we believe will help us classify tweets, and how to classify them.

A Twitter stream often contains large numbers of incoming documents. It is computationally difficult to compare each incoming tweet with each news article in the KB to determine relevance and centrality. We need to constrain the problem, so it becomes computationally feasible. We attempt to accomplish this in part by filtering tweets by entity mention and limiting the time scope in which we look for tweets for particular stories.

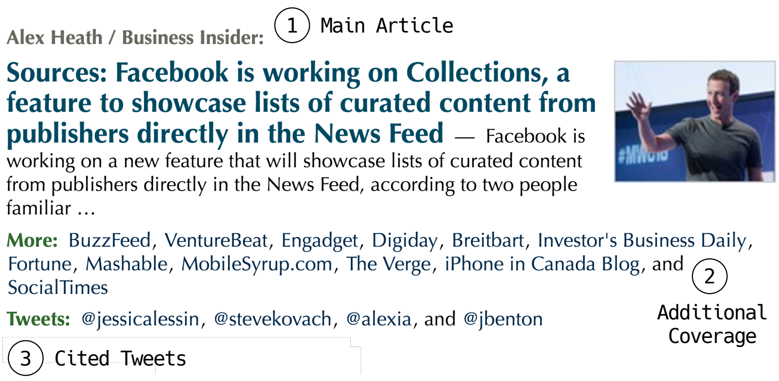
We will look at a theoretical solution for solving our flavour of the problem of CCR with incoming tweets as the data stream and a collection of news articles as the KB. Our solution will treat the CCR problem as a binary classification task, attempting to separate central tweets from relevant tweets. We will work on feature creation. We will also investigate what effect user's relationships to other users has on the importance of their tweets by creating features inspired by principles from Collaborative Filtering (CF).

4.2 Model A

Introduction

Knowledge Base

Our KB consists of stories from Techmeme, a tech news aggregator. A story on Techmeme is a collection of news articles from different publishers about the same topic. Each story has a main news article covering the story (Note 1 the figure) with links to additional coverage by other publishers (Note 2 in the figure). If the story has created interesting chatter on Twitter, the most important of these tweets will be presented alongside the story on Techmeme (Note 3 in the figure). A story can have related stories. We store the articles about the story in our KB. We make a note of which article is featured as the main article, and also the order of the items listed under additional coverage. The order says something about the priority of these articles. The cited tweets represent the central or vital documents in the problem of KBA.


Alex Heath / Business Insider:
Sources: Facebook is working on Collections, a feature to showcase lists of curated content from publishers directly in the News Feed — Facebook is working on a new feature that will showcase lists of curated content from publishers directly in the News Feed, according to two people familiar …

More: BuzzFeed, VentureBeat, Engadget, Digiday, Breitbart, Investor's Business Daily, Fortune, Mashable, MobileSyrup.com, The Verge, iPhone in Canada Blog, and SocialTimes

Tweets: @jessicalessin, @stevekovach, @alexia, and @jbenton

Figure 4.1: Example screenshot of a Techmeme Story

A story is only visible on Techmeme for about a day, on average. The KB includes information about when a story was published on Techmeme. This time frame helps us know when to look for tweets related to that particular story. We will also scrape the text of each of the news articles and perform term extraction.

Techmeme uses a combination of automation and humans to discover and present the top ten most important tech stories at any given time[1]. When the service first launched in 2005, they used computers to automatically aggregate and select stories. In 2008, however, they switched to a model that use machines and humans to select stories. Gabe Rivera, the founder and CEO of Techmeme, stated at the time

> Automation does indeed bring a lot to the table – humans cannot possibly discover and organise news as fast as computers can. However, too often the lack of real intelligence leads to really unintelligent results.[2]

Mr Rivera's remarks validate the need for KBA. A maintainer of a large KB often has to deal with an overwhelming amount of new data from the document stream. KBA attempts to automatically filter incoming data and sort it according to centrality. Our task is not to update the KB with new articles. We want to help automate the aggregation and selection of candidates for tweets worth citing alongside a story. We consider the KB to be incomplete and use data from the incoming stream of tweets to complete the incomplete KB with suggestions for citable tweets. This task is known as Knowledge Base Population (KBP) and focuses on principles such as slot filling and entity linking.

To our knowledge, a dataset that contains news articles and incoming tweets is not publicly available. A large part of this master thesis will therefore be devoted to the creation of such a dataset. When we both create the dataset and the model representing the dataset, we risk introducing selection bias. Selection bias occurred when the selection of data was not done randomly (Cortes et al., 2008). If true, the data might not represent the population it was intended to represent. We have tried to avoid selection bias in our dataset and expect it to be, at most, low. For instance, we finished the process of creating the dataset before we started developing the model. We extracted all the data that exists in the relevant time range. Also, most of the data were extracted without changes to content. However, since we knew the intent of our system from the beginning, the way we have gathered data might still be biased.

Below is an overview of the data we gather for the KB.

Attribute	Description
cited	A list of all tweets cited alongside a story.
news articles	A list of all news articles associated with this story.
story time	The time when the story was published on Techmeme.

Table 4.1: Data extracted of each story of Techmeme

As explained earlier, each story contains links to additional coverage by other

[1]`http://techmeme.com/about`
Retrieved on December 2, 2016.

[2]`http://news.techmeme.com/081203/automated`
Retrieved on May 15, 2017.

publishers. Below is an overview of the data we gather for each of those stories.

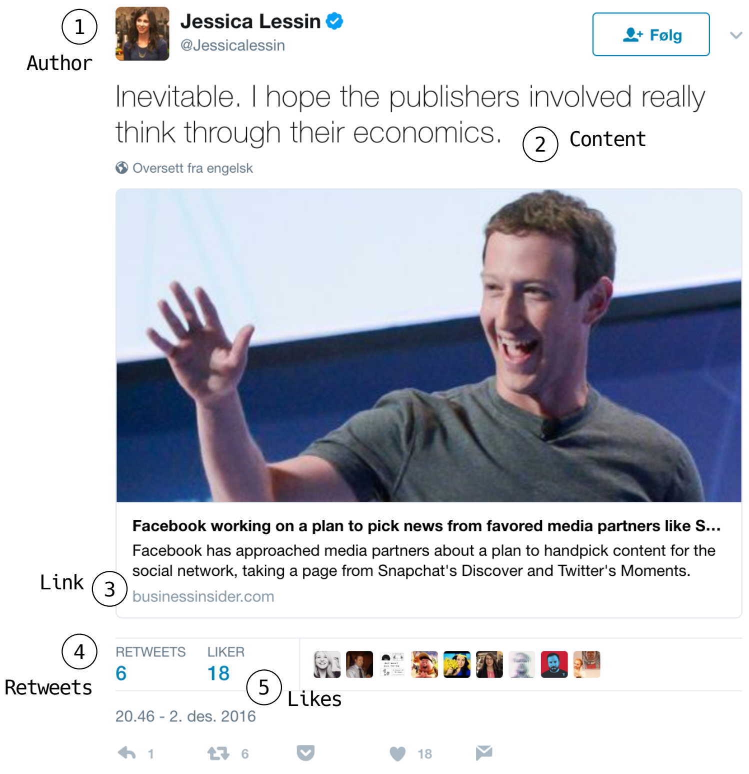
Attribute	Description
url	The URL of the news article.
content	The content of the news article.
author	The author of the news article as stated by Techmeme.
position	The position of the news article on Techmeme relative to the other articles from the same story.

Table 4.2: Data extracted of each news article in each story

Data stream

Our model takes as input a stream of documents. The stream consists of tweets that directly link to one of the news articles in a story in the KB. Each story contains links to news article coverage by various publishers about the same story. By *tweets that link to*, we mean tweets that have an actual URL in the content of the tweet. We monitor Twitter for new tweets linking to any news article found in our KB. We keep monitoring each story for 24 hours after it was published. Most tweets cited alongside a story on Techmeme are chosen within a few hours. Also, it is unlikely that a story will remain on Techmeme for more than 24 hours. At that time it is unnecessary for us to look for related tweets. Thus, 24 hours gives us enough time to gather the needed data.

We use Twitter's developer API to collect data and generate our dataset. Below we will look at a graphical representation of a tweet and some of the information we extract from that tweet. We find the tweet's author (Note 1 in the figure), the content of the tweet (Note 2 in the figure), the number of retweets (Note 4 in the figure) and likes (Note 5 in the figure). The tweets we consider will also contain a link to a news article (Note 3 in the figure).



Jessica Lessin
@Jessicalessin

Inevitable. I hope the publishers involved really think through their economics.

Oversett fra engelsk

Facebook working on a plan to pick news from favored media partners like S...
Facebook has approached media partners about a plan to handpick content for the social network, taking a page from Snapchat's Discover and Twitter's Moments.
businessinsider.com

RETWEETS 6 LIKER 18

20.46 - 2. des. 2016

1 6 18

Figure 4.2: Example screenshot of a tweet

We have listed all the data we extract about each tweet in a table below.

Attribute	Description
id	The id of the tweet.
language	The language of the tweet. Most tweets are English (en).
retweets	The number of retweets. That is, the number of times a user has forwarded the tweet to their timeline.
likes	The number of likes the tweet has received.
followers	The total number of followers the user has.
following	The total number of users the target user is following.
favourites	The total number of likes a user has received.
number of tweets	The total number of tweets the user has created.
verified	Whether the user is verified.
content	The content of the tweet.
user biography	The content of the user's biography.
username	The user's username.
name	The user's name.
tweet created date	The time the tweet was published on Twitter.
user created date	The time the user was created on Twitter.

Table 4.3: Data extracted for each tweet

As new tweets arrive, we evaluate the content and generate relevant features. These features will then make up the input for our centrality classifier.

Feature creation

Creating and selecting the right features is very important in machine learning (Levi and Weiss, 2004). Good features help expose signals in the data which allows the classification method to identify the correct bounds. It is not obvious in advance which features will be most important. To levitate this uncertainty, we will base our feature engineering on domain knowledge and previous work. One particularly good overview related to finding or creating features relevant to document streams and KB is found in Balog et al. (2013). We have also studied the tweets cited on Techmeme to look for patterns and concepts that seem unique for those tweets. We share these findings in the overview below. We have also chosen to not move forward with some features, these will be discussed at the bottom of this review.

Features based on tweets

Tweet Language We only consider tweets that are written in English. Since we will perform text analysis and term extraction, we need to be able to assume that all terms are written in the same language. Language will not be an input parameter to the actual classification algorithm. We will use it to filter the results.

Number of Tweet likes A *like* is earned when another user likes your tweet. A user can only like one tweet once. We count how many likes each incoming tweet has received. Similar to retweets, this number says something about how interesting the tweet is perceived. A tweet can also receive artificially high numbers of likes and retweets by bots or users with questionable intents. We expect the model to calibrate the importance of these values based on a sufficiently large dataset.

Number of Tweet retweets A *rewteet* occurs when one user forwards a tweet of another user to his or her timeline. We look at the number of retweets each tweet has received. This number says something about how interesting other users find the tweet. We believe that users who get cited on Techmeme often write interesting tweets, and should have higher retweet counts.

Tweet Sentiment We know through data analysis that the sentiment of cited tweets on Techmeme is either more positive or more negative than for tweets that are not cited. In other words, non-cited tweets tend to be more neutral. We believe sentiment will be an important feature for the centrality classifier. A sentiment analysis will output three separate values for each tweet indicating Positive, Negative and Neutral sentiment. Each value is between 0.0 and 1.0.

Features based on Users

These features are given when requesting tweet data from Twitter.

Number of followers Users can follow each other on Twitter. When a user follows another user, the user will see that other user's tweets on their timeline. This is the primary way for a user to discover new content. We count how many followers a user has. Sometimes the word *friends* is used instead of followers. If a user has many followers, there is a higher chance their tweets will be seen. A user that has many followers has more influence. We believe that influential users are more likely to write tweets that can be cited on Techmeme.

Number of Users following This indicator is the opposite of the *Number of followers*. It says how many users a particular user is following. We do not expect this indicator to be very significant. It is easy for a user to follow many users. Bots are particularly bad as they have a tendency to follow many users. It is not obvious what this indicator will tell us. A deeper analysis of *who* a user is following would be more interesting. We will touch on user similarity when we create features inspired by CF, later on.

Number of Tweets liked This is the number of tweets liked by a particular user. A high number shows that the user is actively reading other tweets and marking them as liked, or at least has done so in the past. A low number could either show a new user or a user who does not usually read or *like* other tweets.

Number of Tweets written The number of tweets written by a user. We are unsure how this feature will affect the model.

Is the User verified Twitter has a mechanism for verifying a user. Not every user can be verified. Twitter will "approve account types maintained by users in music, acting, fashion, government, politics, religion, journalism, media, sports, business, and other key interest areas" when the "account is of public interest". [3] This can be a good indicator that the individual is worth listening to and that it is a more likely candidate for a central tweet.

Features based on relationship to Knowledge Base

Cited This is the truth and is only used for testing and to calculate scores of the models.

[3] `https://support.twitter.com/articles/20174631`
Retrieved on June 15, 2017.

The number of Techmeme stories As stated previously, we will only look at tweets that link to articles found in our knowledge base. Many users will have linked to multiple theme stories, so we count how many stories each user has linked to on Techmeme. It is not obvious for us what we will learn about this number, but we expect that some users that act as commentators on tech news or journalists will receive high scores. We believe that at least some of the users that get their tweets promoted on Techmeme will fall into this category.

The number of Techmeme citations The number of times that a user's tweet has been promoted on Techmeme. We believe this will be a good indicator that they could be cited again. As we look at stories found on Techmeme, we see that some people are cited more than once within a short period, even a day.

The number of articles in a story Here we count the number of articles that a story has. This can be a good indicator of how important the story is. If it is an important article, most publishers will write about it. It is not obvious to us if there will be a correlation between how important a story is and how many tweets that are promoted on Techmeme.

Features based on Natural Language Processing (NLP)

It will be interesting to analyse the feature importance of content-based features. Balog et al. (2013) shows in their result that the document content of the data stream was not important. "It is somewhat surprising that the strongest features all work as a prior and do not consider document content at all."

Tweet word count This is the count of words in a tweet. Since the length of tweets can vary, this feature will capture if a tweet is short or long. It is not apparent to us if this number is important.

Term count This is the unique count of extracted terms in a tweet that also are story terms. This number will be less than or equal to *Tweet word count* since term extraction will leave out stop words. We use *N-gram* to extract and score terms.

Term percentage This number show the percentage of terms in a tweet. The number is calculated with *Term count* divided by *Tweet word count.*

Term score Each story has some articles, and all their document content is collected in a document corpus. A term score is determined for each term in the corpus. The score for each term in a tweet is given by the score from the matching terms in the story. We are unsure in what degree this feature will influence the results, as the text content in tweets is not rich.

Hot term count We introduce a new consept of hot terms. These are varied by a k, and are the top k terms for a given story. This feature has three different variations for $k = 2$, $k = 5$, and $k = 10$. The count is done by counting how many terms in a tweet is also in this set of k hot terms.

Bio Term count We do the same term extraction for a user's bio and sum the intersection of terms between user bio terms and story terms. We believe that if the user is especially invested in a topic, he or she might have described that in their bio. This could help the classifier determine if a user and tweet are worth citing.

Is the username mentioned Is the username mentioned in any of the text related to a story? We collect text from all news articles in a story as documents and combine this into a corpus of text for a given story. We search for the username in the whole corpus using *Levenshtein Distance*, where distance d is set to max 1.

Is the user an author Is the user an author of one of the news articles in a story. We believe this feature can be influential in the classification task because we have seen that such tweets have been promoted on Techmeme multiple times.

Features based on news article's relationship with Story

News article is the featured article in a Story Is the news article the featured article in a story. Each tweet is connected to a story and a news article. If the news article that the tweet is connected to is also the featured article, this feature value is $True$, and $False$ otherwise.

News article's position in a Story This is the articles' position in the list of all articles for a Story. It is similar to *News article is the featured article in a Story*, but instead of binary classifying news articles as featured or non-featured, we use their position in the story list. The new article's position is determined by Techmeme editors, and we assume that this is important for which tweets should be promoted.

Temporal based features

Tweet Created at At what date and time were the tweet created. This value should not be important to the classifier since the dataset should contain a lot of tweets, where cited tweets are some what evenly distributed.

Story Created at At what date and time were the story created. Like *Tweet Created at*, this should not affect the classifier much.

User Created at At what date and time were the user created. This temporal feature is different from the two features, *Tweet Created at* and *Story Created at*. We believe that this feature can show the classifier that some users are well established. We see that this has some importance to the tweets that are promoted on Techmeme.

Time between Story and Tweet The time between story and tweet. This value could be negative if the tweet was written before the creation date for the story. A story is always composed of news articles, and will in most cases be created after some tweets. The feature value can also be 0 or positive. 0 if the tweet is written at the same time, and positive for tweets that are written after a story. We believe that an important factor to promoted tweets is the time they were created about stories.

Number of Tweets for Story last h hours The number of tweets for the associating story, last h hours from the time of the tweet was looking at. This feature has a dynamic h that creates three distinct features, one where $h = 1$, $h = 2$, and $h = 3$. These features are constructed to capture the velocity of other tweets in the time around and after this tweet.

Number of Tweets for Story last 24 hours We look at a tweet and count a number of tweets found after between the time the tweet was written and 24 hours later. All tweet need to write about the same story. This feature is a bit *slow*, since it needs to wait for 24 hours before it can give the correct count.

Number of Tweets for news article last 24 hours This feature is similar to *Number of Tweets for Story last 24 hours*, but it now only counts tweets that also link to the same news article.

Tweet position for Story The position of a tweet for the associating story. This value can be found by sorting all tweets for a story by its date, and then take that position. Tweet position can be very important if there is an upper limit to which tweets are selected to be promoted to Techmeme.

Normalised Tweet position for Story The position of a tweet normalised by a number of tweets for a story the last 24 hours, given in *Number of Tweets for Story last 24 hours*. So by dividing *Tweet position for Story* by *Number of Tweets for Story last 24 hours*, we get a number between $0 - 1$ if the tweet was written within a day. This value can also be more than 1 if the tweet was written after 24 hours.

Tweet position for news article The position of a tweet for the associating news article. This value is calculated similarly to *Tweet position for Story.* The difference is that looks at tweets that link to the same article, within the same story.

Normlised Tweet position for news article The position of a tweet normalised by a number of tweets for a news article the last 24 hours, given in *Number of Tweets for news article last 24 hours.* The value is found by dividing *Tweet position for news article* by *Number of Tweets for news article last 24 hours.*

Classification

Now that we have defined all the features we need, we are ready to describe the training of the model. Our dataset describes the ground truth. Using this truth, we want to use supervised learning to train a classification model. The purpose of creating our classification model is to help a human editor of a KB evaluate new documents by providing signals about which tweets may be particularly relevant to a given story. Our model should help to distinguish documents that should be cited from those that are not worth citing.

We want to use the Random Forest (RF) algorithm for classification. We could have used some more advanced algorithms, from the exemplary Artificial Neural Network (ANN) family. However, the reason we want to use RF is that we do not have a direct interest in changing and tweaking parameters to get the best possible values. Our goal is instead to construct comparable models that can predict cited and non-cited documents from a data stream using well-crafted features. Our reason is similar to that given by Balog et al. (2013) in their paper. Since we have constructed our own dataset, the result of a model in itself is not of great importance. Also, we do not have another classifier to compare our results against.

RF gives us a confidence score for each element in the test set. Confidence score is $P(\mathit{cited} = 1.0)$ and returns a value ranging from 0.0 to 1.0. The value is calculated based on the number of trees predicting the current class divided by the total number of trees in RF. For the model to predict a given class, it uses a cutoff value between 0.0 and 1.0. The cutoff value is often set to 0.5 by default. We want to choose the cutoff value that gives us the highest F_1 score. This is also the metric used in vital filtering (CCR)[4]. We calculate F_1 at each confidence threshold and take the maximum F_1 as the single score for the system.

[4]`http://trec-kba.org/trec-kba-2014/vital-filtering.shtml`
Retrieved on May 14, 2017.

4.3 Model B

Introducing Features based on principles from Collaborative Filtering

We want to investigate if quantifying the relationship between Twitter users influences Model A's ability to find central tweets. We are especially interested to know how the user of an incoming tweet relates to the users of other tweets written about the same story. The idea is that similar users act in similar ways. Moreover, by linking these similarities to signals from neighbouring users, we can transfer the beneficial or disadvantageous data found in those other user's data to the target user. Our hypothesis is that a user's associations will either reinforce or weaken the chance that their tweet will be cited.

In this section and the next, we will outline two different approaches that generate new features. We add these features to Model A and attempt to produce better classifications. These models will first determine which users are similar to the user of an incoming tweet. They will then consider features from users who wrote about the story in question. The models will produce weighted features based on the similarities. The goal is to let Model A use these new features to better predict centrality. These principles come from CF and are based on User-based Collaborative Filtering.

Introducing Model B

We want to extend Model A by adding feature inspired by CF. We denote the work of extending Model A as Model B. We will use the output of Model A as the input of Model B, create a new feature, and add it back again to Model A to perform new classifications. We hope this new feature will improve the results of the model.

This auxiliary model will take as input confidence scores from the output of Model A and use these to create new predicted confidence scores. These predicted confidence scores will be created using neighbouring users' confidence scores, weighted by their similarities to the target user. We do this for each story or cluster of stories. At this stage, we do not consider individual tweets, but rather a user's contribution to a story as a whole. If a user has written multiple tweets about a story, we will only extract the features from the tweet that performed best.

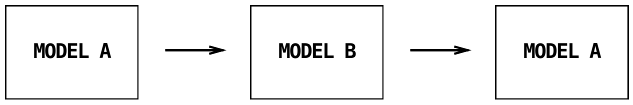


Figure 4.3: The reloationship between Model A and Model B

Below we will introduce sets, variables and parameters used in this model.

