Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2014 Open access
Writer: Chris Håland	…… (Writer's signature)
Faculty supervisor: Krisztian Balog	
Thesis title: Tweet-based Event Summarization	
Credits (ECTS): 30	
Key words: Twitter, Data and Information Management, Machine Learning, Naïve Bayes, Random Forest, C4.5	Pages: 45 + enclosure: 1 Stavanger, June 12th, 2014 Date/year

Front page for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

Tweet-based Event Summarization

Chris Håland

June 2014

Department of Electrical Engineering and Computer Science

University of Stavanger

Supervisor: Krisztian Balog

Task Description

Structuring and analyzing data published to social media services has become an appealing field of research in later time, as social media has grown to contain massive amounts of data. This project aims to make a live event summarization application based on data published to social media, more specifically, Twitter.

Preface

This is the thesis of my Master's degree in Computer Science at the University of Stavanger, which has elapsed over a period of 20 weeks during the spring of 2014.

The thesis is in the field of Data and Information Management and presents a method of summarizing events, performing event detection on Twitter data.

I would like to thank my supervisor, Krisztian Balog. Your feedback and guidance during the development of the thesis has been invaluable for the result.

Stavanger, 2014-06-12

Chris Håland

Summary

Social media has become an ever-growing source of information over the last years. Facebook, Twitter, Instagram and other types of social media services have all grown to contain large amounts of data, written by anyone from everyday users to companies and institutions.

In this thesis, we explore the possibility of creating an event summarization system, which summarizes events based on microblog posts published to Twitter. We design a website interface for displaying event-related data and store all tweets in a scalable solution using Hbase. To determine a tweet's relevance to an event we introduce a two-step filtering technique, where we use simple regular expression matching and apply a machine learning technique to predict a tweet's relevance, based on feedback on previously accepted data.

We provide a viable solution for creating a tweet-based event summarization system. The system delivers a scalable and responsive end user experience by storing all event-related data in a non-relational database, namely the row-key store, Hbase. By using machine learning algorithms to determine if a tweet is event-relevant, we effectively reduce the number of false-positive tweets passing the filter. We evaluate three different classifiers, Random Forest, Naive Bayes and C4.5, and measure their precision over time as the system receives feedback. We also test three different model training strategies, using a single model strategy, where we creating a single model for all topics, a split model strategy where we use two models, one for ambiguous topics and one for unambiguous topics and an individual model strategy, creating a model per topic. Our results show that using a single training model with Random Forest perform best.

Contents

Task Description i
Preface ii
Summary iii

1 Introduction **1**

2 Related Work **3**

3 Problem Statement and Overview **6**

3.1 Tweets 7
3.1.1 The Anatomy of a Tweet 7
3.1.2 Doing a Live Data Sample 9
3.2 Filtering Microblog Documents 9
3.2.1 Filtering Event Related Tweets 9
3.2.2 Creating a Topic Model 10
3.3 Choosing a Scalable and Live Storage System 11
3.4 Graphical Representation for Topic-Document Relations 11

4 Storage System **13**

4.1 Non-relational Databases 13
4.2 Choosing a Scalable and Live Storage System 14
4.3 Structuring a Data Schema for Hbase 15
4.4 Communicating with the Storage System 16
4.4.1 Comparing Response Time of Different Thrift Querying Methods 17

5 Filtering System **19**

5.1 The First Filtering Step 19

5.2 The Second Filtering Step 22

5.2.1 Features 26

5.3 Model Training Strategies 27

6 Experimental Evaluation **29**

6.1 Experimental Setup 29

6.1.1 Topics Used in the Live System Test 30

6.1.2 Registering Positive and Negative Feedback 30

6.1.3 Evaluation Measures 31

6.2 Live Testing Statistics 32

6.3 Classifier Evaluation 33

6.3.1 Evaluating Classifiers Using Single Training Model 34

6.3.2 Evaluating Classifiers Using Split Training Models 35

6.3.3 Evaluating Classifiers Using Individual Training Models 37

6.3.4 Comparing Macro-averaged Precision of the Model Training Strategies . . . 38

6.4 Proposing Topic Trackers Based on Feedback 39

7 Conclusion **41**

7.1 Future Work 42

References **43**

A Attachments **46**

Chapter 1

Introduction

Social media and microblog services have grown over the later years to become large sources of information. Facebook, Twitter and other social media networks generate large amounts of data on a day-by-day basis, as users all over the world publish posts to these services. Posts may contain a user's political, religious or social opinion and can be described as being event related. An event may or may not contain other events that are categorized as hierarchically being a lower tier of the given event. E.g., if we consider a given TV show to be an event, than specific seasons and episodes of that show are lower tier events. For the remainder of this thesis, we refer to events as topics and refer to the collection of our topics as an event.

We aim to create an event summarization system, based on data from Twitter. The system should be able to identify topic-related tweets and categorize them according to topics. We focus on applying machine learning techniques to improve upon the identification accuracy, using user feedback of previously identified tweets. The feedback tells the system whether a tweet belongs in the topic it's been placed. In addition, we need to choose a suitable method of storing the data, where scalability and liveness are important factors. Lastly, we develop a user interface that displays the tweets according to topics, as well as allowing users to give feedback to the system. Our focus is to evaluate the effectiveness of the proposed machine learning techniques. We compare different machine learning algorithms and ways of structuring machine learning models to see which performs best for our system.