Setting the stage

Variables and parameters

$\mathbf{D}$	The entire dataset
$\mathbf{dt}$	Row vector of $\mathbf{D}$, represents one tweet
$\mathbf{df}$	Column vector of $\mathbf{D}$, represents one feature
$quart$	Last row of the first quarter in the dataset
mid	Middle row in the dataset
g	The trained classifier
$\mathbf{cf}$	A vector with confidence scores from Model A
U	Set of all users, u
J	Set of all stories, j
M	Total number of users in U, $\|U\|$
N	Total number of stories in J, $\|J\|$
$\mathbf{r}$	*User to story* matrix with confidence scores from Model A
r_{uj}	Confidence score for a user u and a story j
$\hat{r}$	The target story
r_j	*User to story* matrix that only contain users who have written at least one tweet about target story j.
$\tilde{U}$	Set of users who have written at least one tweet about any of the stories in the story cluster.
$\tilde{u}$	A user who have written at least one tweet about any of the stories in the story cluster.
$\tilde{J}$	Set of top-k similar stories in the cluster.
$\tilde{j}$	A top-k similar story
$\tilde{M}$	Total number of users in $\tilde{U}$, $\|\tilde{U}\|$
$\tilde{M}$	Total number of stories in $\tilde{J}$, $\|\tilde{J}\|$
$\tilde{r}_j$	r_j including users from $\tilde{U}$ and stories in $\tilde{J}$.
I_u	The set of indices for stories which user u has written.
$I_u \cap I_v$	Common stories between users u and v.
$\mathbf{sim}$	*user to user* cosine similarity matrix between all users given a story j
$\hat{r}$	Prediction matrix with confidence scores based on similarity to neighbouring users.
rc_{uj}	Concatenated score for story j and for all users in $\tilde{U}$.
$\hat{R}$	A matrix with column vectors being rc_{uj} for every story j in J.

Functions

$\text{CosineSim}(\tilde{u}, v)$	The cosine similarity between user $\tilde{u}$ and v
$\text{ShinglesSim}(j, k)$	The shingles similarity between two stories j and k
$\text{Q}_{\tilde{u}}(\tilde{u})$	The set of users v who also wrote about the same story $\tilde{j}$ as user $\tilde{u}$

Creating confidence scores for each tweet

Using Model A, we let $\mathbf{D} = [\mathbf{d}_{tf}]$ be the entire dataset. Each row vector, $\mathbf{d}_t$, represents one tweet, and each column vector, $\mathbf{d}_f$, represents one feature. We use Model A to train the classifier on the first quarter of the dataset using all available features. We denote the last row in the first quarter of dataset as *quart* and the trained classifier as g. Using the newly trained classifier in Model A, we generate predictions for each tweet in the last three-quarters of the dataset. So far, we have performed step 1 and 2 in Figure 4.4.

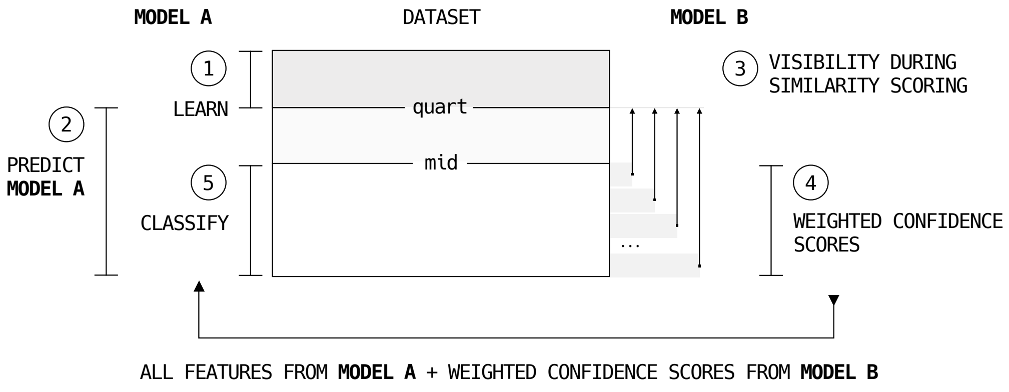

MODEL A
DATASET
MODEL B
1
LEARN
quart
3 VISIBILITY DURING SIMILARITY SCORING
2
PREDICT MODEL A
mid
5
CLASSIFY
4
WEIGHTED CONFIDENCE SCORES
...
ALL FEATURES FROM MODEL A + WEIGHTED CONFIDENCE SCORES FROM MODEL B

Figure 4.4: Overview of Model B

In addition to the class predictions, the classifier from Model A also provides us with confidence scores for each prediction. Since we are using a Random Forest learner, the confidence score is the number of trees predicted to the positive class, that is, whether a tweet should be cited, divided by the total number of trees. We denote these confidence scores by column vector $\mathbf{cf}$.

We only train on the first quarter of the data because we cannot use predictions based on training data. They are most likely biased. Data is biased when the expected value is different from the true value of the population (Mandel, 2012). Predictions based on training data are often overfitted and do not necessarily represent the population. Why won't we train on a larger portion of the data? Because an important part of Model B is to identify the similarity between users, and it needs access to history about those users. We use the second quarter of the data $\mathbf{D}$ exclusively as the basis for user similarity calculations. We will not start producing weighted confidence predictions until after the midpoint, *mid*. The use of the various parts of the data is illustrated in Figure 4.5.

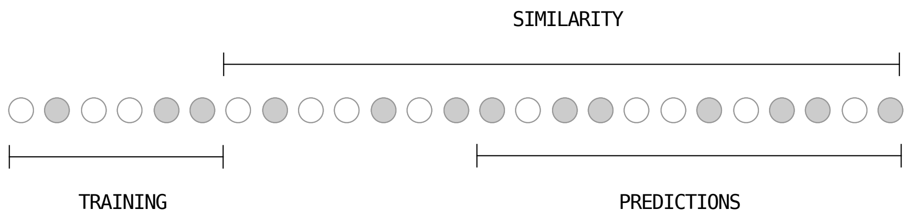


Figure 4.5: How data is used in Model B

As we move through the second half of the data predicting confidence scores, we increase the window for similarity calculations accordingly. Having access to history is especially important for the first stories we try to predict. Without this, the model will encounter what is known as the cold-start problem, where it cannot draw any inference about those first stories because it does not have enough information about their users. The history we use for similarity calculations cannot have been used as part of the training. Thus, we have to strike a balance between providing the model with history for similarity calculations and having a large enough part of the data available to create good confidence scores.

Similarity Scores

Having received the confidence scores, **cf**, from Model A, we are now ready to calculate similarity scores between each user. We will use these similarities to calculate weighted confidence scores between users who have written about the same story. Creating similarity scores is the first step of Model B, and they are created by calculating the cosine similarity between users using a *user to story* matrix. The *user to story* matrix $\mathbf{r} = [r_{uj}]$ has a dimension of $M \times N$ with M users and N stories. u is a user from the set of all users U. j is a story from the set of all stories J. M is the number of users from the set of all users U. N is the number of stories from the set of all stories J. Each element, r_{uj}, contains a confidence score from Model A. This matrix makes up the history from which we calculate similarities.

$$\mathbf{r} = \begin{array}{c|cccc|} & j_1 & j_2 & j_3 & j_4 \\ \hline u_1 & X & & & X \\ u_2 & & X & X & \\ u_3 & & X & & X \\ u_4 & X & & X & \\ \hline \end{array}$$

Figure 4.6: Example of a *user to story* matrix, **r**

To account for the temporal aspects of tweets and news articles, we create a new similarity matrix for each story. We do this because we want to use as much relevant data as possible to determine the similarity between users, but at the same time, we need to prevent the model from looking into the future. We have visualised the model's increased vision over time in step 3 in Figure 4.4. If the model required insight into future stories to calculate satisfying confidence score predictions, the entire model would be limited to offline learning or working in large batches, and would always provide delayed predictions.

We only proceed with stories that have not had any tweets used in the training of the classifier g. This is important because we do not want the model to be biased. Let's say we are considering story x. If the classifier g in Model A has already seen the data from story x, the predictor will likely be unnaturally good at predicting correct values, with high confidence, for tweets from that story. If we want to use these confidence scores weighted by the similarity to neighbouring users to predict new confidence scores, those predictions might be biased. Let's consider an example. Let's assume that similar users always act similarly. If two users are identical, and we know the outcome of one user, the model, because of the first assumption, would also know the outcome of the other user. Further, let's assume that all predictions calculated by Model B are based on biased confidence scores. Later, we use the output of Model B and append it to the original input of Model A and train the model on the extended dataset. It will now be hard for Model A to generalise. The model will learn to prioritise the predicted confidence scores because they are indirectly based on the ground truth, and will correlate with the truth. The problem will manifest itself when we give the model new data. The classifier g in Model A, and the algorithm in Model B will now create predicted confidence scores based on new data they have not seen before. The output of Model B will therefore not have the same quality as the output the model produced when the predictions were based on trained data. When Model A now takes Model B's output as its input, it will give weight to the predicted confidence feature, as it has previously learnt. However, the feature will contain more noise now and the model will not fit the new data well.

For each story j we create an $\mathbf{r}_j$ matrix. j is here the target story. We will refer to this story later as $\hat{j}$. In each of these $\mathbf{r}_j$-matrices, we only include users who have written at least one tweet about target story j. In Figure 4.7, j_3 is the target user.

$$\mathbf{r}_j = \begin{array}{c|ccc|} & j_1 & j_2 & j_3 \\ \hline u_2 & & X & X \\ u_4 & X & & X \\ \hline \end{array}$$

Figure 4.7: Example of a reduced *user to story* matrix, $\mathbf{r_j}$

Since most tweets are not central to the KB, and Model A can identify this , most confidence scores are zero. The confidence score is a reflection of how sure the model is that it should predict a tweet as "cited". Since the confidence is zero, it means the model is sure it should not classify the tweet as "cited". Since confidence scores determine similarity and similarity are used to calculate weighted confidence scores, many confidence score predictions will be zero. Such values are not helpful as the intention of creating features based on principles from CF is to quantify and express relationships between users. To address this issue, we introduce clustering of stories.

For story j we calculate story similarities between it and every story written previous to it. We ignore stories with similarity scores below a threshold p. We identify top-k similar stories. We use *ShinglerSim*$(\hat{j}, j)$ to calculate the similarity between target story $\hat{j}$ and story j. In Figure 4.8 we see an example of calculating story similarities between target story j_3 and every other story written before j_3. This is the intermediate step between $\mathbf{r}$ and $\mathbf{r}_{\hat{j}}$. Let the set $\tilde{U}$ contain all users who have written at least one tweet about any of the stories in the story cluster. Let $\tilde{J}$ be the set of top-k similar stories in the cluster.

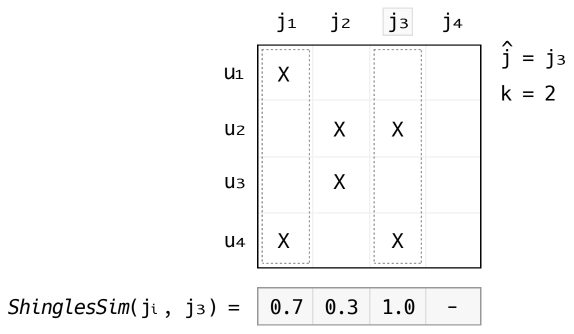


Figure 4.8: Example of finding a cluster in *user to story* matrix

We extend $\mathbf{r}_j$ to include users $\tilde{U}$, and denote this $\tilde{r}_j$. $\mathbf{r}_j$ will now have a dimension of $\tilde{M} \times \tilde{N}$ with $\tilde{M}$ users and $\tilde{N}$ stories. $\tilde{M}$ is the number of users in $\tilde{U}$. $\tilde{N}$ is the number of stories in $\tilde{J}$.

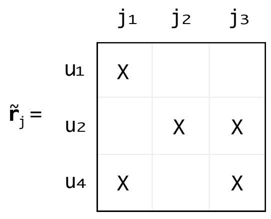


Figure 4.9: Example of an extended *user to story* matrix, $\mathbf{r_j}$

Once we have created the extended $\mathbf{r}_j$ matrix, $\tilde{r}_j$, we can create the similarity matrix. We use the cosine function on each combination of users $\tilde{u}$ and v. Let $I_{\tilde{u}}$ represent the set of indices for which user $\tilde{u}$ has written. Moreover, similarly for user v. We express common stories by the intersection between the indices of stories written by both users. Thus, $I_{\tilde{u}} \cap I_v$ denotes common stories between users $\tilde{u}$ and v. The normalisation factor in the denominator is based on all items from each user, and not the common items (Aggarwal, 2016). This means that two users not only need to receive the same confidence scores for the same stories, but they also need to write about the same story, to get high similarity scores. This helps solve a problem with users receiving high similarity scores when they only have a few stories in common.

$$\text{CosineSim}(\tilde{u}, v) = \frac{\sum_{s \in I_{\tilde{u}} \cap I_v} r_{\tilde{u}s} \cdot r_{vs}}{\sqrt{\sum_{s \in I_{\tilde{u}}} r_{\tilde{u}s}^2} \cdot \sqrt{\sum_{s \in I_v} r_{vs}^2}} \tag{4.1}$$

The *user to user* similarity matrix $\mathbf{sim} = [\text{CosineSim}(\tilde{u}, v)]$ has a dimension of $\tilde{M} \times \tilde{M}$ where $\tilde{M} = |\tilde{U}|$. Each element contains the cosine similarity between users $\tilde{u}$ and v. Both users come from the set $\tilde{U}$.

$$\mathbf{sim} = \begin{array}{c|ccc|} & u_1 & u_2 & u_4 \\ \hline u_1 & 0 & 0 & ? \\ u_2 & 0 & 0 & ? \\ u_4 & ? & ? & 0 \\ \hline \end{array}$$

Figure 4.10: Example of a *user to user* similarity matrix, **sim**

We have not used the Pearson similarity function since the confidence scores we use as input to the function come from the same distribution. The Pearson similarity function is similar to the cosine function, except it subtracts the user's mean value. This is useful when each user rates differently. However, in our case, this was not necessary as the same predictor in Model A gave confidence values to each tweet.

Predictions

We now want to predict confidence score based on similarity to neighbouring users. We denote the prediction matrix as $\hat{r}$. It has a dimension of $\tilde{M} \times \tilde{N}$ with $\tilde{M}$ users and $\tilde{N}$ stories. $\tilde{M}$ is the number of users in $\tilde{U}$. $\tilde{N}$ is the number of stories in $\tilde{J}$.

We iterate over each story $\tilde{j}$ in $\tilde{J}$ where $\tilde{J}$ is the set of all the top-k similar stories in the story cluster. Let $Q_{\tilde{j}}(\tilde{u})$ be the set of users v who also wrote about the same story $\tilde{j}$ as user $\tilde{u}$. Then, for each user $\tilde{u}$ we sum the confidence scores of all other users v who wrote about the same story, weighted by their similarity to user $\tilde{u}$. We also normalise the results. In Figure 4.11 we see an example of calculating $\hat{r}_{1,1}$ by taking the dot product between sim_1 and $r_{j_1}^T$. These are references to earlier examples.

$$\mathtt{sim[1]} \cdot \mathtt{r}_j^T\mathtt{[1]} = \begin{bmatrix} 0 & 0 & ? \end{bmatrix} \cdot \begin{bmatrix} X \\ \\ X \end{bmatrix} = Y_{11}$$

Figure 4.11: Example of calculating *user to story* matrix, $\hat{r}$

Thus, we create predicted confidence scores for each user weighted by their similarity to all other users who wrote about the same story $\tilde{j}$.

$$\hat{r}_{\tilde{u}\tilde{j}} = \frac{\sum_{v \in Q_{\tilde{j}}(\tilde{u})} CosineSimilarity(u, v) \cdot r_{v\tilde{j}}}{\sum_{v \in Q_{\tilde{j}}(\tilde{u})} CosineSimilarity(u, v)} \tag{4.2}$$

$$\hat{\mathbf{r}} = \begin{array}{c|cc|} & j_1 & j_3 \\ \hline u_1 & Y_{11} & Y_{13} \\ u_2 & Y_{21} & Y_{23} \\ u_4 & Y_{41} & Y_{43} \\ \hline \end{array}$$

Figure 4.12: Example of a *user to story* matrix, $\hat{r}$

We now have predicted confidence scores for each user $\tilde{u}$, for each similar story $\tilde{j}$. Since we are currently predicting confidence scores for target story $\hat{j}$, we need to make an average confidence score for each user. This average will be weighted according to how similar each story is to target story $\hat{j}$. We use $ShinglesSim(\hat{j}, j)$ to calculate the similarity between target story $\hat{j}$ and story j.

$$\mathbf{rc}_{u\hat{j}} = \frac{\sum_{j \in \tilde{J}} ShinglesSim(\hat{j}, j) \cdot \hat{r}_{uj}}{\sum_{j \in \tilde{J}} ShinglesSim(\hat{j}, j)} \tag{4.3}$$

The end result is a column vector for story $\hat{j}$ with an average confidence score for each user $\tilde{u}$, which we denote as $\mathbf{rc}_{\hat{j}}$. See step 4 in Figure 4.4. Figure 4.13 gives an example of how we would go about calculating the weighted confidence similarity for user u_1 with respect to target story j_3. Y_{uj} represents the confidence scores of each user u and each story j. In the example, $k = 2$, so we only consider the two most similar stories, including the target story.

$$\mathit{ShinglesSim}(j_i, j_3) = \begin{array}{ccc} j_1 & j_2 & j_3 \\ \hline 0.7 & 0.3 & 1.0 \end{array}$$

$$\frac{Y_{11} \cdot \mathit{ShinglesSim}(j_1, j_3) + Y_{13} \cdot \mathit{ShinglesSim}(j_3, j_3)}{\mathit{ShinglesSim}(j_1, j_3) + \mathit{ShinglesSim}(j_3, j_3)} = Z_{13}$$

Figure 4.13: Example of calculating cluster average

In Figure 4.14 we see the transformation from $\mathbf{r}$ to $\mathbf{rc}$.

$$\hat{\mathbf{r}} = \begin{array}{c|cc|} & j_1 & j_3 \\ u_1 & Y_{11} & Y_{13} \\ u_2 & Y_{21} & Y_{23} \\ u_4 & Y_{41} & Y_{43} \end{array} \Rightarrow \begin{array}{c|c|} & j_3 \\ u_1 & Z_{13} \\ u_2 & Z_{23} \\ u_4 & Z_{43} \end{array} = \mathbf{rc}_{uj}$$

Figure 4.14: Example of a cluster average vector, rc_{uj}

Once we have iterated over every story in U, we concatenate all column vectors rc_{uj} into the matrix $\hat{R}$. This final matrix is the output of Model B and will be the input to Model A. In Model A we will append it to the original data and then train the classifier on the extended dataset. See step 5 in Figure 4.4.

We believe Model B is reasonable and well thought out. It will be interesting to perform experiments based on this approach later in the project, and we have high hopes to get good results based on Model B.

4.4 Model C

Instead of adding a combined feature, as we did in Model B, we add some features inspired by CF. Each feature is weighted according to the user's similarity to other users. Once we have created new features, we expand the original dataset in Model A with these new features. We then run a new classification process on the expanded dataset. The work on expanding Model A is called Model C.

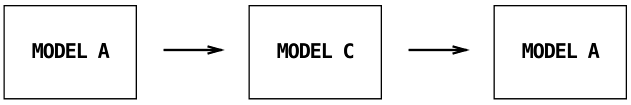


Figure 4.15: The reloationship between Model A and Model C

We start by calculating equality between users based on stories they both have written about and the similarities between these stories. We use these similarity values to create a weighted average of close users' features. The Features we retrieve come from the original dataset. By expanding Model A with more features, rather than just generating a combined feature as we do in Model B, we want to the RF classifier to choose which new features are worth the most. See the process in Figure 4.16.

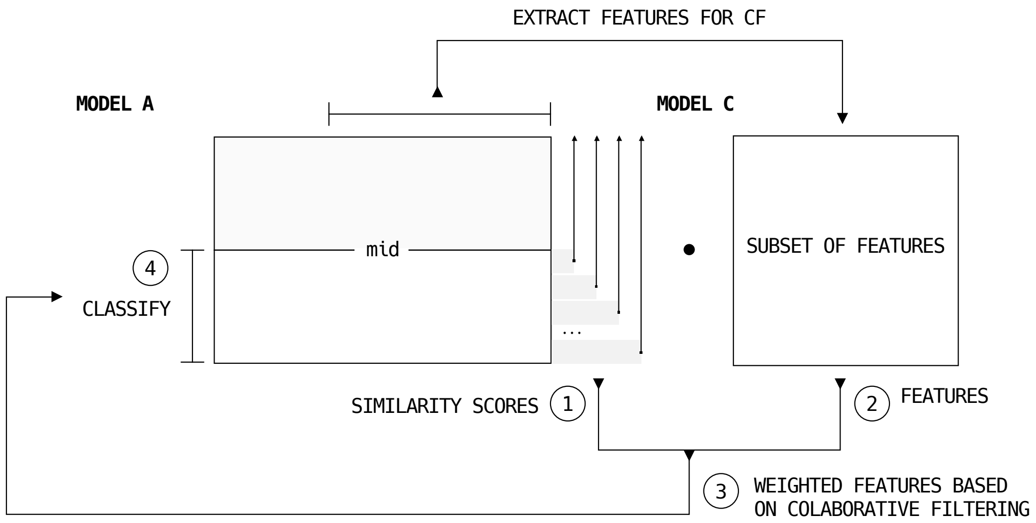


Figure 4.16: Overview of Model C

Since there are much more tweets than the central tweets, we think the models need to be precise to find central tweets. The principles behind CF are based on average values, such that the similarity calculation between users or the weighted average behind a prediction for a user, for a given story. The disadvantage of using average in this context is that averages often looks at the whole and may not capture the details necessary to be able to find specific tweets. This is what we want to investigate with Model C. In Model B we generated confidence scores, where a confidence score by itself is a kind of summary of how important a tweet is. We believe the approach behind Model C will perform better than Model B since we give the model the ability to self-examine if there are correlations with the close users' features.

Below we will introduce sets, variables and parameters used in this model.