Microblog posts, such as tweets, propose analytical challenges, seeing their size is minimal. The posts are often only one sentence and colloquially formed and may not always state their

full meaning or intention. Topics may have an ambiguous title, meaning the topics title is often mentioned when referring to other topics or settings. As an example, the TV show “Suits” is an ambiguous topic, seeing the word “suits” may refer to a suit, bathing suit or a verbal expression like “that suits you”. Another example is the TV show “Vikings”, where “Vikings” amongst other things, may also refer to the Minnesota Vikings, an American football team playing in the NFL[1].

We continue our work from [1] where we sampled a static test data set and compared the performance of the machine learning classifiers C4.5, Random Forest and Naive Bayes. This thesis expand the work done in the paper by creating a full implementation of the proposed system and running a live system test. In addition, we choose a suitable database for storing large amounts of data and evaluate classifier performances from the live system. We use 9 popular TV shows as topics when running the live system test.

In Chapter 2 we explore work related to the thesis. In Chapter 3 we give a brief overview of the system and examine challenges we face in creating such a system. In Chapter 4 we choose a suitable storage structure for storing topics, documents and relations between documents and topics. In addition, we evaluate different communication protocols for the chosen database and compare query response times. In Chapter 5 we define the proposed filtering system for establishing if a document is related to a given topic. In Chapter 6 we explain the experimental setup for the live system test, list statistics as well as evaluate the classifier performances over the course of the live system test period. Lastly, we draw a conclusion and propose future work in Chapter 7.

[1] NFL - National Football League

Chapter 2

Related Work

Over the latest years, social media and microblog services have grown to contain large sources of information. This information is not easily accessible and structured for the everyday user, which has caused a growing research interest in the field of data and information management. Twitter has been a fertile area for research in [2], [3], [4], [5], [6] and [7].

In [2] Albakour, Macdonald and Ounis extends an effective traditional news filtering technique to approach the problem of filtering tweets in real-time. The filtering technique is based on Rocchio's relevance feedback algorithm, which both builds and dynamically updates the relevance of the documents being processed by the system. One of the issues that arise when dealing with document (tweet) filtering is sparsity. A document is limited to 140 characters and it may not contain sufficient enrichment to be reliably filtered, which is much less of a problem in, e.g., newspaper articles. They therefor propose adding a query expansion (QE) to enrich the filtering process. In our work, we do not embed QE, but rather use the idea as a baseline and define words and phrases, or trackers, which we expect to be present in topic-related documents.

In [3] Benson, Haghighi and Barzilay propose a method of discovering event records from social media feeds. The discovery method uses an event record website as baseline for comparison against the social media service, Twitter. Extracting information from social media services, such as Twitter, bids more challenging than extracting information from formal media (e.g., newspapers). The text is often one sentence and colloquially formed and may not express its full meaning or intention. Their work is mostly based on [8], a paper by Mann and Yarowsky. They use a method of fusing information extracted from multiple documents (here: tweets)

by individually extracting information from each document and then merging them together. However, they cluster the output and label the documents simultaneously, not serially.

In [4] Pennacchiotti and Popescu propose a method of determining the ethnicity and political orientation of a Twitter user by applying machine learning. By focusing on information such as user behavior, network structure and the linguistic content of the user's tweets, they build a classification method. They test their method by trying to detect the political affiliation and ethnicity of a user, as well as whether the user is a Starbucks fan or not. The paper propose using regular expressions to find data matches they find relevant. We also implement a similar method of approach when determining if a document is topic-relevant.

In [5] Meij, Weerkamp and de Rijke explore the possibilities of adding semantics to microblog posts. They propose a method for linking tweets to Wikipedia articles by matching n-grams from a tweet to concepts matching articles from Wikipedia. Their method creates ranked lists of concepts for tweets, where a higher ranked concept, is a concept more relevant for the tweet. Furthermore, the ranked list is composed by analyzing the likeness of a tweets n-grams and concepts found as articles from Wikipedia. The method presented is based on high recall concept rankings and high precision concepts and aims for effective semantic linking of microblog posts, not high performance or efficiency.

In [6] Mahmood, Iqbal, Amin, Lohanna and Mustafa try to predict the outcome of the 2013 Pakistan Election, by pre-processing tweets of relevant Twitter users and constructing predictive models for each of the representative political parties. They classify tweets as either being in favor of or against a political party and create models for each of the political parties, which they then use to predict the election winner based on Twitter data from the four last days before the election. The results presented in the paper show that the political party PTI[1] was predicted to win the election, while the actual winner was PMLN[2]. Even so, the conclusion states that analysing Twitter data to predict the outcome of an election do have some value. PTI did not win the election, but won a province and in several other constituencies. Seeing these results, the following is stated "[...] Twitter can have some type of a positive influence in the election results, but it cannot be considered completely representative of the voting population.". As previously

[1] PTI - Pakistan Tehreek-e-Insaaf
[2] PMLN - Pakistan Muslim League Nawaz

mentioned, they define documents to be either in favor of or against a given political party. In similarity to our task, a document can either be related to a topic or not. We try to predict if a document relation to a topic is either positive or negative, meaning if it is topic-relevant or not.

In [7] Efron presents some challenges with extracting information from microblog services. We presented some of these challenges in Section IV in [1] and we discuss them further in Chapter 3, along with the information retrieval challenges of microblog services expressed in [2] and [3].

Chapter 3

Problem Statement and Overview

Before we begin exploring the proposed system, we should briefly state the challenges we face. We need to address the challenges of choosing an appropriate storage system for storing large amounts of data as well as a filtering system for determining a document's relevance to our predefined topics within our event. In addition, we give a quick overview of how we do a live data sample and the design of the system's website interface.

The flow chart we see in Figure 3.1 is a simplistic overview of the system layout. In general, we sample documents from Twitter, use our filtering system to determine which documents are stored to the storage system (documents determined as related to our predetermined topics) and use the data stored to the storage system to display the documents in topic categories in the website interface.

In order to make it easier for the reader to understand the meaning of some of the terms we use in this report, we introduce our terminology in Table 3.1. We generalize some microblog service terms as well as machine learning terms.

The sections in this chapter covers as follows: In Section 3.1 we look at the anatomy of a tweet and how to sample data from Twitter. In Section 3.2 we list the challenges of filtering

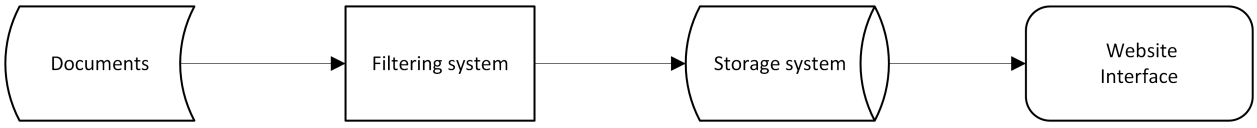

Figure 3.1: System layout

Term	Description
topic	an event or part of an event we want to gather information about.
document	a microblog post or a tweet
tracker	a word or phrase, defined for a topic, which is used to establish whether there is a topic-document relation
user	an individual responsible for implementing the system
end user	an individual using the implemented system
publisher	an individual that has written and published a document
classifier	a machine learning algorithm
feedback	labeled documents, based on end users registering positive or negative matches

Table 3.1: Terms used in this paper and their descriptions

microblog posts and establish a topic model. In Section 3.3 we look at the challenges of choosing a suitable storage system for our system and in Section 3.4 we describe the website interface, which displays the tweets we have related to our event.

3.1 Tweets

In this section, we give an overview of how a tweet is structured and how we do a live data sample from Twitter. The different overviews are given in Subsection 3.1.1 and 3.1.2 respectively.

3.1.1 The Anatomy of a Tweet

The microblog service Twitter limits publishers to writing messages of a maximum of 140 characters. These messages are referred to as ”tweets”. A tweet’s content can be categorized into four types of categories: text, hashtag, mention and URL-links. Text and URL-links in a tweet are common terms, while the meaning of a hashtag and a mention may not be common knowledge.

A hashtag is a sequence of characters, starting with a ”#”, followed by a word or phrase containing non-special characters. Hashtags typically indicate an event, a topic or some sort

Username	Tweet
@Shelunaita	Just watched Marvel's Agents of S.H.I.E.L.D. S01E15 Yes Men
@mplikespotatoes	RT @cheerandjesuscr : Doctor Who is so awesome!!! #doctorwho
@James_mcuk	Watching: Blu-ray: Doctor Who - 7x02: 'Dinosaurs on a Spaceship' #DoctorWho #tvtag http://t.co/zk73zZ0opJ
@AshleySarrasin	I took Zimbio's 'Big Bang Theory' quiz and I'm Amy! Who are you? http://t.co/XRBAOp0pfT
@RaksGeek	#Firefly #Serenity fans: What's your favorite episode & why? #Browncoats

Table 3.2: Tweet examples

of happening. This gives a publisher the ability to add semantic meaning to his or her document, by adding a hashtag suitable for the topic related to the document. Some typical hashtags are "#2tdf" and "#doctorwho", where "#2tdf" is a hashtag used when commenting on Tour de France to the Norwegian TV channel, TV2, and "#doctorwho" comments on BBC's TV show, Doctor Who.

A mention is a sequence of characters, starting with a "@", followed by a set of non-special characters, which correspond with a Twitter user's username. Mentions indicate a Twitter user in a tweet, by referring to his or her Twitter username. To give an example of a mention, "AgentsofSHIELD" is the official Twitter account name of the ABC TV show "Marvel's Agents of S.H.I.E.L.D.". A publisher would use the mention "@AgentsofSHIELD" in a tweet to be referring or mentioning the show.