Setting the stage

Variables and parameters

U	Set of all users, u
J	Set of all stories, j
$\tilde{U}$	Set of users who have written about story j
M	Total number of users in U, $\|U\|$
N	Total number of stories in J, $\|J\|$
I_u	Index of user u
$I_u \cap I_v$	Common stories between users u and v
$\mathbf{A}$	*user to story* indicator matrix of M ×N
$\mathbf{B}$	*story to story* similarity matrix of N ×N
b_{jk}	*ShinglesSim* between story j and k, where $k.date < j.date$
$\mathbf{C}$	*user to user* similarity matrix of M ×M
c_{uv}	Average story similarity on k closest stories to target story j that both user u and v have written about
$\mathbf{c}_u$	Row vector with the similarities between user u and all other users for story j
$P_j(u,v)$	Set of k closest stories to target story j that both user u and v have written about
$\mathbf{D}$	Similarity adjusted features for each user u for story j
$Q_j(u)$	Set of l closest users to target user u for story j
$\mathbf{d}_u$	Row vector with original features
$\hat{\mathbf{d}}_u j$	Row vector with weighted averages of the original features, based on the similarities between user u and all other users for story j
$\hat{\mathbf{D}}$	Weighted averages for all users and all features

Functions

ShinglesSim(j,k)	The shingles similarity between two stories j and k
Sim(u,v)	Average story similarity across all common stories between two users
P$_j(u,v)$	The set of k closest common stories to j for two users
Q$_j(u)$	The set of l closest users to target user u for story j

Similarity Scores

To create a weighted average of features from the nearest neighbours, we need to calculate the similarity between users. In Model C, we base similarity between users as the sum of stories both users have written about and the similarities between these stories.

The first step is to make an overview of what stories a user has tweeted about. We create a *user to story* matrix that indicates 1 if a user u has written about story j. The *user to story* matrix $\mathbf{A} = [\mathbf{a}_{uj}]$ has a dimension of $M \times N$ with M users and N stories. M is the number of unique users in the set of all users U, $M = |U|$. N is the number of unique stories from the set of all stories J, $N = |J|$.

$$\mathbf{A} = \begin{array}{c|cccc|} & j_1 & j_2 & j_3 & j_4 \\ \hline u_1 & 1 & 1 & 1 & 1 \\ u_2 & 1 & & 1 & \\ u_3 & & 1 & & 1 \\ u_4 & 1 & & 1 & 1 \\ \hline \end{array}$$

Figure 4.17: Example of a *user to story* indicator matrix, **A**

A contains column vectors, $\mathbf{a}_j$.

$$\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_N] \tag{4.4}$$

A column vector $\mathbf{a}_j$ represents one story j, and each row represents each user u in U and indicates whether user u has tweeted about story j.

$$\mathbf{a}_{uj} = \begin{cases} 1, & \text{if user } u \text{ has tweeted about story } j \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Story Similarity Scores

Now that we have *user to story* matrix, **A**, we need to define the similarity between stories. We make a *story to story* matrix and definie this as $\mathbf{B} = [b_{jk}]$. It says how similar a story j is to another story k. The matrix has a dimension of $N \times N$ where N is the number of stories in the set of all stories J, that is $N = |J|$.

$$\mathbf{B} = \begin{array}{c|cccc} & j_1 & j_2 & j_3 & j_4 \\ \hline j_1 & & & & \\ j_2 & 0.3 & & & \\ j_3 & 0.4 & 0.2 & & \\ j_4 & 0.1 & 1.0 & 0.8 & \end{array}$$

Figure 4.18: Example of a *story to story* similarity matrix, **B**

The model takes into consideration the temporal aspects of tweets and stories. It does not calculate any similarities past the story it is currently evaluating. All future similarities, which is similarities between story j and any future stories, are not defined. As we move through the data predicting new features, we increase the window for similarity calculations accordingly. We do this because we want to use as much relevant data as possible to determine the similarity between users, but at the same time, we need to prevent the model from looking into the future.

$$b_{jk} = \text{ShinglerSim}(j, k) \quad \text{if story } j \text{ is more recent than } k \tag{4.6}$$

The similarity scoring method used to compare stories is called Shingler Similarity Score (see Section 2.1). We extract the full text of each article for each story. Then we assemble all text so that each story now appears as a document. Using Shingler Similarity, we then calculate the similarities between the different stories. Shingler Similarity compares stories based on principles such as N-grams and Jaccard similarity.

User Similarity Scores

With the *story to user* indicator matrix **A** and *story to story* similarity matrix **B** we are ready to create *user to user* similarity matrix **C**. This matrix says how similar two users are.

$$\mathbf{C_j} = \begin{array}{c|cccc|} & u_1 & u_2 & u_3 & u_4 \\ \hline u_1 & X & & & X \\ u_2 & & X & X & \\ u_3 & & X & & X \\ u_4 & X & & X & X \\ \hline \end{array}$$

Figure 4.19: Example of a *user to user* similarity matrix, c_j

We iterate over each story j in the set of all stories J and create a *user to user* similarity matrix for each target story j. We denote a story-specific similarity matrix as $\mathbf{C}_j = [c_{uv}]$ with a dimension $M \times M$ where M is the number of users in $\tilde{U}_j$. That is, for each story j we only calculate similarity scores for users u and v from the set $\tilde{U}_j$. Let $\tilde{U}_j$ be the set of users who have written about story j. From the *user to story* indicator matrix $\mathbf{A}$ we remember that $\mathbf{a}_j$ is a logical column vector for story j. Each vector element of $\mathbf{a}_j$ represents a user in U. Thus, a_{uj} indicates if user u wrote a tweet about story j. Putting these concepts together, $\tilde{U}_j$ is the set of elements $\{\mathbf{a}_j = 1\}$.

To illustrate how we create $\tilde{U}_j$, the set of users from story j, we use data from matrix $\mathbf{A}$ from Figure 4.17. If we were to create $\tilde{U}_{j3}$, it would contain these three users:

$$\mathbf{\tilde{U}_{j3}} = \begin{array}{|ccc|} \hline u_1 & u_2 & u_4 \\ \hline \end{array}$$

Figure 4.20: Example of a set of users who have written about story j_3

For each combination of user u and user v (where $u \neq v$) we sum the story similarities between the target story j and any other story that both users have written, s. Each similarity between stories j and s is given by b_{js}. After summing the story similarity from common stories between user u and user v, we divide by the number of common stories. Thus, we create an average story similarity across all common stories between two users.

Let $P_j(u, v)$ be the set of k closest stories to target story j that both user u and v have written about. We express common stories by the intersection between the indices of stories written by both users. Let I_u represent the set of indices for which user u has written. Thus, $I_u \cap I_v$ denotes common stories between users u and v.

$$c_{uv} = \text{Sim}(u, v) = \frac{\sum_{s \in P_j(u,v)} b_{js}}{|P_j(u, v)|} \tag{4.7}$$

We have now created a similarity scoring function that uses the similarity between common stories between two users to determine user similarity. It is based on the idea that similar users write about similar things. However, many users tweet about different topics. Two users might have some of those topics in common, while other topics are unique for each user. We want to have the option to only include the similarity from some of the most similar stories, so that two users can be determined as similar for the stories they do have in common, even though they also have other interests. That is why we included the parameter k.

With a *user to user* similarity matrix, we are ready to start predicting new features. This concludes step one in Figure 4.16.

Predictions

We create similarity adjusted features for each user u for each story j. Let $\mathbf{D} = [\mathbf{d}_f]$ be all the features which we want to use to create similarity adjusted features. The column vector $\mathbf{d}_f$ contains the original feature values for each user in the set U for a particular feature f. Extracting these features is illustrated as step two in Figure 4.16.

What we want to do next is to calculate a dot product between the *story to story* similarity matrix $\mathbf{C}_j$ and the feature matrix $\mathbf{D}_j$ from those same users. This is denoted as step three in Figure 4.16. We will describe this process by calculating only one element from the dot product result. That is, we will calculate the predicted feature vector for user u for story j. Let $Q_j(u)$ be the set of l closest users to target user u for story j. We use the row vector $\mathbf{c}_u$ in $\mathbf{C}_j$, which lists the similarities between user u and all other users for story j, to determine which users are closest. For each of the users found in $Q_j(u)$ we calculate new features by weighting the original features found in the row vector $\mathbf{d}_v$ using the similarity between users u and v. We then take the average of these new features weighted by their respective similarity scores.

$$\hat{\mathbf{d}}_{uj} = \frac{\sum_{v \in Q_j(u)} \text{Sim}(u, v) \cdot \mathbf{d}_v}{\sum_{v \in Q_j(u)} |\text{Sim}(u, v)|} \tag{4.8}$$

Sometimes a user has tweeted multiple tweets about the same story. If this is the case we will calculate the average of each feature for that story and for that user.

We calculate $\hat{\mathbf{d}}_{uj}$ for each user u in U and each story j in J.

Now that we have a generated new features based on principles from CF we can append these to the data used in Model A and train on the new data. This last step is illustrated as step four in Figure 4.16.

We hope that by providing the learner with multiple features that all share the same similarity weights the learner will be able to give priority to the features that matter. These new features should give users who are similar to users that get cited a chance to be promoted. If too many users who write irrelevant tweets also write about the same stories as the top users, then the weighted features will not be pure enough to differentiate between central tweets and relevant/irrelevant tweets.

4.5 Description of Implementation

In this section, we review the process required to create the dataset that will be used for Models A, B and C. The feature creation step, which produces the dataset, is divided into two pipelines. The first pipeline is part of Model A, to create features to be used for training and classification. The first part of the first pipeline is written in Python. We have used Python because we know the language and found libraries that helped us achieve our implementation goals, including a simple connection to MongoDB and a good API wrapper for Twitter. We use the Java programming language to write the last part of the first pipeline, as we use Datumbox to perform sentiment analysis and term extraction from text. We had python script that we manually ran twice a week for over a month, to collect all raw input data necessary for the pipeline.

The second pipeline use features from Model A, to create features inspired by CF for Model B and Model C. Model B and Model C create different CF features, and the implementation of these features will be reviewed. At the end of this section, we will review the implementation of the classification methods.

System Architecture

In this section, the core platforms and services used in the system's pipeline will be discussed. At the end, the overall architecture of the system will be presented.

MongoDB

MongoDB is a document-oriented database that saves documents that have a similar structure to JSON with a key and value for all fields. All documents are saved in collections. It is not necessary to instruct MongoDB what the incoming data will be, as long as it has a key and value for each data row, thus it is possible to push virtually any data to a collection. The data placed in the same collection does not need to follow the same structure as other data currently in the collection. This type of database is called schemaless. Because of these inequalities to traditional SQL systems, and others, MongoDB is a member of another type of database system, called NoSQL. NoSQL systems work well with data that is unrelated, indefinite, or when data requirements are under development. This was especially important for us. Also, NoSQL databases focus on speed and scalability. MongoDB has the connection library *PyMongo* for Python and *MongoJavaDriver* for Java.

The pipeline process is divided into several steps, where each step can be viewed as an isolated task. Although some of the stages have dependencies backwards, they can be run as isolated processes, as long as the steps before have completed. This is possible because we use a database as an intermediate storage between each step. Having this flexibility was important to us right from the start when we did not know how everything would fit together. We had an understanding of where the information would come from and how the final dataset would look like, but most of the steps needed to create the dataset were unknown when we started the project. Therefore it was important that we could look at parts of the system as isolated tasks and work incrementally to create the desired dataset. We have chosen to use MongoDB as a database as it is easy to use and requires little configuration to add new data.

GridFS GridFS is a filesystem used by MongoDB to store large files. MongoDB has a size limit on documents, which means documents cannot be larger than 16MB. The thought is that images, movies and other file formats that usually or potentially use more than 16MB should be stored in GridFS. We added this library to our pipeline since some of the tweet lists became too big for the normal MongoDB collection to handle.

Text Analysis

Text analysis methods were required to create some features in the dataset. We choose to use a framework that does the text analysis for us, as this goes beyond the scope of our task. The framework had to contain the methods we needed to create NLP features. Also, we wanted to find something that was free or cheap. An advantage was whether it was open source. Here is a review of some of the different frameworks we reviewed.

Google Cloud Natural Language API [5] The first option was the Google Cloud Natural Language API. The platform can be used to extract information in text documents. It showed a lot of impressive data after analysing text, such as entity, sentiment and syntactical analysis. The platform cost money once data was being analysed at scale, and was not open source.

[5]`https://cloud.google.com/natural-language/`
Retrieved on May 29, 2017.

Microsoft Azure Text Analytics API [6] Microsoft also its own NLP platform online, through their Azure service. Their platform can perform NLP tasks such as Sentiment Analysis, Key Phrase Extraction, Topic Detection, and Language Detection. This platform also cost money once data was being analysed at scale.

Datumbox Datumbox is a machine learning platform. The platform has a powerful API that can be used online, as well as an open source framework is written in Java, that can be used offline. Datumbox comes with a large number of pre-trained models which allow Sentiment Analysis (for both document and Tweets), Topic Classification, etc. to be performed. [7] In addition to Machine Learning (ML) methods and other pre-trained models, Datumbox include more NLP methods, including Keyword Extraction, Text Extraction and Document Similarity. Datumbox does not cost anything and is open source.

We considered to pay for one of the other text analysis tools, but when we found Datumbox, the choice was easy. The reasons why we chose Datumbox were the following:

- It was open source and free to use.
- Other researchers can replicate our work because the methods used by Datumbox are textbook methods. Unlike Googe and Microsoft services that have many of their take on NLP methods, and that it is not open source.
- Can run on our machines and configure exactly to our specifications.
- We could now experiment more with different methods and techniques. Easier to tailor the methods for our pipeline.
- Java has a simple library to connect to MongoDB, and we know how to write Java code.

Twitter API

Twitter has their API for researchers to extract data. The Twitter API used to be more open than it is today. At the time of this writing, Twitter lets researcher only use their API with usage limits. To get all their data one would have to buy the data from third parties. There were two limitations in the Twitter API that we had to solve to be able to use Twitter as our data stream of documents. The first limitation was (1) "The Twitter Search API searches against a sampling of recent Tweets published in the past seven days." [8] and the other was (2) The standard

[6] `https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/` Retrieved on May 29, 2017.

[7] `https://github.com/datumbox/datumbox-framework/` Retrieved on May 28, 2017.

[8] `https://dev.twitter.com/rest/public/search` Retrieved on May 19, 2017.

API only allows 450 search API requests every 15 minutes. [9]. We were able to find sustaining solutions to both limitations, mostly because we had identified the problems early. For limitation (1) we had to complete the necessary scripts early in the project phase to start scrape data of Twitter as soon as possible. If we had not done the necessary research, we could have jeopardised a big portion of our project. For our system to find tweets, we also had to complete the Techmeme story scraper mentioned above, before we could use the Twitter Search API. For limitation (2) we were able to soften the impact by registering two sets of Twitter Developer API keys that we could cycle through. Our implementation uses all requests available for both sets of keys every 15 min and waits until the request window timeout is lifted. In this way, we can efficiently get all the tweets we need for our dataset. We are only using the Twitter API in the portion of the pipeline that is written in Python, and for that, we use a Twitter API wrapper library called $python-twitter$ [10]

Apache Spark

Apache Spark is a big data processing framework. Some of its main features are speed, ease of use, and that it runs everywhere. [11] Spark uses an Resilient Distributed Datasets (RDD) as its data structure. “All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset. The transformations are only computed when an action requires a result to be returned to the driver program”.[12] It also has many high-level methods for working with big data and can be used with Python and Java. We have used Apache spark for many of the operations within Model B and C.

Architecture

The system used different components in the pipeline to produce the wanted datasets. As seen in Figure 4.21, the system was built on top of two separate pipelines. The first pipeline was used to create Content-Based features, for Model A, while the second pipeline was used to create features inspired by CF, for Model B and C.

[9] `https://dev.twitter.com/rest/public/rate-limits`
Retrieved on May 19, 2017.

[10] `https://github.com/bear/python-twitter`
Retrieved on May 20, 2017.

[11] `https://spark.apache.org/`
Retrieved on June 14, 2017.

[12] `https://spark.apache.org/docs/latest/programming-guide.html`
Retrieved on June 14, 2017.

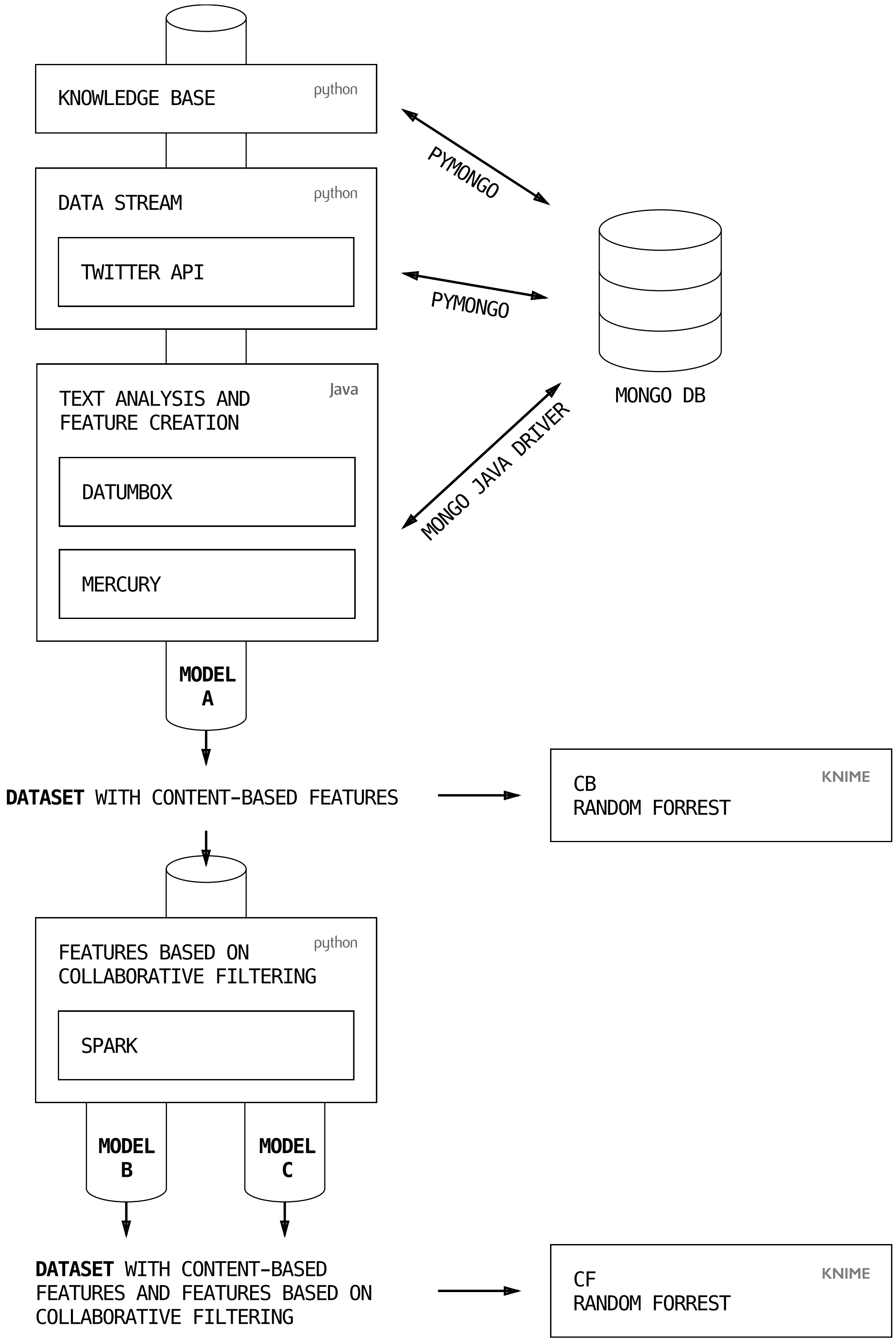

KNOWLEDGE BASE
python
DATA STREAM
python
TWITTER API
TEXT ANALYSIS AND FEATURE CREATION
Java
DATUMBOX
MERCURY
MODEL A
PYMONGO
PYMONGO
MONGO JAVA DRIVER
MONGO DB
DATASET WITH CONTENT-BASED FEATURES
CB RANDOM FORREST
KNIME
FEATURES BASED ON COLLABORATIVE FILTERING
python
SPARK
MODEL B
MODEL C
DATASET WITH CONTENT-BASED FEATURES AND FEATURES BASED ON COLLABORATIVE FILTERING
CF RANDOM FORREST
KNIME

Figure 4.21: Overview of the system architecture

First Pipeline The first pipeline can be split into three parts, called *Collecting KB-data*, *Tweets as Data Stream*, and *Text Analysis and Feature Creation*. Each step in the pipeline gathers new data or processes already collected data and saves it to MongoDB. Thus the steps involved don't need to communicate directly, which makes things easier, as components can be replaced or rerun at any time. When the first pipeline was complete, the dataset had been populated with content-based features and is called *dataset for Model A*.

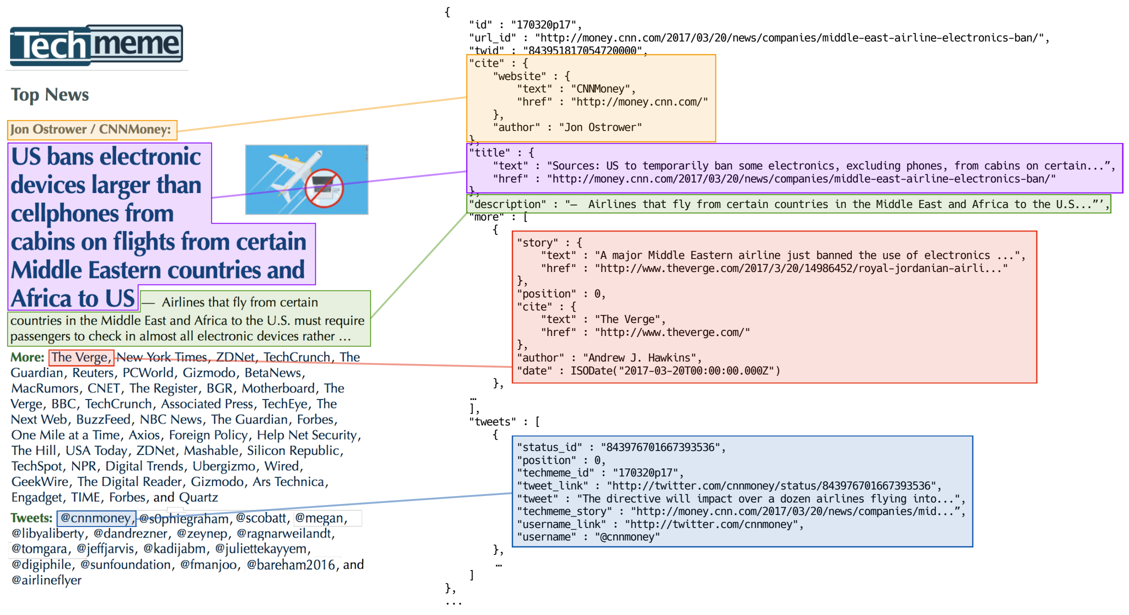
Second Pipeline The second pipeline uses the dataset from the first pipeline and adds features inspired by CF, and outputs two separate datasets, one part of Model B, and the other part of Model C. Both models compare stories or users in order to weight certain features, thus those models are inspired by CF.

Knowledge Base

The following 11 Steps are part of the first pipeline, and show how we create the dataset that is the basis for Model A and also used for both Model B and C.

Step 1. Scrape the front page of Techmeme

The first step is to scrape the front page of Techmeme. The Techmeme front page is well organised hierarchically with CSS classes and nested HTML elements to tell the different stories and web page elements apart. Techmeme lets a user go back in time to look at earlier stories on the front page. This is an important feature that makes it possible to scrape data of Techmeme at any time and is not common among news sites. We made a script that iterates over the last x amount of days, and then start at *today*$-$*x days*. We are also able to set the time of the day for each day. We scrape at the following 5 times for each day *08:00*, *12:00*, *16:00*, *20:00*, *02:00*. The times was chosen to cover all news from a day. The script inserts each story from the home page into a MongoDB collection. To reduce the complexity, we overwrite any previous entry of the story in the database. We do this because we assume that when we only look at the last occurrence, Techmeme will have had time to make the story complete with promoted Tweets and included all relevant articles.


Top News

Jon Ostrower / CNNMoney:

US bans electronic devices larger than cellphones from cabins on flights from certain Middle Eastern countries and Africa to US — Airlines that fly from certain countries in the Middle East and Africa to the U.S. must require passengers to check in almost all electronic devices rather …

More: The Verge, New York Times, ZDNet, TechCrunch, The Guardian, Reuters, PCWorld, Gizmodo, BetaNews, MacRumors, CNET, The Register, BGR, Motherboard, The Verge, BBC, TechCrunch, Associated Press, TechEye, The Next Web, BuzzFeed, NBC News, The Guardian, Forbes, One Mile at a Time, Axios, Foreign Policy, Help Net Security, The Hill, USA Today, ZDNet, Mashable, Silicon Republic, TechSpot, NPR, Digital Trends, Ubergizmo, Wired, GeekWire, The Digital Reader, Gizmodo, Ars Technica, Engadget, TIME, Forbes, and Quartz

Tweets: @cnnmoney, @s0phiegraham, @scobatt, @megan, @libyaliberty, @dandrezner, @zeynep, @ragnarweilandt, @tomgara, @jeffjarvis, @kadijabm, @juliettekayyem, @digiphile, @sunfoundation, @fmanjoo, @bareham2016, and @airlineflyer

```
{
    "id" : "170320p17",
    "url_id" : "http://money.cnn.com/2017/03/20/news/companies/middle-east-airline-electronics-ban/",
    "twid" : "843951817054720000",
    "cite" : {
        "website" : {
            "text" : "CNNMoney",
            "href" : "http://money.cnn.com/"
        },
        "author" : "Jon Ostrower"
    },
    "title" : {
        "text" : "Sources: US to temporarily ban some electronics, excluding phones, from cabins on certain...”,
        "href" : "http://money.cnn.com/2017/03/20/news/companies/middle-east-airline-electronics-ban/"
    },
    "description" : "—  Airlines that fly from certain countries in the Middle East and Africa to the U.S...”’,
    "more" : [
        {
            "story" : {
                "text" : "A major Middle Eastern airline just banned the use of electronics ...",
                "href" : "http://www.theverge.com/2017/3/20/14986452/royal-jordanian-airli..."
            },
            "position" : 0,
            "cite" : {
                "text" : "The Verge",
                "href" : "http://www.theverge.com/"
            },
            "author" : "Andrew J. Hawkins",
            "date" : ISODate("2017-03-20T00:00:00.000Z")
        },
    …
    ],
    "tweets" : [
        {
            "status_id" : "843976701667393536",
            "position" : 0,
            "techmeme_id" : "170320p17",
            "tweet_link" : "http://twitter.com/cnnmoney/status/843976701667393536",
            "tweet" : "The directive will impact over a dozen airlines flying into...",
            "techmeme_story" : "http://money.cnn.com/2017/03/20/news/companies/mid...”,
            "username_link" : "http://twitter.com/cnnmoney",
            "username" : "@cnnmoney"
        },
        …
    ]
},
...
```

Figure 4.22: The scraper will create a JSON Object for every story. And organize all story into a JSON Array for each day.

Step 2. Scrape Tweets from @Techmeme on Twitter

The only unique id of a story, scraped in Step 1, on Techmeme is formatted in this way ”$(year)(month)(day)p(Story\ Position\ That\ Day)$”. As an example, if a story were published as the first story on the 26th of April 2017 it is id would be *170426p1*. This is the only information we have about a story’s origin date and time. We found that Techmeme tweet about all their stories on Twitter and link to that tweet in every story. After Step 1, we already have this information. We used the Twitter API to request all of the Techmeme tweets and map them to stories from Step 1. If some tweets were missing, we simply took the related story out of the pipeline. In this way, we were able to set a specific time and day for almost every story.

Step 3. Merge dates from Techmeme tweets with Stories

After Step 1 and 2, we merge the dates we found in Techmeme tweets, and map them to the corresponding story. Thus, we can set a specific time for each story.

Data Stream

Step 4. Scrape Cited tweets from Twitter

Techmeme has hand-picked tweets for most of their stories, and these are the stories we use as the basis for our knowledge base, in other words, we only look at stories with promoted tweets. We only have access to the text content of a tweet,

tweet link, username for the Twitter user, tweet position in a story, and Techmeme story id. This was not enough data, so we had to request all tweets that have been promoted using Twitter's API. Data mined from Techmeme and Twitter for each tweet was then merged and saved to a MongoDB collection. This MongoDB collection contains all tweets promoted to Techmeme, which is all cited tweets.

Step 5. Scrape Tweets with Links to a story from Twitter

At this point, we only have cited tweets in our potential dataset, so in this step of the pipeline, we query the Twitter Search API for all other tweets that linked to any of the articles in a story on Techmeme. This step uses many requests towards the request window limitations set by the Twitter API. As we have two sets of keys, this process is twice as fast. For example, it will take about 2 hours to scrape tweets that link to any Techmeme story in one week. This results in 50 000 - 100 000 new tweets depending on the type of tech articles on Techmeme that week.

Step 6. Flatten stored Tweets into a Single Collection

Some articles were linked to by thousands of Twitter users, and these lists of tweets are too large for a MongoDB collection to store (see MongoDB 4.5). Therefore, many tweets are stored in GridFS and not only in its full form in a MongoDB collection. The reference key is stored alongside the other tweets in the collection. To be able to run queries on tweets, the tweets list for each story need to be flattened. We, therefore, iterate though the list of tweets for a given story and does two things. First, we check if the retrieved element is a tweet document or a reference key to a file in GridFS. If it is a GridFS file, we have to read the file, which is a JSON array of tweets, and extract each one. We also use this opportunity to strip all tweets of unnecessary data, such as *background-color* for a Twitter user's own Twitter feed. In the end, we store all flattened tweets into a new collection.

Add cited tweets We now have two sets of tweets. The first one is the set of all cited tweets, $\textbf{tweets}_{cited}$. These are the tweets that were promoted in Techmeme. The second on is the set of tweets that we just flattened, $\textbf{tweets}_{query}$. One would hope that we were able to find most of $\textbf{tweets}_{cited}$ inside of $\textbf{tweets}_{query}$, but this is not the case. We only found about 10% of $\textbf{tweets}_{cited}$ in $\textbf{tweets}_{query}$. There are more tweets about a story than those who link to an article in a story, but we have no way of knowing how a tweet is connected to a story other than the link. Therefore, we append the rest of the tweets found in $\textbf{tweets}_{cited}$ into $\textbf{tweets}_{query}$ and call this $\textbf{tweets}_{all}$. $\textbf{tweets}_{all}$ is the complete Data Stream that we use for our dataset.

Feature Creation

Step 7. Creating lookup collections

This step was added later on because some of the *aggregate*-queries in MongoDB took too long to complete. This is understandable as it would have to look at around 600 000 tweets for each *aggregate*. *aggregate*-queries are very powerful because it is possible to do group, sum and date ranges. The problem arrises when we do an *aggregate* for each tweet. Therefore we added some scripts that create helper collections for us, so that the information we need can be queried using a *find* function in MongoDB, which is a lookup, and not an *aggregate* function. This saved us much time.

Date lookup collection We assume that there are minimal tweets created at the same time. The timestamp of a tweet can be used as an approximate position. For each story, there is be a sorted list of dates for the tweets belonging to this story. If we need to find the position of a tweet relative to when the story was written, we only need to find the same timestamp in the list and count the number of items above that tweet.

User lookup collection For the two features *User Techmeme Stories*, the number of stories that a user has written about, and *User Techmeme Citations*, the number of stories that a user has written about and was cited, it was once again a problem with the time complexity of the related *aggregate* queries. We created a lookup collection that groups users and their stories.

Step 8. Scrape text content from all Articles in KB

Mercury [13] was used to scrape the content of each article. Tee API was simple to use and generated a JSON document that contained basic information and metadata about the article website, as well as the article content. After cleaning the content for HTML tags, new line characters, etc., the data was stored into the KB. Since the above process was time-consuming, between 1-2 seconds per. Website, we ran the request to scrape each article in parallel. This greatly reduced the time to scrape all stories. The article content was later used to extract terms that would represent each story in the KB.

Step 9. Sentiment Analysis on all Tweets

We use Java for the remaining steps for Feature Creation. The reason we use Java is that we heavily rely on a tool called Datumbox, which only works with Java. Datumbox is open source, which makes it easy for others to test and validate the

[13]https://mercury.postlight.com/
Retrieved on May 22, 2017.

procedures in this project. To perform sentiment analysis, we use a pre-trained model created by the Datumbox team, called *Twitter Sentiment Analysis*, and describes the model with the following description "The Twitter Sentiment Analysis model allows you to perform Sentiment Analysis on Twitter. It classifies the tweets as positive, negative or neutral depending on their context." [14]. The sentiment probability values for positive, negative or neutral sentiment are saved in a new collection in MongoDB. The reason we make collections, and then often a new collection, at almost every step is that we can easily go back and change something if changes happen along the way. If the data has in any way become dirty due to errors in the code or that the requirements and features change, it is easy to re-run the step without necessarily having to change or re-run other steps.

Step 10. Term Extraction for Tweets, Stories, and Users Bio

The method used to extract terms is called N-gram. N-gram looks at a word or words sequences and counts the occurrence of each word or sequence, and ranks them according to how often it occurs in the document. A penalty is given to sequences, where if some of the words in the sequence are used between two occurrences of a sequence, it is not likely that this sequence is special. It is important to preprocess data before using N-gram. Both stop word removal and stemming were applied to extract more relevant terms that can be compared between tweets, stories and a users bio. The used stop words can be found in Appendix 1. N-gram and text preprocessing is part of the Datumbox framework and was setup using Java. Term Extraction was applied to all tweets, stories, and users bio.

Step 11. Finalising dataset

The last step in the pipeline is to join the values stored in different collections into a feature vector for each tweet. This step is divided into 3 phases, to incrementally process data into features.

Feature Based on Collaborative Filtering

To create features inspired by collaborative filtering we used Python for both Model B and C. We mainly used the following Python libraries: (1) *numpy* [15] for matrix operations, (2) *Apache Spark* for processing bigdata, (3) *sklearn*[16] for cosine simil-

[14]`https://github.com/datumbox/datumbox-framework-zoo/`
Retrieved on May 22, 2017.
[15]`http://www.numpy.org/`
Retrieved on June 13, 2017.
[16]`http://scikit-learn.org/stable/`
Retrieved on June 13, 2017.

arity and other similarity methods, and (4) *pandas*[17], and (5) *pickle*[18] for working with data.

Model B

In the following section, we will look at the pseudo code for Model B (see Section 4.3). Let us assume that we have obtained predicted values from Model A, corresponding to the last $\frac{3}{4}$ of the dataset. That is, we have confidence scores for $\frac{3}{4}$ of the dataset, based on the first $\frac{1}{4}$ of the dataset.

The goal of Model B is to calculate similarities between users by using weighted confidence scores between users who have written about the same story.

The first step in Model B is to create the *user to story* matrix **r**. The matrix is created by looking at all users and stories and adding confidence score for a tweet in cells where a user has written about a story. If a user has written multiple tweets about the same story, we choose the confidence score with the highest value. This is because we are interested about the best confidence score for the user since this will best describe the users potentially.

Algorithm 1 Create *user to story* matrix for Model B

```
1: function CREATE-$r$(U, J)
2:     $\mathbf{r} \leftarrow$ sparse matrix
3:     for $u$ in $U$ do
4:         for $j$ in $J$ do
5:             if USER-WRITTEN-ABOUT-STORY(U,J) then
6:                 $\mathbf{r}_{uj} \leftarrow$ MAX(CONFIDENCE-SCORE($u, j$))
       return r
```

For the rest of the algorithm description, **we iterate through each Story j in J**. This loop is commented as *Story loop* in the main code for Model B, Algorithm 6

We want to find a similarity values for each story, compared to all other stories. We use ShinglesSimilarity to calculate the similarity between two stories. (Each Story has a large corpus or text that contains all article texts). We want to find similar stories so that we can use clustering of stories and expand the *user to story* matrix for a Story j. This gives us more users to compare. *TopKSimilarStories*(k, j) gives us a list of k-similar users given a j. See Algorithm 6.

We then create an extended *story to user* array containing all the users who have written about Story j or similar to j (stories limited by k). Here we just transfer the values already found in **r**.

[17]`http://pandas.pydata.org/`
Retrieved on June 13, 2017.
[18]`https://wiki.python.org/moin/UsingPickle`
Retrieved on June 13, 2017.

Algorithm 2 Create the extended *user to story* matrix for a story j in Model B

```
1: function CREATE-$\tilde{r}_j$($\mathbf{r}$, $\tilde{U}$)
2:     $\tilde{\mathbf{r}}_j \leftarrow$ sparse matrix
3:     for $j_i$ in $\tilde{U}$ do
4:         for $u$ in $\mathbf{r}^T_{j_i}$ do
5:             $\tilde{\mathbf{r}}_{ju} = \mathbf{r}_u$
       return $\tilde{\mathbf{r}}_j$
```

We are now ready to make the similarity matrix, **sim**. It is created by calculating *CosineSimilarity* between each user in $\tilde{\mathbf{r}}_j$.

Algorithm 3 Create *user to user* similarity matrix for Model B

```
1: function CREATE-$sim$($\tilde{\mathbf{r}}_j, \tilde{\mathbf{U}}_j$)
2:     $\mathbf{sim} \leftarrow$ sparse matrix
3:     for $(\tilde{u}, v)$ in COMBINATIONS($\tilde{\mathbf{U}}_j$) do
4:         $\mathbf{sim}_{\tilde{u}v} \leftarrow$ COSINE-SIMILARITY$(\tilde{u}, v)$
       return sim
```

Now we have come to the prediction part of the Model B algorithm. This is an intermediate step, in order to calculate the final weigthed average score for all users. We make the reduced array $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}}$ by calculating the dot product between $\mathbf{sim}_{\tilde{u}}$ and the transpose of $\tilde{\mathbf{r}}_j$, $\tilde{\mathbf{r}}_{j^T_{\tilde{j}}}$ for all users in $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}}$

Algorithm 4 Create *user to story* matrix for Model B

```
1: function CREATE-$\hat{r}_{\tilde{u}\tilde{j}}$($\tilde{\mathbf{r}}_j$, $\mathbf{sim}$, $\tilde{J}$)
2:     $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}} \leftarrow$ sparse matrix
3:     for $(\tilde{u}, \tilde{j})$ in $\tilde{J}$ do
4:         $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}} \leftarrow \mathbf{sim}_{\tilde{u}} \cdot \tilde{\mathbf{r}}^T_{j\,\tilde{j}}$
         return $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}}$
```

At the end of each story in the story loop, we calculate the average confidence scores for every user $\tilde{u}$ in $\hat{\mathbf{r}}_{\tilde{j}}$ for Story j, using CREATE-rc_j($\hat{\mathbf{r}}_{\tilde{j}}$, $\tilde{j}$).

Algorithm 5 Create weighted average for all users of a story j in Model B

```
1: function CREATE-$rc_j(\hat{\mathbf{r}}_{\tilde{j}}, \tilde{j})$
2:     $\mathbf{rc}_{u\tilde{j}} \leftarrow []$
3:     for $(idx_u, u)$ in $\hat{\mathbf{r}}_{\tilde{u}\tilde{j}}$ do
4:         $\mathbf{v}_{num} \leftarrow []$, $\mathbf{v}_{den} \leftarrow 0$
5:         for $(idx_j, j)$ in $\hat{\mathbf{r}}^T_{\tilde{u}\tilde{j}}$ do
6:             $\mathbf{v}_{num} \leftarrow \mathbf{v}_{num} + \hat{\mathbf{r}}_{uj} \cdot$ SHINGLES-SIM$(j, \tilde{j})$
7:             $\mathbf{v}_{den} \leftarrow \mathbf{v}_{den} +$ SHINGLES-SIM$(j, \tilde{j})$
8:         $\mathbf{rc}_{u\tilde{j}} \leftarrow \mathbf{v}_{num}/\mathbf{v}_{den}$
       return $\mathbf{rc}_{\tilde{j}}$
```

We add each rc_j to $\hat{R}$. After the Story loop is finished and $\hat{R}$ is filled up, we have the weighted confidence scores for all users in the dataset. Below is the complete algorithm for Model B. It calls the functions from the above algorithm tables.

Algorithm 6 Main algorithm for Model B

```
1: function MODEL-C
2:     $\mathbf{U} \leftarrow$ set of all users
3:     $\mathbf{J} \leftarrow$ set of all stories
4:     $k \leftarrow$ #stories to consider for cluster
5:     $\mathbf{r} \leftarrow$CREATE-$r(\mathbf{U}, \mathbf{J})$
6:     $\hat{R} \leftarrow$ sparse matrix
7:     for $j$ in $J$ do                                        ▷ Story loop
8:         $\tilde{\mathbf{J}} \leftarrow$ TOP-K-SIMILAR-STORIES$(k, j)$
9:         $\tilde{\mathbf{J}} \leftarrow j$                                    ▷ Append $j$
10:        $\tilde{\mathbf{U}}_j \leftarrow$ STORIES-WRITTEN$(U, \tilde{J})$
11:        $\tilde{\mathbf{r}}_j \leftarrow$CREATE-$r_j(\mathbf{r}, \tilde{U})$
12:        $\mathbf{sim}_j \leftarrow$CREATE-$sim(\tilde{\mathbf{r}}_\mathbf{j}, \tilde{\mathbf{U}}_j)$
13:        $\hat{\mathbf{r}}_{\tilde{j}\leftarrow}$CREATE-$\hat{r}_{\tilde{j}}(\tilde{\mathbf{r}}_j, \mathbf{sim}_j, \tilde{J})$
14:        $\hat{R}[j] \leftarrow$CREATE-$rc_j(\hat{\mathbf{r}}_{\tilde{j}}, j)$
       return $\hat{R}$
```

Model C

In the following section we will look at the pseudo code for Model C (see Section 4.4). In Model C we wish to create new features that extend some of the same features from Model A. These new features are modified versions of those corresponding features from Model A and are weighted average according to the similarity score for neighbouring users.

The main part of the algorithm is at the end of this section 11. We start by creating **A**, which is a *user to story* indicator matrix that contains a value 1 for each user u that has written about a story j. Since the matrix is a *sparse matrix*, it does not need to be filled with zeros in the non-matching cells.

Algorithm 7 Create a *user to story* matrix for Model C

```
1: function CREATE-A(U, J)
2:     $\mathbf{A} \leftarrow$ sparse matrix
3:     for $u$ in $U$ do
4:         for $j$ in $J$ do
5:             if USER-WRITTEN-ABOUT-STORY(U,J) then
6:                 $A_{uj} \leftarrow 1$
       return A
```

The next step is to create a *story to story* matrix, called **B**. For each story, we calculate the ShinglesSimilarityScore against each other story that has been written in the past. We do not want to create similarities for a story with future stories, compared to the target Story j. It also helps us when doing matrix operations later.

Algorithm 8 Create a *story to story* similarity matrix for Model C

```
1: function CREATE-B(J)
2:     $\mathbf{B} \leftarrow$ sparse matrix
3:     for $j_1$ in $J$ do
4:         for $j_2$ in $J$ do
5:             if $j_1 \neq j_2$ AND TIME-CREATED($j_1$) $\leq$ TIME-CREATED($j_2$) then
6:                 $B_{j_1 j_2} \leftarrow$ SHINGLES-SIM($j_1$, $j_2$)
       return B
```

For the rest of the algorithm description, **we iterate through each Story** j **in** J. This loop is commented as *Story loop* in the main code for Model C, Algorithm 11

The following function is called CREATE-$\tilde{U}_j$. It creates a vector of all users from **A** that has written about a story j.

Algorithm 9 Create a vector of users of a story for Model C

```
1: function CREATE-$\tilde{U}_j$($j$, $\mathbf{A}$)
2:     $\mathbf{AT} \leftarrow \mathbf{A}^T$
3:     $\tilde{\mathbf{U}}_j \leftarrow []$
4:     for $u_i \leftarrow 0$ to LEN($\mathbf{AT}_j$) do
5:         if $\mathbf{AT}_{ju_i} == 1$ then
6:             $\tilde{\mathbf{U}}_j$.APPEND($u_i$)
        return $\tilde{\mathbf{U}}_j$
```

Now that we have $\tilde{U}_j$, we are ready to create the *user to user* similarity matrix (This step is done for each Story j). The way we create this matrix is to go through all combinations of users in $\tilde{U}_j$. For each combination u and v, we find the intersection of stories that they both have written about and calculate the average score between the users, based on the story similarity found in **B**.