In addition to a tweet's structure, we also explain the term "re-tweet": a re-tweet is a republished tweet by a non-original publisher of an original tweet. Originally, a re-tweet started with "RT @'username of original publisher':", followed by the original tweet's content. Twitter later refined re-tweeting, so it does not include the previously mentioned text, leaving more room for the tweet's original content itself within the limit of 140 characters.

Tweet examples can be seen in Table 3.2. The examples cover what we have explained in this section.

3.1.2 Doing a Live Data Sample

In Section IV in [1] we described how we got a static data sample of 500,000 tweets, which we used in the experimental setup of that report. As we now proceed a step further, we do a live system test in this report, processing documents as they arrive to the system.

In order to sample documents from Twitter we use Twitter4J[1], which is an unofficial Java library for connecting to Twitter through the Twitter API[2]. The Twitter4J library lets us choose between doing a $sample$ or a $filter$ of the Twitter stream. A $sample$, as the name suggests, returns a selection of newly posted tweets while the connection is active, while a $filter$ lets us specify the selection by setting certain parameters. We can specify language preferences for tweets, the location of a tweet and words or phrases (known as trackers) we want the tweets to contain. Preliminary studies show that $filter$ returns more relevant tweets than $sample$. It seems that $sample$ does not return all tweets being published to Twitter, meaning a $filter$ return more tweets relevant to our event, given we define appropriate trackers.

3.2 Filtering Microblog Documents

In this section, we look at the challenges of filtering microblog documents. Subsection 3.2.1 covers the challenges we need to take in to consideration when creating a filtering system. Subsection 3.2.2 explains how we structure topics defined for our event.

3.2.1 Filtering Event Related Tweets

Extracting information from microblog services, such as Twitter prove more challenging than extracting information from formal media, such as newspapers. Benson, Haghighi and Barzilay states some of these challenges clearly in [3] and we have also covered them in [1].

Documents published to microblog services are small texts and often only one sentence. The documents may also often be colloquially formed and not express their full meaning or intent. This means the texts we aim to analyze and determine a semantic meaning of, prove more challenging than analyzing a formal media text, such as newspaper articles. Formal media has

[1] Twitter4J - An unofficial Java library for the Twitter API http://twitter4j.org/en/index.html
[2] Twitter API https://dev.twitter.com/docs

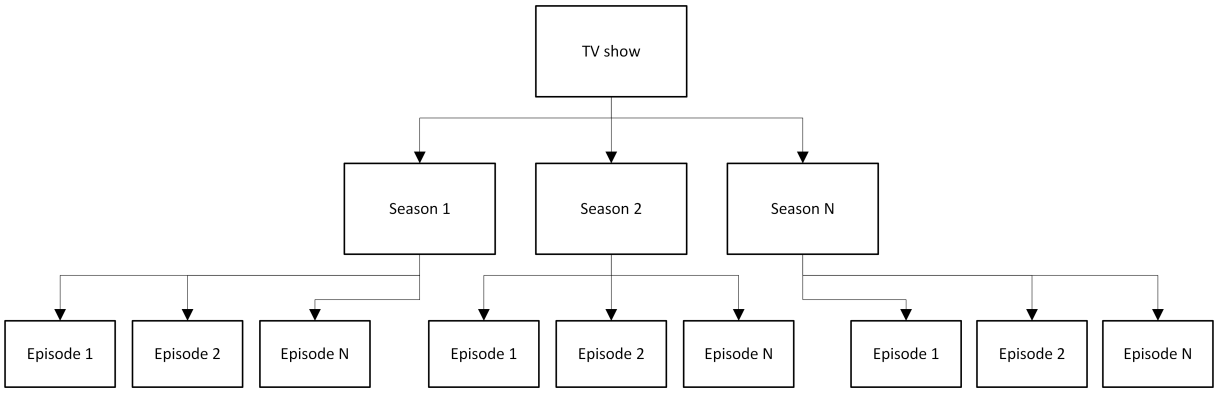

Figure 3.2: Hierarchical topic structure

as prerequisite of maintaining a high linguistic standard, being both structurally and grammatically intact. Microblog documents on the other hand may contain shortened expressions for words or phrases as well as intentionally misspelled words or slang. Each publisher structure his or her document in his or her own manner, meaning documents in general are subjective. The subjective writing may also apply to the spelling of a single word, where an individual may have a unique way of spelling that specific word, deriving from cultural, social or locational differences. These challenges are important to address and have an impact of how the filtering system work. We continue to address these challenges in Chapter 5.

3.2.2 Creating a Topic Model

As publishers publish documents to microblog services, we aim to filter and relate them to an event. Therefore, we need to define the event we use for the system. We represent the event as a tree structure, where topics are added to the tree structure if they are expected to be a part of the event. All topics added to the topic tree should then be child topics of the event or of any of the topics already added to the tree. E.g. if our event is football, our root topics for the tree could be the individual football leagues. The child topics of a football league could again be the different football teams competing in that league. We could also define a third level of topics, which could be the individual players playing for a team. If we apply this structure to our event of TV shows, we get a structure as shown in Figure 3.2. We consider each TV show to be a root topic, having seasons as its child topics. Each season's child topics are also, naturally, episodes within each season.