Algorithm 10 Create *user to user* similarity matrix for Model C

```
1: function CREATE-C($j, k, \mathbf{A}, \mathbf{B}, \tilde{\mathbf{U}}_j$)
2:     $\mathbf{C} \leftarrow$ sparse matrix
3:     for $(u, v)$ in COMBINATIONS($\tilde{\mathbf{U}}_j$) do
4:         $\mathbf{sim}_{uv} \leftarrow$ TOP-K(LOGICAL-AND($\mathbf{A}_u, \mathbf{A}_v$) $\cdot \mathbf{B}_j^T, k$)
5:         if LEN($\mathbf{sim}_{uv}$) $> 0$ then
6:             sim-value $\leftarrow$ AVERAGE($\mathbf{sim_{uv}}$)
7:             $\mathbf{C}_{uv} \leftarrow$ sim-value, $\mathbf{C}_{vu} \leftarrow$ sim-value
    return $\mathbf{C}$
```

Finally, we are ready to predict weighted averages for features. For each of the users found in $Q_j(u)$, $Q_j(u)$ being the set of users that are are l closest to the target user, we calculate new features by weighting the original features found in the row vector d_v using the similarity between users u and v. We then take the average of these new features weighted by their respective similarity scores. If the user has tweeted more than once about the same story, we calculate the average of each feature for that story and for that user.

Algorithm 11 Main algorith for Model C

```
1: function MODEL-C($\mathbf{D}$)
2:     $\mathbf{U} \leftarrow$ set of all users
3:     $\mathbf{J} \leftarrow$ set of all stories
4:     $k \leftarrow$ #stories to consider for similarity score
5:     $l \leftarrow$ #users to consider for weighted average
6:     $\mathbf{A} \leftarrow$CREATE-A($\mathbf{U}, \mathbf{J}$)
7:     $\mathbf{B} \leftarrow$CREATE-B($\mathbf{J}$)
8:     $\hat{\mathbf{D}} \leftarrow [][]$
9:     for $j$ in $J$ do                                        ▷ Story loop
10:        $\tilde{\mathbf{U}}_j \leftarrow$ CREATE-$\tilde{U}$($j, \mathbf{A}$)
11:        $\mathbf{C} \leftarrow$CREATE-C($j, k, \mathbf{A}, \mathbf{B}, \tilde{\mathbf{U}}_j$)
12:        $\mathbf{d} \leftarrow \mathbf{D}_j)$
13:        for $u$ in $\tilde{\mathbf{U}}$ do
14:            $\mathbf{c}_u \leftarrow \mathbf{C}_u$
15:            $\mathbf{v}_u \leftarrow Q_j(u)(\mathbf{c_u}, u, l)$
16:            $\mathbf{v}_{num} \leftarrow []$, $\mathbf{v}_{den} \leftarrow 0$
17:            for $v$ in $\mathbf{v}_u$ do
18:                $\mathbf{v}_{num} \leftarrow \mathbf{v}_{num} + \mathbf{C}_{uv} * \mathbf{d}_v$
19:                $\mathbf{v}_{den} \leftarrow \mathbf{v}_{den} + \mathbf{C}_{uv}$
20:            $\hat{\mathbf{d}}_{uj} \leftarrow \mathbf{v}_{num}/\mathbf{v}_{den}$
21:        $\hat{\mathbf{D}}_j \leftarrow \hat{\mathbf{d}}$
    return $\hat{\mathbf{D}}$
```

Classification

We use a data analysis framework called KNIME Analytics Platform[19] to perform preprocessing and run the RF learner. KNIME had all the tools and modules that we needed to complete the classification part of our project. We have chosen to use a RF learner to train our models. Since our goal has been to construct comparable models that can predict cited and non-cited documents from a data stream using well-crafted features. It is not a focus for us to get the highest accuracy score, even though that might be achieved using different learners. We also chose RF since it requires just a few parameters to get good results.

Using KNIME, we import our dataset into memory and perform some preprocessing on the data. We convert data to correct types, remove unused features (See Section 5.2), and remove all retweets from the original dataset. This dataset will be the basis from Model A, B, and C. We also create a Base Case model, based on popularity, which uses the number of likes that each tweet has. We will discuss this further in Experiments (See Section 5.2). After preprocessing, we split the data up into the various sets described earlier in this chapter. We add a module that runs the learner using stratified Cross Validation (CV). The learner outputs confidence scores for each model. This score is used to evaluate the performance of each model. We have created multiple KNIME workflows for this above process. These can be found in Appendix 2, and show a more thorough view of the details of the structure and implementation.

One thing the classifier needs to handle is missing values or *nan*. The dataset for Model A has a value for all features, for all users. However, when we use the classifier to calculate confidence scores for Model B and C, there are many *nan* values in the data set. This is because when we calculate a similarity between the *users and story*, *storiy and storiy*, and *user and user* similarity matrices, the links are so few. Thus, the similarity matrix becomes very sparse. In the implementation, we work with matrix structures optimised for sparsity, but when the data is exported, the values for each empty cell is given the value *nan*. It is not correct to set the values in these cells as 0. Thus, when we finish Model B and C, we add features to the original dataset where several of the values can be *nan*. It is important to handle this reliably so that the classifier does not simply ignore rows with missing values. Fortunately, RF in KNIME has a good approach to dealing with missing values. "The basic idea is to try for each split to send the missing values in every direction and the one yielding the best results (i.e. largest gain) is then used. If no missing values are present during training, the direction of a split that the most records are following is chosen as direction for missing values during testing."[20]

In this Chapter, we have outlined the basis of our three models, Models A, B,

[19] `https://www.knime.org/knime-analytics-platform`
Retrieved on June 17, 2017.

[20] `https://www.knime.org/files/node-documentation/org.knime.base.node.mine.treeensemble2.node.randomforest.learner.classification.RandomForestClassificationLearnerNodeFactory.html`
Retrieved on June 9, 2017.

and C. We also show how we create the datasets we use in this thesis. In the next Chapter, we will review the experiments we will perform and the evaluation methods that we will use to verify our results.

5 Experiments and Evaluation

In this Chapter, we will review the experiments that will be done and the evaluation methods that we will use to verify our results. In Section 5.1 we give valuable details about our datasets. In Section 5.2 we describe how we will facilitate for the experiments that we will do for Models A, B, and C. In Section 5.3 we will go through the evaluation methods that will be used to present and validate our results.

5.1 Dataset

We primarily work with two datasets, a Knowledge Base (KB) and a Data Stream. Our KB consists of stories from Techmeme, a tech news aggregator. The stream consists of tweets that link directly to one of the news articles in a story in the KB. Twitter, according to its Terms of Service, does not allow re-distribution of the downloaded data. This means that there are not many publicly available datasets from Twitter. Techmeme has no API to extract data. To our knowledge, a dataset that contains news articles concerning relevant tweets and a reservoir of incoming tweets is not publicly available. We have therefore taken on the time-consuming task to extract both datasets manually. This is a tedious and important work that allows us to evaluate our models. One of the major contributors to this master thesis is to show how the principles used in the Knowledge Base Acceleration (KBA) track also works on new datasets, through Model A. Furthermore, we want to explore the effect of adding features inspired by Collaborative Filtering (CF), through Model B and C.

Knowledge Base

We say that a KB contains different entities. An entity in our KB is a story that contains a collection of news articles about the same topic. In line with the challenge of the KBA track, we want to facilitate a human editor to find key tweets related to the story among a long stream of incoming documents. We have already explained what features we collect in ourKB in Table 4.1. We retrieve most of the data from Techmeme's website. Also, we scrape the content of each of the articles in a story through the scraping tool Mercury.

Techmeme

We scraped data from Techmeme in the period March 21st, 2017 to May 14th, 2017. During that time, 636 stories were published. Of these, 39 stories lacked cited tweets. Not all stories have quoted tweets attached to them. This is usually true of smaller news that does not create any particular engagements. We have chosen to ignore these stories to reduce the dataset and to, initially, focus on finding central tweets rather than learning to ignore them. We assume that the model will try to find cited tweets for any story. Having withdrawn these empty stories, we have 597 stories left from the period. Each of these stories has an average of 5.06 cited tweets with a standard deviation of 6.89.

Article content

Each story contains references to various news articles that mention the same story. On average, each story contains references to 14.12 articles with a standard deviation of 12.26. We use the Mercury service to extract title and text for each article. Mercury downloads the HTML code of each article and tries to extract relevant values about the document. The service usually finds the articles and the correct metadata. However, errors occur in which Mercury fails to find article text or include items from the website such as menu navigation. We do not monitor the results continuously, so if any errors occur in the collection, we will not take this into account. We link the text from all the articles and create a large corpus of documents. The effect of errors in obtaining certain articles is thus watered out. By putting all the documents together, the keywords that are common to all articles will be more clearly displayed. This corpus of documents is the basis for term extraction. There are many approaches to how to make term extraction. We have chosen to use a simpler model based on n-grams so we can easily run the operations on our local machine. Providers like Google and Microsoft have models that are far more advanced and can deliver cleaner results. However, these services cost much money for the number of documents we would need to process.

Incoming Document Stream

We need to create a dataset that contains all the tweets from the same period as stories in KB. Over a period of almost two months, it has been tweeted more than 30 billion tweets [1]. It is not possible for us to download such a large amount of tweets. Similar to Balog et al. (2013), we use a filter as the first step in the classification process by only retrieving documents that mention an entity in KB. In our case, we say that a tweet mentions a story by linking directly to one of the

[1] See Twitter's IPO filing
`https://www.sec.gov/Archives/edgar/data/1418091/000119312513390321/d564001ds1.htm`
Retrieved on June 17, 2017.

articles in a story. We have outlined what features we retrieve and generate in Table 4.3.

There are hardly any publicly available datasets from Twitter due to restrictions Twitter has put on re-distribution of tweets. For this reason, we must collect tweets through Twitter's developer API. This API has certain limitations that we have explained in section 2.1, but it allows us to retrieve the tweets we need. During the scraping period, we collected 612 145 tweets, each of which tweet links to one of the articles in one of the 597 stories. Each story has an average of 1030.5 tweets with a standard deviation of 1808.09 tweets.

We see that the number of cited tweets is much less than the number of tweets per story, 5.06 cited tweets per story against 1030.5 tweets per story. We want to trim the dataset to increase this ratio and to reduce the number of tweets, so the computational load does not become too huge.

A large portion of tweets in the dataset are retweets, where one user shares another's tweet without adding new information. These tweets only create noise and therefore be trimmed from the dataset. There were 319 774 retweets in the period from March 21st, 2017 to May 14th, and only one of those was cited by Techmeme. We, therefore, remove retweets from the dataset before moving on. The reduced dataset now has 322 699 tweets, and there are 541.27 tweets per story with a standard deviation of 691.55 tweets.

To further reduce the dataset, we removed tweets that are written later than six hours after a story was published. On average, 20 stories are published each day on Techmeme. Only ten stories are promoted at each point during the day. This means that all stories are exchanged on average twice a day. There is, of course, some variation here as some major stories can be promoted for over a day while other minor stories are exchanged after a few hours. However, we can further see that the tweets that are cited are mainly written around the publication of the article. We measure this time range through a feature we call *tweet_time_-between_story_and_tweet*. Since there are a few extreme outliers, we make a box plot below in log scale. We can see that if we set a limit of six hours, we only leave out a few cited tweets from our dataset.

When we remove tweets written more than six hours after a story is published, we reduce the number of cited tweets in the data set from 2925 to 2641.

Unused Features

The following features were part of the original dataset, but had to be removed, because they can look into the future.

- Normalised Tweet position for News Article
- Normalised Tweet position for Story
- Number of Tweets for news article last 24 hours
- Number of Tweets for Story last 24 hours

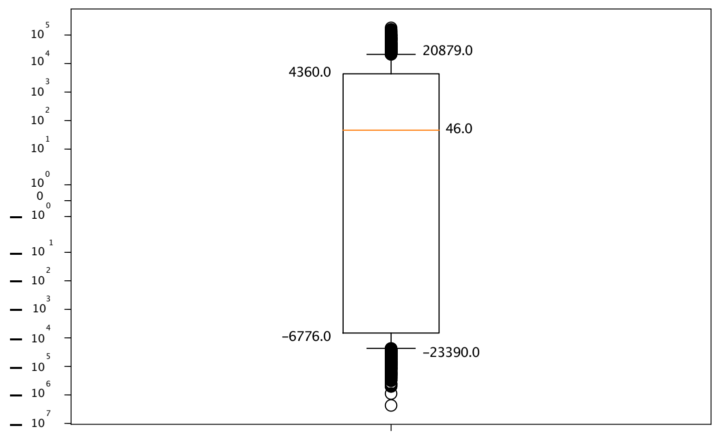


Figure 5.1: Time between cited tweets and the published story time

- Number of Tweet retweets
- Number of Tweet likes

We can not include features about likes and retweets. The number of likes and retweets are calculated relative to when the tweet was retrieved through Twitter's API. This means that the tweets we scrapped will have the number of likes and retweets accumulated over time and often more than one day. The reason why it is important not to include these features is that popular tweets that get many likes or retweets would receive further attention in the form of more likes and retweets. If we had included these features in the training set, the model would have access to the ground truth indirectly. This makes the model biased so that if we later predicted tweets continuously, the model would give too much emphasis on the number of likes and retweets. The incoming tweets would have very short likes and retweet history, unlike the values the model originally taught, which has likes and retweets accumulated over time and usually over more than one day. Unfortunately, we must ignore the number of likes and retweets.

Furthermore, we remove features that are normalised according to a given period, as in our case were the last 24 hours. We also ignore features that make recordings from the last 24 hours.

Distribution of data between training, testing and similarity scores

As we will look at in the next section, we will conduct experiments for Model A, B, C and the Popularity model. All of these models will train and predict on the last half of the dataset. We do this so that we can use the first half of the dataset for equality calculations for Model B and C.

In Model B, we use the first quarter to train and predict confidence score to use for similarity history. The similarity calculation uses the last 3/4 of the data so that when we predict from the midpoint of the dataset, the first stories, from that point onwards, will have history equivalent to a quarter of the dataset. History improves when the model predicts more data. The final training and prediction of the complete dataset, with the new feature based on CF, will be done for the last half of the data. We visualize this in Figure 4.5

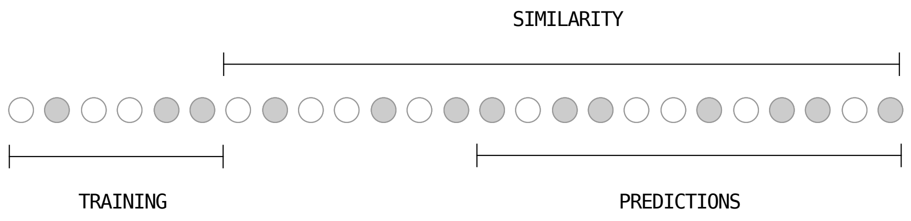


Figure 4.5: How data is used in Model B (repeated from page 41)

Model C uses the entire dataset to create similarity scores. However, it only uses the last half of the dataset to train the model and give final predictions. This means that the first stories after the centre point have similarity history for the first half of the data.

5.2 Experiments

We have three goals with our experiments. The first is to explain how long the model will look at tweets after a story has been published. The second is to investigate how Model A, which is the starting point for our work, is compared to a model based on popularity. The last goal is to investigate improvements Model B and C have over Model A.

Datasets

Since the dataset is quite large with an original size of 612 145 tweets, we have looked at different ways to reduce the dataset. We have already described how we reduce the dataset in half by removing retweets since only one cited tweet was a retweet.

In this experiment, we want to look at the effect of pruning the tweets written at 1, 2, 6, 24 or 48 hours after a story has been published. We have already seen a box plot of when cited tweets are written relative to when a story is published (see Figure 5.1). We take the pruned datasets and run them through Model A, as described below. We evaluate the different runs by looking at how it affects max F_1 score.

Since we can see from the box plot that most cited tweets are written early to when a story is published, we can expect that as we reduce the dataset, the results will be better. The ratio between cited tweets and regular tweets increases and the noise in the dataset is reduced.

Base-case

To determine whether our ground work in Model A is good, we need a *base case*. The Base-case is based on a popularity indicator, namely the number of likes a tweet has received. As mentioned in the previous section, we can not use the number of likes from the dataset to create Model A, B, or C as the number of likes is extracted at different times and usually several days after the tweet was written. In other words, if we received a new tweet that needed to be classified, we could not classify the tweet until after some time, to know the number of likes the tweet had received over time. This would make the model unresponsive and not very useful.

We still want to use the number of likes to create the Base-case because the number of likes is in itself a good indicator and easy to understand. We will not use Base-case to predict new data. Furthermore, it will be to the Base model's advantage that it can gain insight of the outcome, viewed from a popularity perspective, through the history embedded in the number of likes.

We use a dataset that only includes tweets written within the first six hours after a story is published. We then retrieve the last half of the data to make it comparable to the other models. The only feature we build the model on, as said, is the number of likes. To get the best balance between variance and bias, we use stratified ten-fold cross-validation for model selection (Kohavi et al., 1995). The model is based on a Random Forest (RF) learner with 256 different trees. The split criterion is the Information Gain Ratio, as we have had better results with this compared to Gini Index.

We evaluate the model by looking at the maximum F_1 score and the shape of the Precision-Recall (PR) curve.

Model A

Through this experiment, we want to evaluate one of the major contributions of this master thesis, namely Model A. This is the first model that utilises all the features described in Section 4.2, except for features we have had to take out as described in Section 5.1. An overview of all the features used in Model A is given in the table below. Many of these features are inspired by the work done by Balog et al. (2013). Model A helps to validate their work, and others work within the KBA track, using similar methods, but on a new dataset.

Features
hot_term_count_10
hot_term_count_2
hot_term_count_5
story_date
story_document_position
story_entity_related
story_featured_article
story_term_count
story_term_percentage
story_term_score
story_word_count
tweet_date
tweet_position_article
tweet_position_story
tweet_sentiment_negative
tweet_sentiment_neutral
tweet_sentiment_positive
tweet_time_between_story_and_tweet
tweets_pr_story_last_1_hours
tweets_pr_story_last_2_hours
tweets_pr_story_last_3_hours
user_bio_term_count
user_created_at
user_favorites_count
user_followers_count
user_friends_count
user_is_author
user_name_mentioned
user_statuses_count
user_techmeme_citations
user_techmeme_stories
user_verified

Table 5.1: Features used in Model A

We use the same dataset as Base-case uses, where we only include tweets written within the first six hours after a story is published. To make the results comparable to the other models, we only use the last half of the data. The model is based on a RF learner with 256 different trees. Split criterion is Information Gain Ratio as we have had better results with this than Gini Index, which is commonly used with RF. Furthermore, we use ten-fold cross validation with stratified sampling to get more accurate estimates of prediction error.

The RF predictor gives us a confidence score for each prediction. We need to decide which cutoff point we will use to determine if something is to be classified as central or non-central. To calculate the optimal cutoff value, we create a graph showing different F_1 scores along the y axis and the various cutoff values along the y axis. We then select the cutoff value that gives the highest F_1 score. This is according to the KBA track its evaluation goal[2], which also looks for the highest F_1 score.

In addition to finding the maximum F_1 score, we look at the shape of the PR curve. We also do a simple feature analysis to see which features are most commonly used by the RF learner.

Although the model based on popularity is the *base case* for all models, Model A becomes a *base case*, sort to speak, for Model B and C. In these latest models, we will add new features to Model A and see if performance is improved. We then consider performance improvements relative to Model A.

Model B

The goal behind Model B is to create a new feature inspired by principles from CF. We will add this feature to the data from Model A and run Model A again with the extra feature. Let's look at Model B overview.

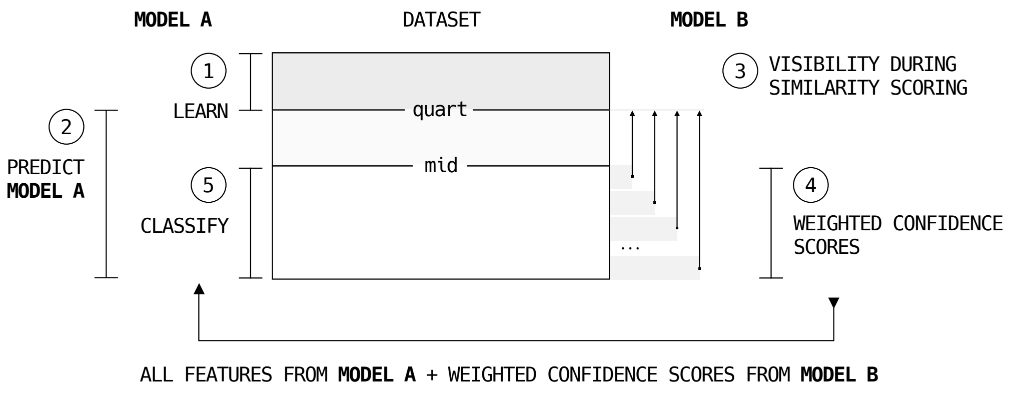

MODEL A
DATASET
MODEL B
1
LEARN
quart
3 VISIBILITY DURING SIMILARITY SCORING
2
PREDICT MODEL A
mid
5
CLASSIFY
4
WEIGHTED CONFIDENCE SCORES
...
ALL FEATURES FROM MODEL A + WEIGHTED CONFIDENCE SCORES FROM MODEL B

Figure 4.4: Model B overview (repeated from page 40)

An important component of CF is to determine the similarity between users (step 3 in the Figure). To calculate equality between all users and all stories we need a basis for determining equality. In Model B, we use Model A's confidence score (whether a given tweet should be classified as cited, $P(cited = 1.0)$) as the basis for similarity (step 2 in the Figure). As previously described, it is important that Model A not predict values for tweets that have been part of the training of the model (Step 1 and 2 in the Figure). This would lead to overfitting and will result in uneven results between tweets that were part of the training and tweets

[2]`http://trec-kba.org/trec-kba-2014/vital-filtering.shtml`
Retrieved on May 14, 2017.

that have not been. This is particularly the case when we later wish to use the model for brand new data.

The second critical component of CF is to predict new values for a user based on the user's similarity to his neighbours (step 4 in the Figure). For us to be able to predict new values, we need similarities based on a sufficiently long history to identify similarity. We have made an attempt where we predict values for users who initially have no similarity statistics. The results of such runs are much worse than the results from runs where there is similarity history. This problem is known as cold start and is a known problem in recommender systems.

We first begin to predict values for users after the midpoint *mid*. If we let Model A in step 1 in the Figure train on data up to *quart*, Model B can use history from *mid* to *quart* to calculate similarity for the first users after *mid*. The window that Model B can use to calculate similarity grows as it evaluates new tweets in step 4 in the Figure.

It is not clear where to put the *quart* split indicator as we must take into consideration both the quality of Model A's predictor and Model B's ability to calculate good similarity values. Since it is quite expensive to experiment with different positions of *quart*, we have chosen to put *quart* in the middle between the start of the data set and *mid*. We think that our choice is a good middle ground for the balance of prediction and similarity history.

A challenge that often appears in recommender systems is sparsity. Two users can easily not have any matched items and have a low or no similarity value. If this happens to many users, there are no values left to predict new values. To overcome this challenge, we have introduced clustering in Model B. Instead of just considering users from the target story; we also look at similar neighbourhood stories. This expands the number of users that can be used to calculate the similarity between users. When the similarity values are calculated, users from neighbouring stories contribute their confidence score according to how close the story is to the target story. In this experiment, we will attempt to use varying sizes of the cluster. We estimate the cluster values $k = \{5, 10, 25, 50, None\}$, where $None$ means we have no limit to how many neighbours that a story is compared to.

When Model B is ready to calculate predicted confidence scores, these values will be appended to the original data in Model A. We then run Model A again to see if the new feature provide a better result.

In both cases where we drive Model A in Model B we use a RF learner with 256 trees. Information Gain Ratio is used as the model's split criterion. We use ten-fold cross validation with stratified sampling to get more accurate estimates of prediction error.

We calculate the maximum F_1 score in the same way as we do in Model A. If Model B manages to produce higher F_1 maximum score than Model A, we would like to test if Model B is significantly better. To do this, we use McNemar's test where the null hypothesis is that the models have the same error rate. We also do a simple feature analysis to see which features are most commonly used by RF learner.

Model C

Model C has the same goal as Model B, which is to produce new features (Model B only add one new feature) and add them to Model A's data to train a new model. Model C differs from Model B in how it defines similarity between users and how it predicts new values.

The similarity between users for a given story in Model C is defined through two steps. The first step is to find all the stories that two users have tweeted about. The second step is to let each of these stories contribute to a similarity score, based on how similar the target story is to each of these intersecting stories. We then divide the sum of the story similarities with the number of stories so that we get an average similarity score. We can see from step 1 in the Figure that we start creating similarities for users who wrote tweets after the midpoint *mid*. This ensures that we can use half of the dataset as similarity history.

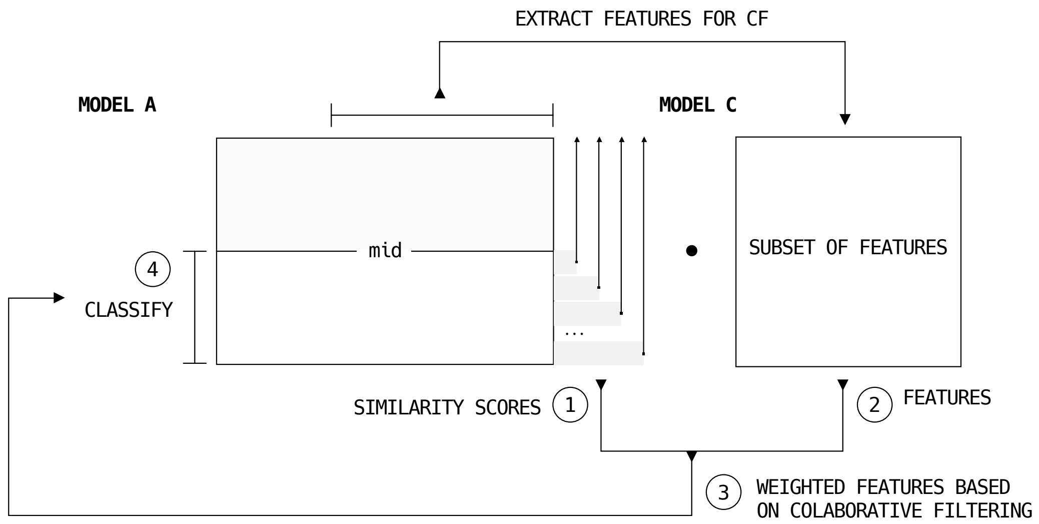


Figure 4.16: Overview of Model C (repeated from page 47)

After we have calculated similarity values between users in the dataset, we extract a set of features for each of these users from the original dataset (step 2 in the Figure). An overview of all the features that we extract from the original data set is given in the table below. We bring as many features from the original data set as we can. We ignore all features that include data about a story, as those features will be similar to all other users in the same story. We also look beyond *tweet_date* and *story_date*.

Features
hot_term_count_10
hot_term_count_2
hot_term_count_5
tweet_position_article
tweet_position_story
tweet_sentiment_negative
tweet_sentiment_neutral
tweet_sentiment_positive
tweet_time_between_story_and_tweet
tweets_pr_story_last_1_hours
tweets_pr_story_last_2_hours
tweets_pr_story_last_3_hours
user_bio_term_count
user_favorites_count
user_followers_count
user_friends_count
user_is_author
user_name_mentioned
user_statuses_count
user_techmeme_citations
user_techmeme_stories
user_verified

Table 5.2: A list of the extra features used to create new features in Model C

We use these features to create new predicted features, weighted according to the similarities between users and target users.

Model C receives two different parameters k and l. Parameter k says how many stories we will compare two users to. If $k = 2$ and user u and user v has 5 stories in common, the two stories that are most like target story j will be used in the calculations. The idea is that two users can write tweets about several different topics that do not overlap. By setting k to a low number, one can premiere users who have some stories in common, even though they also write about different topics.

The l parameter sets the number of users to use as the basis for the final prediction. If we put $l = 2$ and there are hundreds of users who have written about target story j, the new predicted feature would only be based on the two closest neighbours. The idea is that you may not want to water out the results by bringing users who are not similar to the target user. On the other hand, users who have a low resemblance to target user u will also contribute very little. There may be signals from these users that one would also like to include. We experiment with $k = \{5, 10, 25, 50, \infty\}$ and $l = \{0, 5, 25, 50, 75, 100\}$.

We think an interesting contribution from Model C is that the model lets the final classification algorithm (step 4 in Figure 4.16) determine which features are important. In Model B, however, a single feature is created that can be viewed as a summary of the importance of a tweet. We hope this flexibility will make Model C a strong model. Furthermore, the similarity is not determined based on performance, as Model B does through confidence scores, but the similarity is determined based on the topics a user writes about.

When Model C is ready to predict new features, these values will be appended to the original data in Model A. We then run Model A again to see if the new values give a better result. When we run Model A we use a RF learner with 256 trees. The model's split criterion is the Information Gain Ratio. We use ten-fold cross validation with stratified sampling to get more accurate estimates of prediction error.

We calculate the maximum F_1 score the same way we do in Model A. If Model C can produce a higher F_1 maximum score than Model A, we would like to test if Model C is significantly better than Model A, using McNemar's test. We also do a simple feature analysis to see which features are most commonly used by the RF learner.

5.3 Evaluation

All data we have retrieved from Twitter is annotated. When we scraped Techmeme, we got access to which tweets are cited. This helps us know the ground truth, and show that we work with supervised learning. Here we will explain the evaluation methods described in this chapter.

Precision, Recall and F-measure

One of the first evaluation methods one sees in machine learning is accuracy. It takes the number of correct predictions and divides it to the total number of items in the dataset. The problem with this metric is that it does not give us very useful results when we work on imbalanced data. One can quickly get the impression that the model is doing very well as most classes are *negative*, and the model also learns to classify most examples as *negative*.

A better metric is, therefore, precision and recall. Precision states how much of the data is classified as *positive* that is actually *positive*. Recall states how much of the classified data is *positive* of all who should be *positive*. We know that there are many tweets out there that contain useful insights and provide new perspectives. Concerning the dataset we are working on, it is more important for us that the model has high recall and then also includes suggestions for cited tweets beyond those annotated as cited. In other words, recall is more important than precision for us.

It is not a trivial task to optimise for recall alone. As we have described in the experiments, the different models will give a confidence score, which is the model's confidence that the tweet should be cited. We must then decide which cutoff value we will use to determine whether a tweet should be classified as cited or not for a given model. If one set the cutoff point very low, say 0, then all tweets will be classified as cited, but precision will then be 0, and it will help no one. Therefore, we will apply F_1 measure that looks at a weighted harmonic average between precision and recall.

To find the best F_1 measure, we need to evaluate all cutoff values. We then choose the value that gives the best F_1 measure. This is how the contributions in the KBA track weighted their results.

We can also plot precision and recall in a PR curve. We plot Recall along the x axis and Precision along the y axis. The goal is to have as high value as possible of both and therefore be in the upper-right-hand corner.

McNemar's test

When we want to evaluate whether two models are significantly different or not, we choose to use McNemar's test. McNemar's test is a statistical test used on paired labelled data. The zero hypothesis is that the models we are testing have the same error rate. If we get a p-value below 0.05, we can reject the null hypothesis. Then we can not declare that the models are alike.

We provide the contingency table for each McNemar test we perform to display the number of uniquely misclassified examples for each classifier (n_{01} and n_{10}), in addition to the examples, both classifiers got wrong (n_{00}) and both got right (n_{11}). Formally, we say that the null hypothesis is that $n_{01} = n_{10}$.

n_{00}	n_{01}
n_{10}	n_{11}

Feature analysis

A RF learner extracts small samples of data, both row-wise and column-wise. For each sample, it makes a decision tree. Features selected early or high in the tree create the cleanest divisions of the data and are therefore also the most important. The model keeps track of which features are used on the top three levels of the tree and how often each feature was candidates for the three levels.

When we calculate the importance of a feature, we calculate the proportion of usage each feature has within each of the three levels. If feature a was used 32 times and a candidate was 40 times on level 1, its usage share would be $\mathrm{r}_1 = 0.8$ Furthermore, we weigh contributions from levels 1, 2 and 3. We use the following weights for the different levels.

$$\mathrm{w}_1 = \frac{3/3}{2}, \quad \mathrm{w}_2 = \frac{2/3}{2}, \quad \mathrm{w}_3 = \frac{1/3}{2} \tag{5.1}$$

We divide by 2 to normalise the results. The total importance of a given feature is the sum of the usage share weighted according to its level.

$$\text{importance} = \sum_{n=1}^{3} \mathrm{r}_n \cdot \mathrm{w}_n \tag{5.2}$$

Now that we have looked at the datasets, the experiments and the evaluation methods, we are ready to continue to the next Chapter, where we will present our results and discussion of our findings.

6 Results and Discussion

In Section 6.1 we will present the results. In Section 6.2 we will discuss our findings. We will review all models A, B, and C in both sections.

6.1 Results

Datasets

We run Model A using variations of the dataset with imposed temporal constraints on what tweets it should include. We run the model with the following parameter $t = \{3600, 7200, 21600, 86400, 172800\}$. The model will only look at tweets that have been written within t seconds since a story was published.

When we get the classification results from the model, we get a confidence score that says how sure the model is that a given tweet should be classified as cited or not. We need to decide what cutoff value we want to use to say if a tweet should be cited or not. We try each cutoff value and see which F_1 score we get. We can visualise each of these runs in a graph.

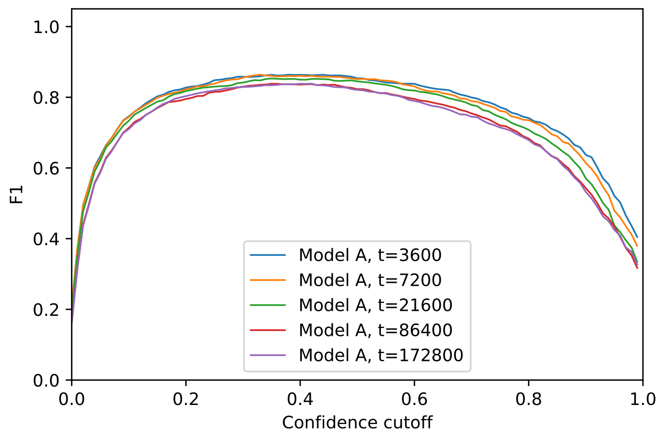


Figure 6.1: F_1 values for every cutoff point for time constrained data on Model A

Maximum F_1 score is the peak of the graphs in the model above. They are listed in the table below. The best results are marked in bold.

F_1	cutoff	Model
.864	.38	Model A (t=3600)
.864	.33	Model A (t=7200)
.853	.35	Model A (t=21600)
.838	.35	Model A (t=86400)
.838	.42	Model A (t=172800)

Table 6.1: Maximum F_1-scores for time constrained data on Model A

We can see that when the dataset is small, the F_1 score increases. We can represent the same runs in a Precision-Recall (PR) curve. We see the same trends as we saw in the maximum F_1 table, that the smaller datasets get better results. In a PR curve, we want to have both High Precision and High Recall.

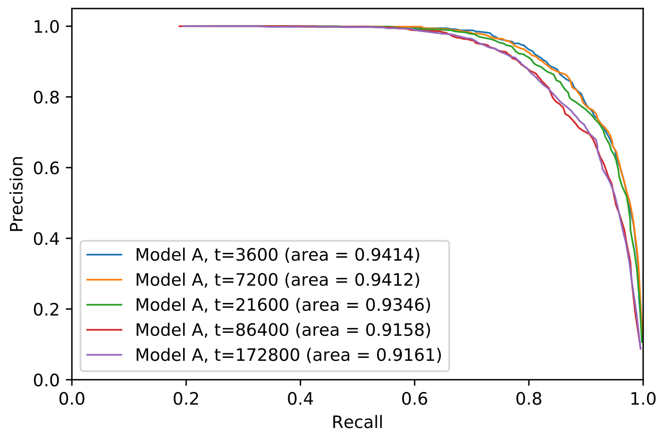


Figure 6.2: PR curve for time constrained data on Model A

Moving forward, we will use Model A with $t = 21600$. The model takes six hours of tweet history for each story. We feel it is important that the model is realistic. If we choose a lower t value, many cited tweets are omitted that should otherwise have been included. The model would then not capture tweets that would normally be promoted at Techmeme. $t = 21600$ fulfils the requirement to represent the population and at the same time gets a good score. All the models in Model B and C below run with datasets where $t = 21600$.

Model A vs. Base-case

We compare Model A versus Base-case, which is a model based on the popularity number of likes. We see that Model A is considerably better. This shows the strength of Model A.

F_1	cutoff	Model
.426	.00	Base-case
.853	.35	Model A (t=21600)

Table 6.2: Maximum F_1-scores from Model A and Base-case

The results from the table are shown in the PR curve. The Base-case model never gets any values with Recall over 0.5. We do not feel the need to present the results from McNemar's test formally as the differences between these two models are very large.

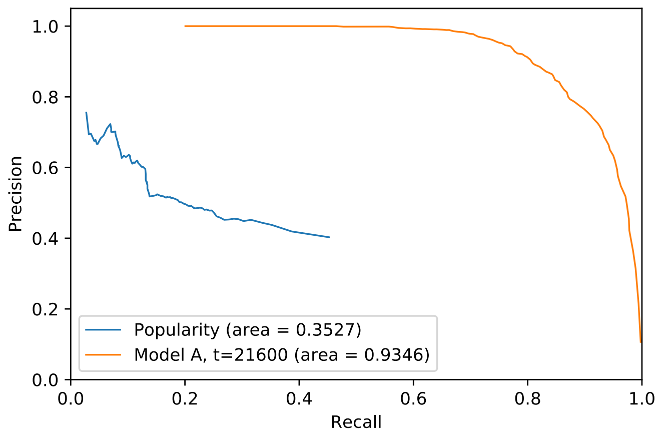


Figure 6.3: PR curve for Base-case and Model A ($t = 21600$)

Model B

We run Model B with five different parameter values $k = \{5, 10, 25, 50, \infty\}$. Maximum F_1 scores are listed in the table below.

F_1	cutoff	Model
.853	.35	Model A (t = 21600)
.858	.42	Model B (k = 5)
.860	.42	Model B (k = 10)
.858	.38	Model B (k = 25)
.862	.36	Model B (k = 50)
.857	.38	Model B (k = ∞)

Table 6.3: Maximum F_1-scores from Model A and Model B

We can see that Model B achieves a higher maximum F_1 score than what Model A achieves. The best parameter in Model B is $k = 50$ which achieves a F_1 score of 0.862 against Model A's ($t = 21600$) score of 0.853. This is an improvement of 1% points. We are very pleased with this result.

Therefore, we would like to see if Model B ($k = 50$) is significantly better than Model A. To investigate this, we make McNemar's test of the model's error rates. First, we have to make a contingency table. When Model A is the first classifier and Model B ($k = 50$) is the second classifier, we get the following table.

$n_{00} = 343$	$n_{01} = 50$
$n_{10} = 24$	$n_{11} = 103190$

Table 6.4: Contingency Table for Model A and Model B ($k = 50$)

When we run McNemar's test based on these values, we get a test statistic of 8.45 with an associated p-value of 0.0037. Since this is lower than the probability threshold of 0.05, we can reject the null hypothesis and retain the alternative hypothesis that Model A's error rate is significantly different from Model B's ($k = 50$) error rate. We see that Model B only makes 24 (n_{10}) wrong in addition to the 343 (n_{00}) errors the models have in common. Model A makes 50 (n_{01}) errors where Model B does not and thus Model A has 27 more errors in total than Model B.

The differences between Model A and Model are not nearly as big as the differences between Model A and Base case, which can be illustrated by this PR curve.

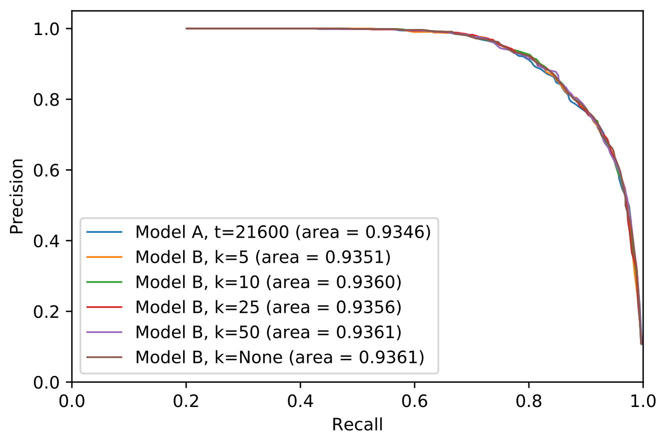


Figure 6.4: PR curve for Model B and Model A ($t = 21600$)

Model C

Model C receives two parameters k and l. We try five different parameter values for k and six different parameter values for l. It provides 30 parameter combinations. Maximum F_1 scores from all the runs are listed in the model below. The best runs for each k are in bold. The best value across all parameter combinations is underlined.

k \ l	5	25	50	75	100	∞
5	.856	.857	.857	.855	.855	**.858**
10	.855	.854	.853	.856	<u>**.859**</u>	.855
25	.853	**.856**	**.856**	**.856**	.854	.855
50	.855	.855	.855	.854	.857	**.858**
∞	.854	.853	.851	.855	.854	**.857**

Table 6.5: Maximum F_1 scores for each combination of k and l for Model C

The best parameter combination from Model C is $k = 10$ and $l = 100$. It achieves a higher maximum F_1 score than Model A achieves, with a F_1 score of 0.859 against Model A's score of 0.853. The difference here is only at 0.6% points.

We want to see if Model C ($k = 10, l = 100$) is significantly better than Model A. To investigate this, we make McNemar's test and create a contingency table. Model A is the first classifier and Model C ($k = 10, l = 100$) is the second classifier.

$n_{00} = 322$	$n_{01} = 71$
$n_{10} = 49$	$n_{11} = 103165$

Table 6.6: Contingency Table for Model A and Model C ($k = 10, l = 100$)

When we run McNemar's test based on these values, we get a test statistic of 3.68 with an associated p-value of 0.0552. Since this is higher than the probability threshold of 0.05, we must keep the null hypothesis that the models have the same error rate.

Feature Analysis

We will here do a feature analysis to look at the most important and the least important features. Table 6.7 lists the 15 best features and the five worst for Model A ($t = 21600$), Model B ($k = 50$) and Model C ($k = 10, 1 = 100$).

We can see from the table that the most important features are about the tweet's meta data and not about the content of the tweet in itself. This may not be unexpected when a tweet contains so few words. Our findings are also consistent with the results Balog et al. (2013) presented in their research.

For example, article position is important. In a story at Techmeme, there is only one main article. Furthermore, the story also links to other articles that also write about the story. The main article has story position 0. There are occasional tweets that write about articles other than the main article being cited. We also see that information about whether the user is verified or whether it is the author of a story is important.

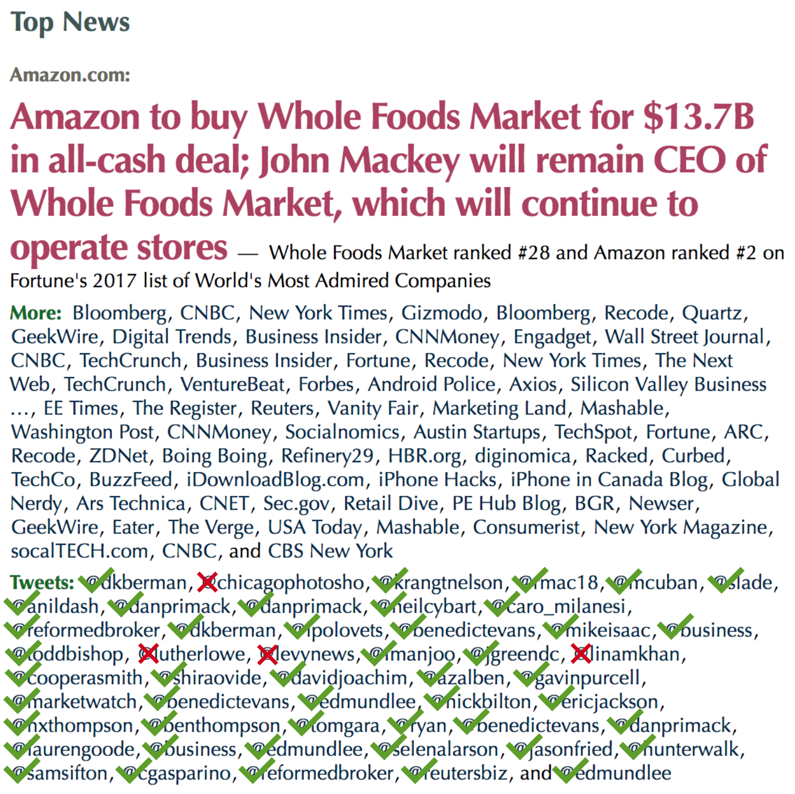
We know that Model B is significantly better than Model A. From the table, we see that the new feature, cf plays an important role and is the 14th most important feature. Model C is also better than Model A, but not significantly better. We see that some features based on Collaborative Filtering (CF) appear high on the list of important features. At the same time, the four worst features for Model C are from the new feature set we made for the model. It is intersting to see how the classifier chooses which features it considers important. This is a strength of the model and we think the approach in Model C has potential, even though its potential was not fully met through our specific implementation compared to Model B.