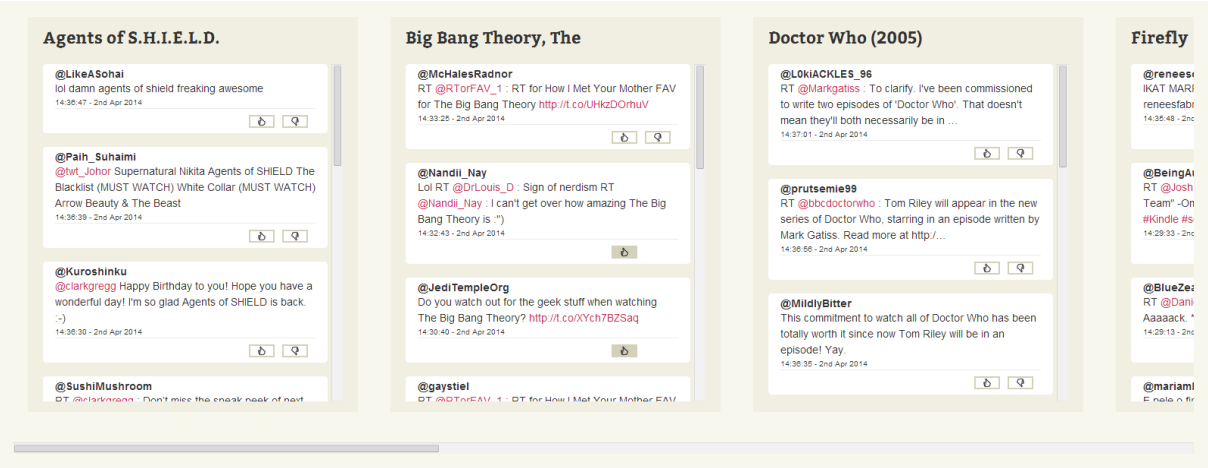

Figure 3.3: Root topics as shown on the graphical interface

3.3 Choosing a Scalable and Live Storage System

Twitter is an ever-growing source of information and has become one of the biggest microblog services to date. With 9000 tweets being published every second and over 500 million active registered users[3] we need to assure that our system is able to handle large amounts of data.

The system need a storage system capable of storing large amounts of data, while staying scalable, being able to return a query result with low latency independent of the number of data entries stored to the system. The storage system should also have capabilities such as liveness and fault-tolerance, making sure our data stays correct and available at any time. Choosing a storage system with high scalability and liveness is important for the end user's ability to give feedback regarding the topic-document relations. The relations are created by the filtering process described in Chapter 5. We choose a suitable storage system in Chapter 4.

3.4 Graphical Representation for Topic-Document Relations

To be able to display the topic-document relations our system establishes, we develop a website interface displaying the topics in a structured manner. Figure 3.3 displays the main page of the website interface. Every TV show is displayed as a single column, with all shows horizontally stacked and sorted by the show's name. By clicking the title of a show, an overlay window dis-

[3]Twitter Statistics - Statistic Brain. 2012 Statistic Brain Research Institute, published as Statistic Brain, 5.7.2013. http://www.statisticbrain.com/twitter-statistics