A Spot Check

Late in the evening three days before the master's thesis was delivered, Amazon released a bomb and announced that they had purchased Whole Foods. The news quickly appeared on Techmeme and Twitter cooked. We wanted to see how well the model would do to identify central Tweets. At the moment we retrieved data, 50 tweets had been quoted on the front page of Techmeme. We scrapped Techmeme and Twitter for new data and collected 8215 new tweets. To make things easy, we only generated the features necessary to run the data through Model A. Then we retrieved Model A trained on data over a month ago and used it to predicted cited tweets from the new test set of 8215 Tweets. The system identified 46 of the 50 central tweets. Also, it found 17 tweets where the vast majority were good alternatives to cited tweets. Figure 6.5 visualises the result of this small experiment. We were very impressed with what is possible with big data and machine learning.

Model A		Model B (k = 50)		Model C (k = 10, l = 100)	
Feature	Importance	Feature	Importance	Feature	Importance
tweet_position_article	0.784	tweet_position_article	0.780	tweet_position_article	0.817
tweets_pr_story_last_1_hour	0.604	tweets_pr_story_last_1_hour	0.642	tweets_pr_story_last_1_hour	0.634
tweets_pr_story_last_2_hour	0.552	tweets_pr_story_last_2_hour	0.507	user_techmeme_citations	0.561
user_techmeme_citations	0.442	tweets_pr_story_last_3_hour	0.433	tweets_pr_story_last_2_hour	0.517
tweet_position_story	0.441	tweet_position_story	0.390	tweets_pr_story_last_3_hour	0.471
tweets_pr_story_last_3_hour	0.414	user_verified	0.382	tweet_position_story	0.441
user_name_mentioned	0.407	user_techmeme_citations	0.382	user_name_mentioned	0.370
story_document_position	0.297	user_name_mentioned	0.365	user_verified	0.349
user_verified	0.290	story_document_position	0.300	story_document_position	0.346
story_featured_article	0.290	user_followers_count	0.238	story_featured_article	0.277
user_followers_count	0.233	story_featured_article	0.228	user_followers_count	0.247
user_is_author	0.150	user_is_author	0.171	cf_user_techmeme_citations	0.241
user_favorites_count	0.117	tweet_sentiment_neutral	0.117	cf_user_techmeme_stories	0.225
user_created_at	0.105	cf	0.115	user_is_author	0.171
story_entity_related	0.103	user_favorites_count	0.112	user_created_at	0.157
...		...		...	
time_between_story_tweet	0.010	story_term_percentage	0.015	user_friends_count	0.003
hot_term_count_2	0.007	hot_term_count_10	0.014	cf_user_statuses_count	0.002
hot_term_count_10	0.005	time_between_story_tweet	0.010	cf_hot_term_count	-
user_friends_count	0.003	user_friends_count	0.005	cf_tweet_sentiment_neutral	-
user_statuses_count	-	user_statuses_count	0.002	cf_tweet_sentiment_positive	-

Table 6.7: Best and worst features based on feature occurrences



Top News

Amazon.com:

Amazon to buy Whole Foods Market for $13.7B in all-cash deal; John Mackey will remain CEO of Whole Foods Market, which will continue to operate stores — Whole Foods Market ranked #28 and Amazon ranked #2 on Fortune's 2017 list of World's Most Admired Companies

More: Bloomberg, CNBC, New York Times, Gizmodo, Bloomberg, Recode, Quartz, GeekWire, Digital Trends, Business Insider, CNNMoney, Engadget, Wall Street Journal, CNBC, TechCrunch, Business Insider, Fortune, Recode, New York Times, The Next Web, TechCrunch, VentureBeat, Forbes, Android Police, Axios, Silicon Valley Business …, EE Times, The Register, Reuters, Vanity Fair, Marketing Land, Mashable, Washington Post, CNNMoney, Socialnomics, Austin Startups, TechSpot, Fortune, ARC, Recode, ZDNet, Boing Boing, Refinery29, HBR.org, diginomica, Racked, Curbed, TechCo, BuzzFeed, iDownloadBlog.com, iPhone Hacks, iPhone in Canada Blog, Global Nerdy, Ars Technica, CNET, Sec.gov, Retail Dive, PE Hub Blog, BGR, Newser, GeekWire, Eater, The Verge, USA Today, Mashable, Consumerist, New York Magazine, socalTECH.com, CNBC, and CBS New York

Tweets: ✓@dkberman, ✗@chicagophotosho, ✓@krangtnelson, ✓@rmac18, ✓@mcuban, ✓@slade, ✓@anildash, ✓@danprimack, ✓@danprimack, ✓@neilcybart, ✓@caro_milanesi, ✓@reformedbroker, ✓@dkberman, ✓@ipolovets, ✓@benedictevans, ✓@mikeisaac, ✓@business, ✓@toddbishop, ✗@lutherlowe, ✗@levynews, ✓@fmanjoo, ✓@jgreendc, ✗@linamkhan, ✓@cooperasmith, ✓@shiraovide, ✓@davidjoachim, ✓@azalben, ✓@gavinpurcell, ✓@marketwatch, ✓@benedictevans, ✓@edmundlee, ✓@nickbilton, ✓@ericjackson, ✓@nxthompson, ✓@benthompson, ✓@tomgara, ✓@ryan, ✓@benedictevans, ✓@danprimack, ✓@laurengoode, ✓@business, ✓@edmundlee, ✓@selenalarson, ✓@jasonfried, ✓@hunterwalk, ✓@samsifton, ✓@cgasparino, ✓@reformedbroker, ✓@reutersbiz, and ✓@edmundlee

Figure 6.5: Amazon experiment performed on Model A

6.2 Discussion

In this section, we will analyse the results of our experiments. We have already explained the first experiment where we examined different temporal constraints on the dataset. Here we chose to proceed with the dataset where $t = 21600$. We will now try to explain why Model A does better than Base-case, why Model B does better than Model A, why Model C is worse than Model B and why Model B is not much better than Model A.

Model A

The task of Model A was dual. It should verify previous work done within Knowledge Base Acceleration (KBA) and create a base case for Model B and C. To evaluate Model A we had to create a base case for this as well.

Popularity model, the base-case

Model A's base case is based on popularity. We trained a Random Forest (RF) learner on only one feature, namely the number of likes. As we have explained earlier, there is a restriction on how we retrieve information from Twitter that causes the number of likes to be retrieved some time after the tweet was written. However, since popularity is a straightforward and easy way to evaluate the importance, we allow the Base-case in this case to use this feature.

The importance of good features

There is a big difference to the outcome of the popularity model and model A. A model based on coincidence would probably do better. Model A is doing better because it has many more features.

The first papers from the KBA track focused a lot on entity based retrieval methods. But these models made it very bad. It was only when feature engineering was taken seriously that the results were getting well. When the KBA track was summarized in 2012,Frank et al. (2012) stated that:

> the highest scoring systems in KBA 2012 were split between rich feature engineering from the KB versus focusing on machine learning tools, such as SVMs. In the future a combination of these approaches might score even higher.

The most important contribution we brought with us from the literature around KBA was related to feature engineering. We created features based on information about the content of tweets, information about tweets, information about the author of a tweet, story information and article from Knowledge Base (KB) that tweet linked to relationships between the content of a tweet and the contents of a Story and finally temporal features. From the feature analysis, we can see that some features are particularly important for identifying key tweets, such as the position of the tweet, how many tweets are written before it, how many citations an author has had on Techmeme and whether the user has been mentioned in story or is the author.

Model B and C

Model B and C are based on Model A by expanding the dataset with new features based on principles of CF.

To guide us in the development of these new features, we used the steps behind User-based collaborative filtering. Here are "[r]ating predictions are given by first identifying similar or neighbouring users to the target user, and making a recommendation for the target item based on the weighted average of those users." (Aggarwal, 2016). We, therefore, shared the job of developing new features in two: create similarities and create weighted averages from equal users.

In Model B we chose to determine the similarity between users based on the score any user had received on previous tweets. Users are thus judged by how well tweets they have written. In Model C, we determine similarity based on what stories two users have written about and look at the similarities between these stories. Intuitively, Model B seems to be a better approach because two users are only similar if they are also good at writing key tweets. In Model C, it is possible that a user is very similar to a bot because they tweet about the same things. The drawback with the approach in Model B is that the basis for all similarity assessments is based on the model's assessment of a tweet. Can weighted average values from own estimates be better than estimates of themselves? This was a big concern we had. That is why we in Model C chose to define similarity as something outside the model, namely similarity between similar stories.

To create weighted averages, we use Model B confidence scores, the same type of values we use to calculate the similarity between users. In Model C we use different features that belong to similar users. Again, we were worried that Model B was the result of very many averages and that it would paint values with slightly too thick brushes. That is when we got the idea of using the equality values to widen more different features so that the final model could decide which weighted features are important for the classification, as we do in Model C.

From the results, we see that Model B is better than Model C. It was not up to us that it would be like that. We wonder if it may be something that the similarity concept in Model C became too general. In retrospect, one can think of several reasons why two users who write the same articles are not necessarily equal to the quality of each tweet. An exciting attempt to become Future Work is to mix Model B and C so that we make similar values as we do in Model B and then create weighted averages with several different features, as we do in Model C.

Parameters in Model B and C

We see no clear trends when we vary with parameter k in Model B. We get good results at low $k = 10$ and at high $k = 50$. It may seem that the variations are more random than anything else.

In Model C, it does not seem that the parameter k does not make a difference. On the other hand, when parameter l is high, the results improve. In four out of five cases, the highest F_1 values are when l is 100 or ∞.

In both Model B and C, we use parameters to do some filtering of users or stories. An extension of these models could be to use better methods for determining equality, then filtering. Instead of basing similarity to *CosineSimilarity* or *PearsonSimilarity* as many do, you could create your own similarity models based on exercise.

Why are Models B and C much better than Model A

This is a difficult question to answer. We can review our experiments that Models B and C are better than Model A and that Model B is significantly better. When we see how Models A, B and C move quite similar to the curve, and how they are the same features that are among the top 15 most important features, it is expected that the results Do not vary very much. However, why do not the results vary more? It appears that Model A manages to pick up pretty well from the dataset the factors that determine whether a tweet should be quoted. We can look at the original features we use in Model A as direct features. They are about tweet and story directly. Then we can say that the new features from Model B and C are indirect features. Since Model A greatly benefits from the direct features, it will take much time before using indirect features.

Usually, when using CF, it is because you do not have more information. When someone explores Netflix and searches for new movies, the model does not even know what the user thinks about the different movies. Then it must predict values, and it does this based on what the user likes and next-door users like. After the user has seen the movie, she can give it a rating and then the model will be updated with the true values. In our model, we do not have these two phases. The value of a CF estimate is therefore not as high as it would have been if we had no access to any other information.

Limitations

One of the biggest limitations we had was related to data collection. This was a tough and tedious process of requiring data retrieval every day for several months. This reduced our ability to explore the dataset in advance.

Several of the operation had to be converted into matrices and vectors for us to be able to perform different calculations and calculations within a timely manner. Some operations, such as similarity calculations, could quickly take 6 or 9 hours for each variation of the parameters.

In this Chapter we have presented and discussed our results. Model A performed much better than the Base-case model, Model B was significantly better than Model A, while Model C was not able to beat Model B.

7 Conclusion and Future Work

In Section 7.1 we present our conclusion where we conclude that information about the relationships between authors of different tweets helps the model to better identify key tweets. In Section 7.2 we present some idea that we have for future work.

7.1 Conclusion

In this project, we have validated previous work done within Knowledge Base Acceleration (KBA). We have created a new dataset and applied methods and ideas related to feature engineering, classification processes and evaluation of data coming from this subject area. All of these principles are aggregated in Model A. The model performs well and achieves a maximum F_1 score of 0.853. It has been very satisfying to see how these principles stand in a new environment.

In addition, we have developed models that expand state-of-the-art within KBA with features based on principles of Collaborative Filtering (CF). We have worked with two different approaches in Model B and C, both of which perform better than Model A. Model B is significantly better than Model A with a maximum F_1 score of 0.862. We can conclude that information about the relationships between authors of different tweets helps the model to better identify key tweets.

Contributions of the Thesis

The task's main contribution to the KBA is the view that taking into account relationships between authors of received documents can improve a model's ability to identify key documents. There are undoubtedly smarter heads than ours who could take these concepts further and explore the potential behind this contribution.

Also, we have verified that relevant principles of the KBA track work on a brand new dataset. We particularly want to validate the elements of literature related to feature engineering.

Goal Achievements

We would like to review the original Research Questions from Section 1.2 and explain how we have answered these in this project.

Research question 1 *How do we adapt principles from the area of Knowledge Base Acceleration to help identify central tweets from a newly created dataset?*

We have studied papers published through the KBA track and extracted relevant principles for feature engineering and classification methods. Correct feature engineering has proven to be one of the most significant contributions from the literature about KBA, as the classification methods themselves are quite straightforward. We have also spent many resources creating a large set of datasets with rich features. It was important for us to have a large dataset to try the methods so that the results could be validated. Moreover, since the participants in the KBA track also had access to a large dataset.

The result of this work was Model A. It achieved a maximum F_1 score of 0.853, which was a very satisfactory result. We feel we have managed to transfer principles from the KBA track, and especially from research done by Balog et al. (2013).

Research question 2 *How can we use Collaborative Filtering to generate new features?*

We based our work on User-based collaborative filtering and found two main tasks behind this method. The first is to identify the similarity between users. The other is to use the similarity to create weighted contributions from neighbouring users. We have spent much time exploring our dataset to look for potential relationships between users. In Model B we look at the similarity between users in terms of how good they have been to write central tweets in the past. In Model C we look at the similarity of two users based on what kind of stories they write tweets about. When we calculate the weighted contributions from close users, we have used the same currency as we used to calculate similarity in Model B. In Model C, the weighted contributions from raw features belong to similar users.

Although none of the features we have created follows the User-based collaborative filtering method perfectly, the principles behind the method have enabled us to quantify the relationships between users.

Research question 3 *How well do the features inspired by Collaborative Filtering perform compared to other features?*

Model B and Model C perform both better than Model A. Only Model B performs significantly better. None of the new features we have made becomes more important than the most important from Model A. This is natural since Model A contains several features that directly affect determining whether a tweet should be cited or not. For example, almost all cited tweets from Techmeme write about the main article in a story and not one of the other articles that a story links to. This feature is therefore very important. Features from CF are the average of contributions from similar users and draw a picture with thicker brush strokes. But they help to distinguish central tweets from noise.

When we see all the research questions in one, we have achieved our goal of:

Goal *Identify in what ways principles from Collaborative Filtering help improve a classifiers' ability to identify central tweets from a stream of tweets?*

Both Models B and C express ways to use the principles of CF to extend Model A and produce better results.

7.2 Future Work

There are some different aspects that we have not had room to cover. Some of these are:

Combining Models B and C We see that Model B does better than Model C. Nevertheless, we believe the way we predict values in Model C are important because it allows the final classifier to decide which features are most Important in a context of all the other features that it can also take into account. We suggest using Model B's approach to calculating similarities and Model C's approach to predicting estimated values.

Deep learning and Word2vec All the classification models could have been replaced with a deep neural network. This can produce better results as these models have the capabilities to pick up more shades in the data set. All term extraction and content work can be done using Word2Vec that retains each word's context.

Predict new stories Since we have now investigated a model that recognises the pattern behind key tweets, it would be interesting to see if you could expand the model to recognise when new key stories are shared on Twitter.

Bibliography

Rafik Abbes, Karen Pinel-Sauvagnat, Nathalie Hernandez, and Mohand Boughanem. Irit at trec knowledge base acceleration 2013: Cumulative citation recommendation task. In *The Twenty-Second Text REtrieval Conference-TREC 2013*, pages pp–1, 2013.

Charu C Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

Krisztian Balog and Heri Ramampiaro. Cumulative citation recommendation: Classification vs. ranking. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 941–944. ACM, 2013.

Krisztian Balog, Heri Ramampiaro, Naimdjon Takhirov, and Kjetil Nørvåg. Multi-step classification approaches to cumulative citation recommendation. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*, pages 121–128. Centre de hautes études internationales d'informatique documentaire (CID), 2013.

Alejandro Bellogín, Gebrekirstos G Gebremeskel, Jiyin He, Jimmy Lin, Alan Said, Thaer Samar, Arjen P de Vries, and Jeroen BP Vuurens. Cwi and tu delft at trec 2013: Contextual suggestion, federated web search, kba, and web tracks. In *Proceedings of the Text REtrieval Conference (TREC)*. Citeseer, 2013.

Benjamin Biegel, Quinten David Soetens, Willi Hornig, Stephan Diehl, and Serge Demeyer. Comparison of similarity metrics for refactoring detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 53–62. ACM, 2011.

Leo Breiman. Manual on setting up, using, and understanding random forests v3. 1. *Statistics Department University of California Berkeley, CA, USA*, 1, 2002.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. *Sample Selection Bias Correction Theory*, pages 38–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-87987-9. doi: 10.1007/978-3-540-87987-9_8. URL `http://dx.doi.org/10.1007/978-3-540-87987-9_8`.

Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

Laura Dietz and Jeffrey Dalton. Umass at trec 2013 knowledge base acceleration track: Bi-directional entity linking and time-aware evaluation. In *TREC*, 2013.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Kaibo Duan, S Sathiya Keerthi, and Aun Neow Poo. Evaluation of simple performance measures for tuning svm hyperparameters. *Neurocomputing*, 51:41–59, 2003.

Miles Efron, Jana Deisner, Peter Organisciak, Garrick Sherman, and Ana Lucic. The university of illinois' graduate school of library and information science at trec 2012. Technical report, DTIC Document, 2012.

Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

Brian S Everitt. *The analysis of contingency tables*. Chapman and Hall, 1977.

John R Frank, Max Kleiman-Weiner, Daniel A Roberts, Feng Niu, Ce Zhang, Christopher Ré, and Ian Soboroff. Building an entity-centric stream filtering test collection for trec 2012. Technical report, DTIC Document, 2012.

Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

Hui Guo. Soap: Live recommendations through social agents. In *Fifth DELOS Workshop on Filtering and Collaborative Filtering, Budapest*, 1997.

John A Hartigan and JA Hartigan. *Clustering algorithms*, volume 209. Wiley New York, 1975.

Stefanie Haustein, Timothy D Bowman, Kim Holmberg, Andrew Tsou, Cassidy R Sugimoto, and Vincent Larivière. Tweets as impact indicators: Examining the implications of automated "bot" accounts on twitter. *Journal of the Association for Information Science and Technology*, 67(1):232–238, 2016.

Berberich K Hoffart J, Suchanek and Weikum G. *AGO2: a spatially and temporally enhanced knowledge base from Wikipedia.* Elsevier, 2013.

Heng Ji and Ralph Grishman. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1148–1158. Association for Computational Linguistics, 2011.

Brian Kjersten and Paul McNamee. The hltcoe approach to the trec 2012 kba track. Technical report, DTIC Document, 2012.

Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

Kobi Levi and Yair Weiss. Learning object detection from a small number of examples: the importance of good features. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society*

Conference on, volume 2, pages II–II. IEEE, 2004.

Yan Li, Zhaozhao Wang, Baojin Yu, Yong Zhang, Ruiyang Luo, Weiran Xu, Guang Chen, and Jun Guo. Pris at trec2012 kba track. Technical report, DTIC Document, 2012.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

Xitong Liu and Hui Fang. Entity profile based approach in automatic knowledge finding. Technical report, DTIC Document, 2012.

John Mandel. *The statistical analysis of experimental data*. Courier Corporation, 2012.

Christopher D Manning, Hinrich Schütze, et al. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Tom M Mitchell et al. Machine learning. wcb, 1997.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

Estelle M Phillips and Derek S Pugh. How to get a ph. *D.: a handbook for students and their supervisors. Buckingham, UK: Open University Press,*, 1994.

Vijay Raghavan, Peter Bollmann, and Gwang S Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems (TOIS)*, 7(3):205–229, 1989.

Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Garrick Sherman, Miles Efron, and Craig Willis. The university of illinois' graduate school of library and information science at trec 2014. Technical report, DTIC Document, 2014.

Valentin I Spitkovsky and Angel X Chang. A cross-lingual dictionary for english wikipedia concepts. In *LREC*, pages 3168–3175, 2012.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World*

Wide Web, pages 697–706. ACM, 2007.

Jingang Wang, Dandan Song, Lejian Liao, and Chin-Yew Lin. Bit and msra at trec kba ccr track 2013. In *TREC*, 2013.

Chao Michael Zhang and Vern Paxson. Detecting and analyzing automated activity on twitter. In *International Conference on Passive and Active Network Measurement*, pages 102–111. Springer, 2011.

Appendices

1 Stop words

a, about, above, after, again, against, all, am, an, and, any, are, aren't, as, at, be, because, been, before, being, below, between, both, but, by, can't, cannot, could, couldn't, did, didn't, do, does, doesn't, doing, don't, down, during, each, few, for, from, further, had, hadn't, has, hasn't, have, haven't, having, he, he'd, he'll, he's, her, here, here's, hers, herself, him, himself, his, how, how's, i, i'd, i'll, i'm, i've, if, in, into, is, isn't, it, it's, its, itself, let's, me, more, most, mustn't, my, myself, no, nor, not, of, off, on, once, only, or, other, ought, our, ours, ourselves, out, over, own, same, shan't, she, she'd, she'll, she's, should, shouldn't, so, some, such, than, that, that's, the, their, theirs, them, themselves, then, there, there's, these, they, they'd, they'll, they're, they've, this, those, through, to, too, under, until, up, very, was, wasn't, we, we'd, we'll, we're, we've, were, weren't, what, what's, when, when's, where, where's, which, while, who, who's, whom, why, why's, with, won't, would, wouldn't, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves

2 KNIME Workflows

Below are the KNIME workflows that we created in order to run our dataset through part of the Models. Both Model B and C do intermediate work on the dataset in order to complete its classification model.

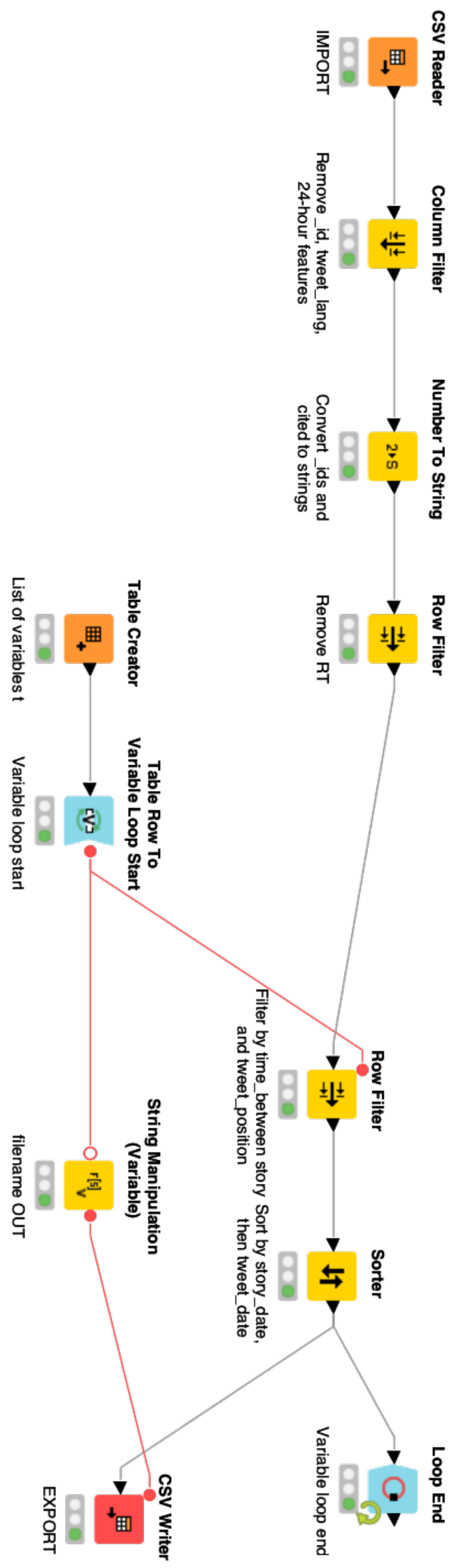

CSV Reader
IMPORT
Column Filter
Remove _id, tweet_lang, 24-hour features
Number To String
Convert _ids and cited to strings
Row Filter
Remove RT
Table Creator
List of variables t
Table Row To Variable Loop Start
Variable loop start
Row Filter
Filter by time_between story and tweet_position
Sorter
Sort by story_date, then tweet_date
String Manipulation (Variable)
filename OUT
Loop End
Variable loop end
CSV Writer
EXPORT

Figure 1: Preprocessing KNIME workflow for all models

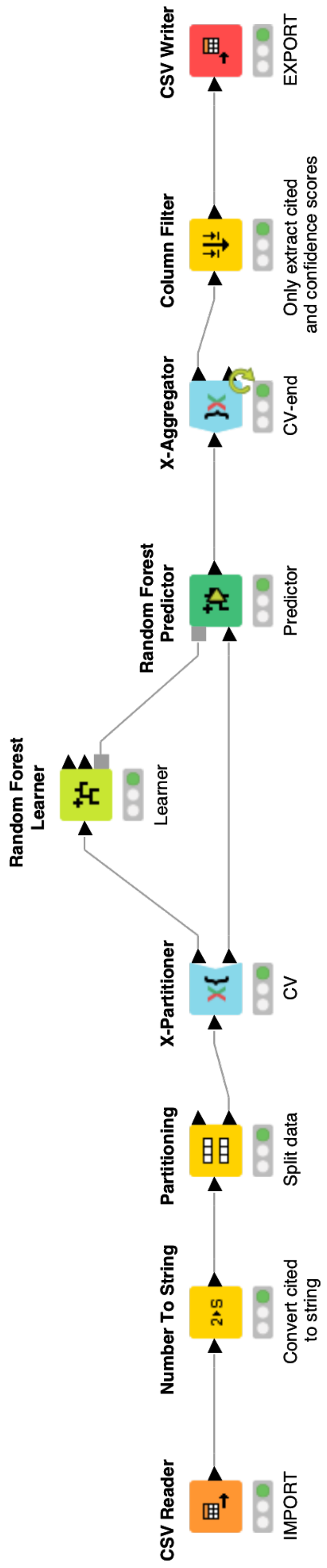


Figure 2: KNIME workflow for Base-case

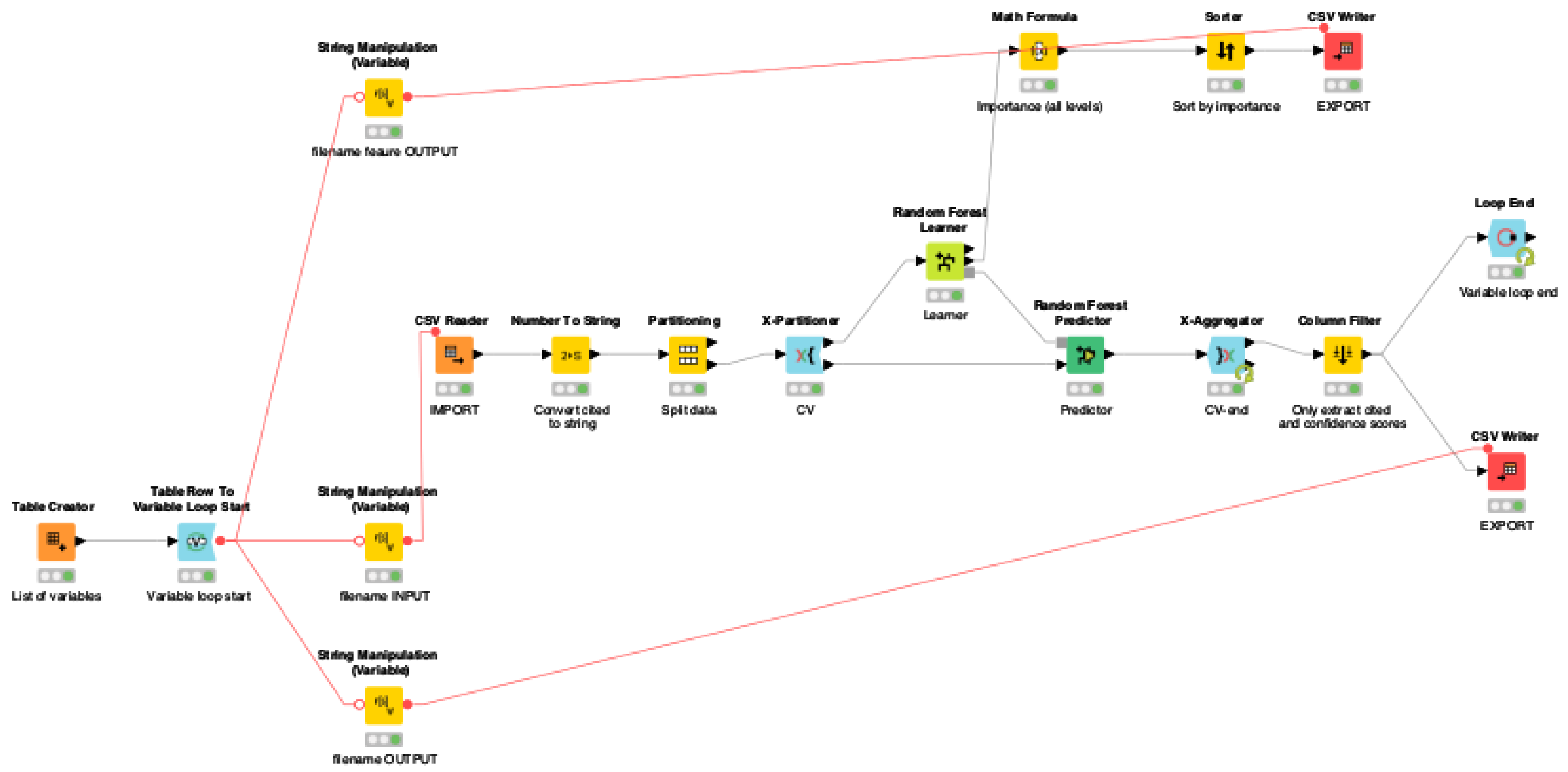


Figure 3: Time constraints KNIME workflow for Model A

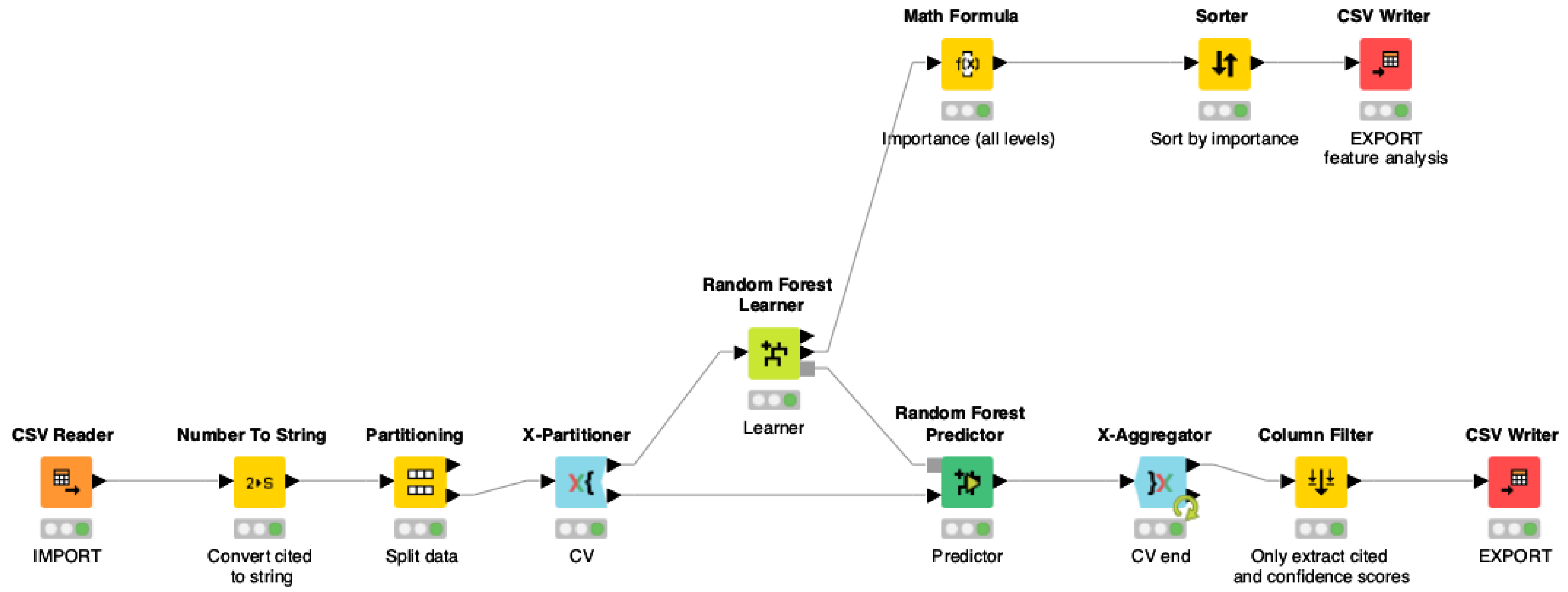


Figure 4: KNIME workflow for Model A

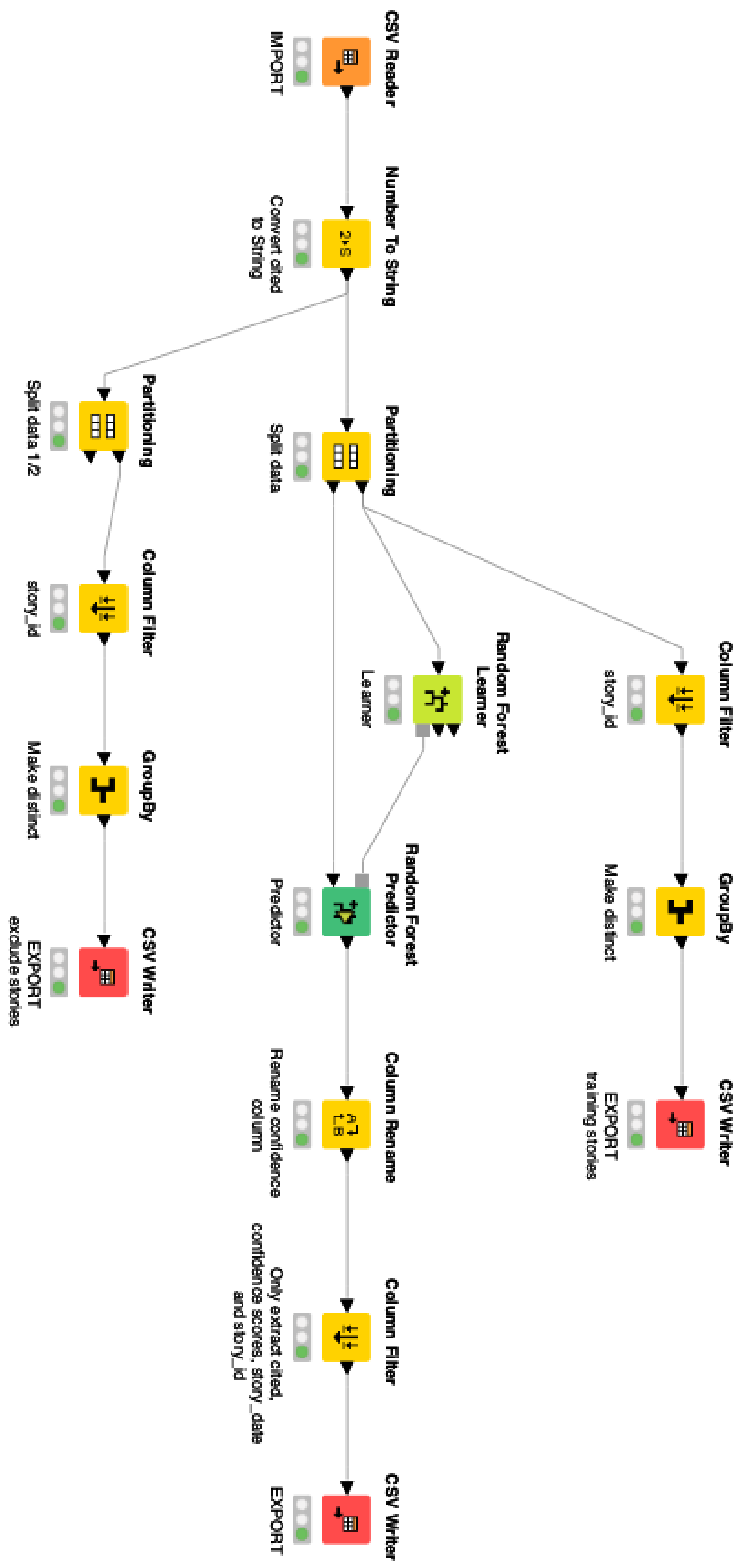


Figure 5: First step of KNIME workflow for Model B

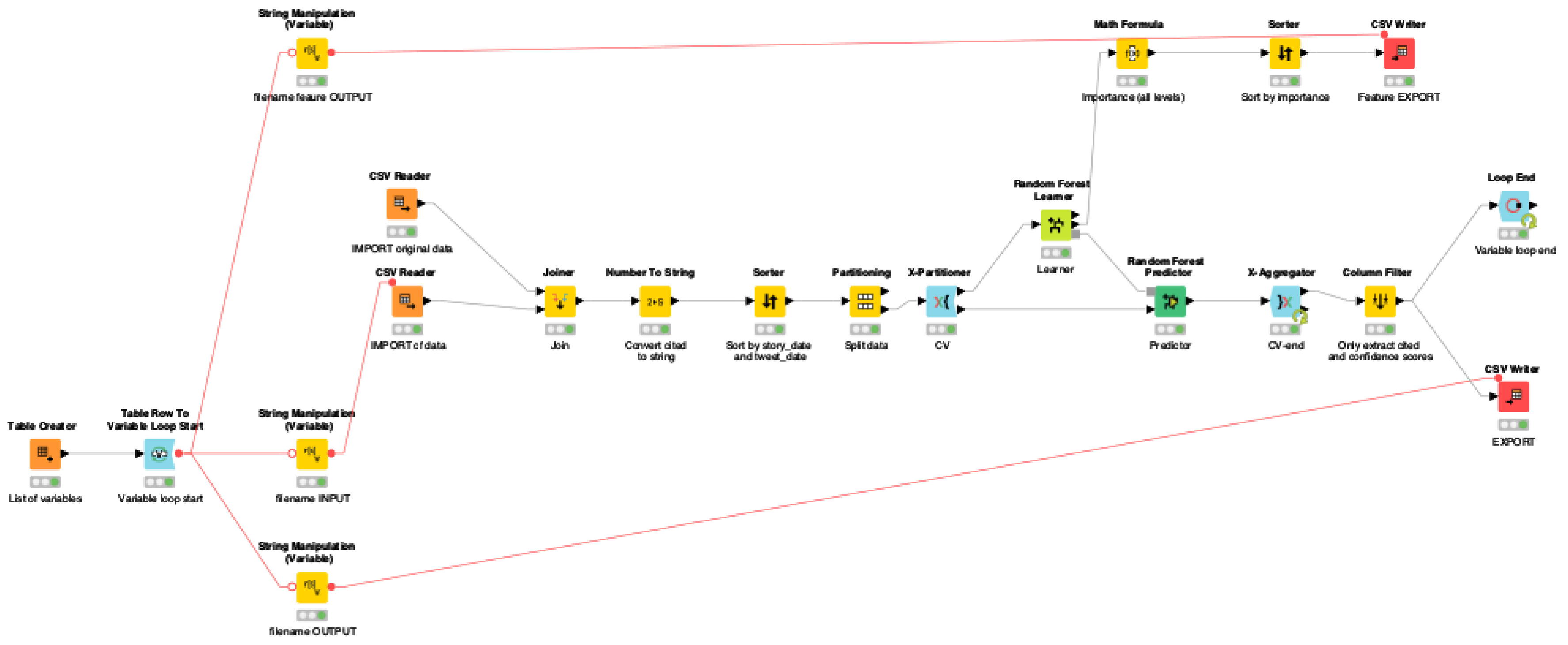


Figure 6: Last step of KNIME workflow for Model B

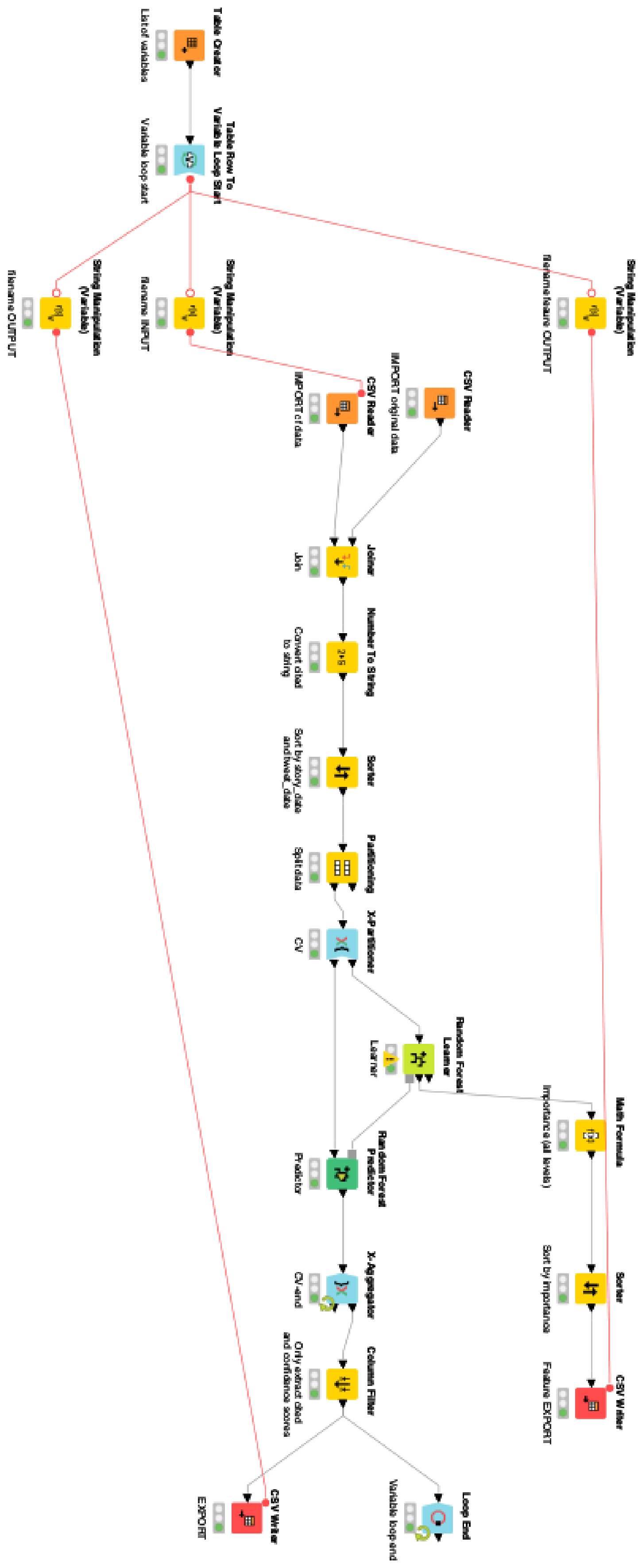


Figure 7: KNIME workflow for Model C