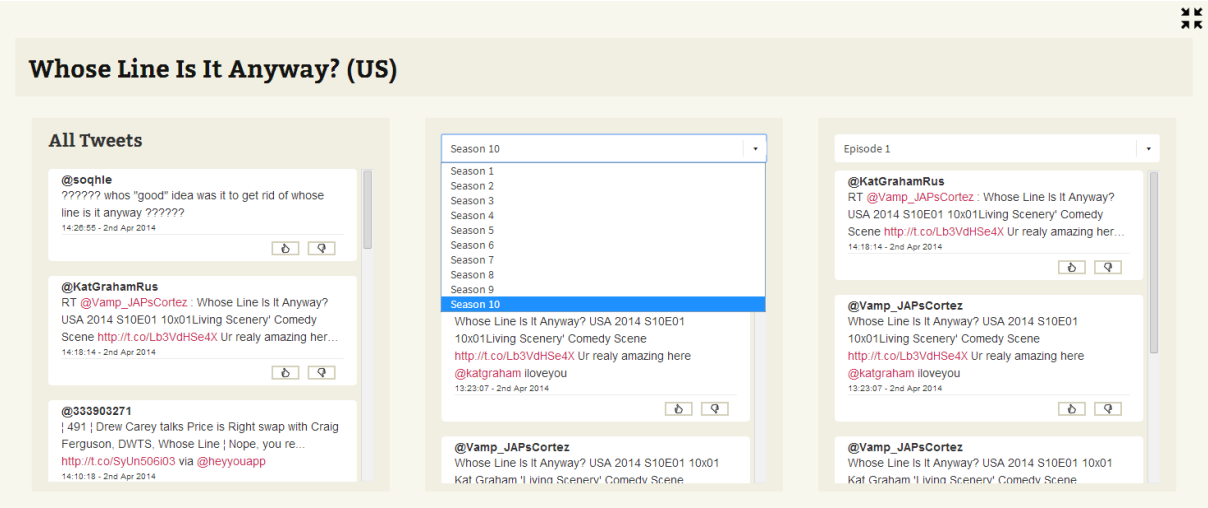

Figure 3.4: Root topic and its child topics as shown on the graphical interface

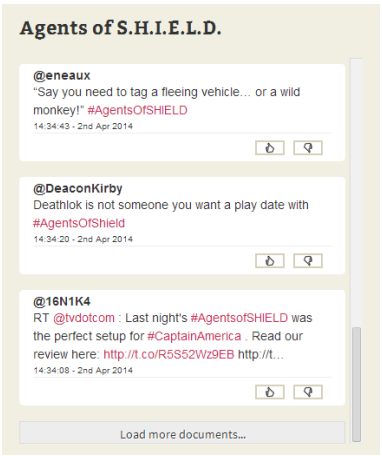

Figure 3.5: Topic column with capability of loading more document-topic relations

plays further information for that given topic, including child topics of the selected root topic. In Figure 3.4 the topic overlay is displayed. The column called "All Tweets" shows documents related to the TV show, or root topic. The following columns each show a lower tree level, with all tree level nodes listed in a selectable list. E.g., the second column contains a season listing of all seasons for the given show and the third column contains all episodes within the selected season of the show.

All columns that show document-topic relations load 10 documents at a time and older documents can be added to the end of the column manually, if the end user desires. This function can be seen in Figure 3.5.

Chapter 4

Storage System

Choosing a proper storage system for storing our data is important. We have already established that large amounts of data pass through our system, and that we need a scalable storage system. A traditional relational database, such as MySQL, does not perform as well with large amounts of data entries as other types of databases. Relational databases have existed for a long time, but now in the age of big data, another type of data storage emerges, non-relational databases. We keep non-expanding data structures in a relational database, which in this case is the topic model. Data from Twitter is stored in a non-relational database, due to its ability to scale.

In Section 4.1 we define the term ¨non-relational database¨. In Section 4.2, we select a non-relational database, which we use as our storage system. In Section 4.3, we describe the general data structure used to store document-related data. Lastly, in Section 4.4, we explore the challenges of choosing a proper way of communicating with the storage system, keeping query response time in mind.

4.1 Non-relational Databases

A non-relational database, often also referred to by the term NoSQL, is a database structure that does not rely on relations between tables. It is defined as being non-relational, distributed and scalable and being eventually consistent, building on the BASE[1] principle. Relational databases

[1] BASE - Basically Available Soft-state Eventually consistent

are not built on BASE, but rather ACID[2]. The history and development of different database structures is explained in more depth in [9].

The data structures graph, tree and key-value occur frequently in non-relational databases. Some examples of non-relational databases are Cassandra, Bigtable, Hbase and Amazon SimpleDB. These are all eventually consistent, distributed and structured as key-value, with a row-column topology.

4.2 Choosing a Scalable and Live Storage System

First, we need to address what structure type we should choose for our non-relational database. Graphs and trees are not natural choices, seeing we do not have a multitude of relations between different data object entries, or a hierarchical structure of the documents. We need a plain storage system for the documents and a key-value structure suits this purpose. We consider some of the open source key-value non-relational databases, namely, Cassandra and Hbase. Both Cassandra and Hbase are a part of and maintained by Apache. They are both key-value stores, but their implementation design differs. Even so, they both rely on Bigtable's key-value structure. Bigtable[10] introduces the concept of storing data as row data, or key, and each data entry's value can contain multiple data points, known as column qualifiers. Column families group qualifiers by a common term, which makes for easier reading and higher semantic meaning.

Cassandra[11] uses the data structure as described for Bigtable and implements a similar gossip technique as Amazon's Dynamo[12]. The Cassandra implementation, as described in [11] shows its API being minimalistic, with three main functions: insert, get and delete. In all of these cases, we need to know the specific row key to be able to extract data from the database. Our use-case requires the ability to search for data entries stored in the database, seeing the website interface is not able to keep record of the row keys of the data stored in the database. We need a method of extracting the latest data entries, which contain our specified topic, without knowing the exact row key.

Hbase[9] is, as previously stated, based on Bigtable's key-value topology. It allows for the same API functions as Cassandra, but in addition it allows for filtering and searching for specific

[2]ACID - Atomicity Consistency Isolation Durability

row key	columns
document_id	document:text
	document:created_at
	document:retweet_count
	document:favorite_count
	publisher:id
	publisher:username

Table 4.1: Document table for Hbase

data stored in the database. As a standalone installation, Hbase uses Zookeeper[13] to distribute and store the data, while a distributed installation uses Hadoop's storage system, HDFS[14]. This solution allows for a scalable and live storage system and storing our data in a suitable schema. In addition, Hbase has the ability of searching and filtering data stored in its database. With the ability of filtering and searching for data and its scalability and liveness, Hbase fits our requirements for storing Twitter data and we choose it as our non-relational database.

4.3 Structuring a Data Schema for Hbase

Now that we have chosen Hbase as our storage system, we need to create a data schema for storing data. Hbase´s data entries contain a row key, column families and column qualifiers. All tables are alphabetically sorted, meaning all row keys stored to the table are sorted from a to z. The actual sorting is a byte comparison of less greater or equal to a neighboring row key.

The table structure for the "document" table can be seen in Table 4.1. The row key is the document´s unique id and its column families: "document" and "publisher". The qualifiers for the column family document contain the documents date of creation, number of retweets and the number of times the document has been marked as a favorite.

The table structure for the "publisher" table can be seen in Table 4.2. The row key for this table is the publisher's unique id and we only define one column family: "publisher". The qualifiers defined are the publisher's name, the publisher's username, the URL to the publisher's Twitter page, the number of documents published by the publisher and the number of people subscribed to the publisher's account.

The table structure for the "relation" table can be seen in Table 4.3. The row key is the topic id, followed by a delimiter, followed by the maximum long value subtracted by the document

row key	columns
publisher_id	publisher:name
	publisher:username
	publisher:url
	publisher:tweet_count
	publisher:followers_count

Table 4.2: Publisher table for Hbase

row key	columns
topic_id::(Long.MAX-document_id)	relation:topic_id
	relation:document_id
	feedback:has_feedback
	feedback:feedback_is_positive

Table 4.3: Relation table for Hbase

id. The latter part of the row key ensures so that newer relations sort to the top of relations with equal topic ids. As publishers, publish documents to Twitter, the document id increments. Using the document id itself results in newer documents, with a higher document id, sorting after older documents with the same topic id in the Hbase table. We define two column families for this table: "relation" and "feedback". The qualifiers defined for the relation family are the topic and document ids, while for the feedback family we define two qualifiers as boolean qualifiers, being defined as whether the relation has gotten feedback by an end user and if the feedback determines the relation as a positive or negative match.

4.4 Communicating with the Storage System

There is a number of different ways of communicating with Hbase, either using the integrated Hbase shell, Hbase Java API, REST[15] or Thrift[16]. The Hbase shell allows a user direct access to manipulate and create data for Hbase, but does not allow for external communication.

The filtering system, as described in Chapter 5 communicates with Hbase using the Hbase Java API. Seeing the filtering system is developed in Java, the Hbase Java API is a natural library to use for this part of the system.

For the website interface, as described in Section 3.4, we can choose either REST or Thrift

for communicating with Hbase. The website runs on a LAMP[3] server, meaning the website is written in PHP. Both REST and Thrift has an underlying support for this language. REST returns a query result as either a XML or a JSON object, so every result returned to the website from Hbase contains the object type´s specific structure characters. Thrift on the other hand sends the data without it being a data type and relies on the client and server libraries to handle and structure the transmitted data. Seeing Thrift then sends less data than REST, we can also assume Thrift has a lower response time than REST. Therefore, we choose Thrift as our communication method between Hbase and the website interface.

4.4.1 Comparing Response Time of Different Thrift Querying Methods

When developing the website interface we tested the response time of different ways of doing queries using Thrift, and comparing them to the response time from using the Hbase Java API. Here, we should also keep in mind that the queries executed by Thrift were localhost queries, meaning that both Hbase and the website run on the same machine. The Hbase Java API on the other hand executed queries from a remote machine on the same local network. Figure 4.1 show the results of the test, listing the results of the different query types. All queries ask for the last 20 relations stored to Hbase, specified by individual TV shows. The "binary prefix" query for both Thrift and Java do a prefix match on row keys, with the row key structured as "topic_id::(Long.MAX-document_id)". The prefix match checks for the topic id and the delimiter “::”. The “regex string” query searches for any row key with a match to the given regex. The row key design for these queries is defined as "(Long.MAX-document_id)::topic_id", where we specify the query regex as the topic id, followed by a document end marker, meaning no more characters can follow the topic id.

The results shown in Figure 4.1 show us that the all queries done using Hbase Java API has a response time of 6-7ms, while Thrift using PHP has some varied results. The "binary prefix" for Thrift increases for every TV show query. The algorithm queries the shows alphabetically, so it seems this method does a top-bottom search. The higher number of rows we pass, the longer the response time gets. The "regex string" query test for Thrift shows a spike for show query number five. The row keys start with the document ids, meaning the newest tweets are always on top

[3]LAMP – Linux, Apache, MySQL, PHP

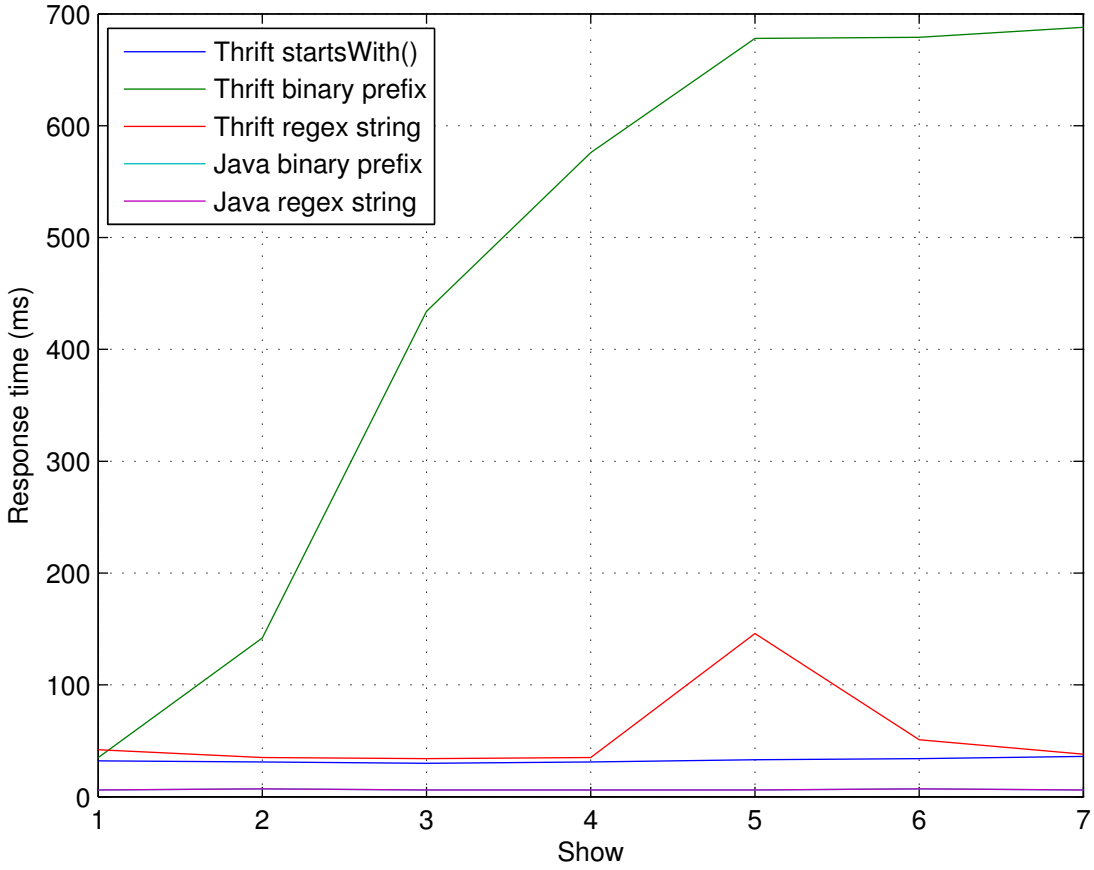

Figure 4.1: Hbase filter response time

of the relation table, independent of the topic id. The results here can be a result of a low rate of documents being stored for that particular show in the current timespan. The Hbase Thrift function "startsWith()" has a response time of 30-36ms, leaving a uniform distributed response time for getting query results using Thrift. The function does not have much documentation, so there is no exact way of comparing how it works in relation to the row key filters we have previously tested. From what we can see, the function opens a scanner and searches for a given prefix. We used the same row key structure for this function, as we did for the "binary prefix" queries. As the results show, there is a big difference in response time for each of the querying methods. The function "startsWith()" performs best for Thrift using PHP, both in regard to response time as well as getting a uniform distributed response time. Even so, the Hbase Java API is the most efficient, using on average a fifth the response time of the “startsWith()” function.

Chapter 5

Filtering System

In this chapter, we explore the two-stage filtering method we use to determine if a document is topic relevant or not. The first filter focuses on recognizing popular phrases for given topics in a document, while the second filter uses a machine learning method to decide if a document is relevant. The latter filter is meant to improve upon the results of the first filter using feedback data from the end user group. We first introduced this proposed filtering method in [1], and continue to use the same principle here. The filtering process is shown in the flowchart in Figure 5.1, where we can see that every document sampled from Twitter is put through the first filter. This gives us our initial set of relations and if the system has enough end user feedback we also put the document through the second filter, classifying the relations established from the first filter. The final set of relations determined by the filtering system is then stored to Hbase and displayed for end users through the website interface described in Section 3.4.

5.1 The First Filtering Step

The first filtering step try to determine if a document is loosely related to a topic, given its text contains some known phrase or word we associate with our topics. For each topic we define, we also define a regular expression for that topic. The regular expression expresses some textual structure we expect to find in a document if it is related to the given topic.

To give an example of some of the regular expressions used to evaluate a documents semantics, Table 5.1 lists regular expressions related to some of the topics we use in this project. Our

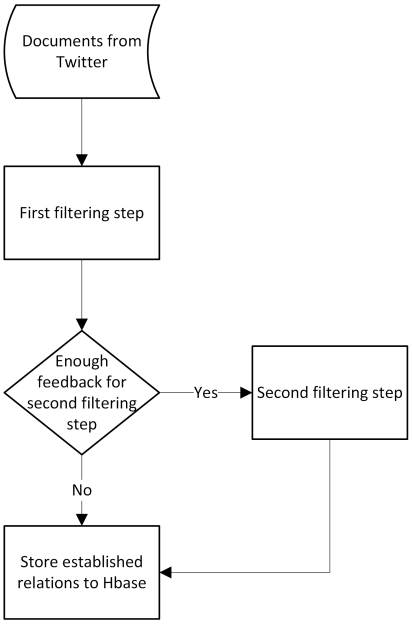

Figure 5.1: Filtering system

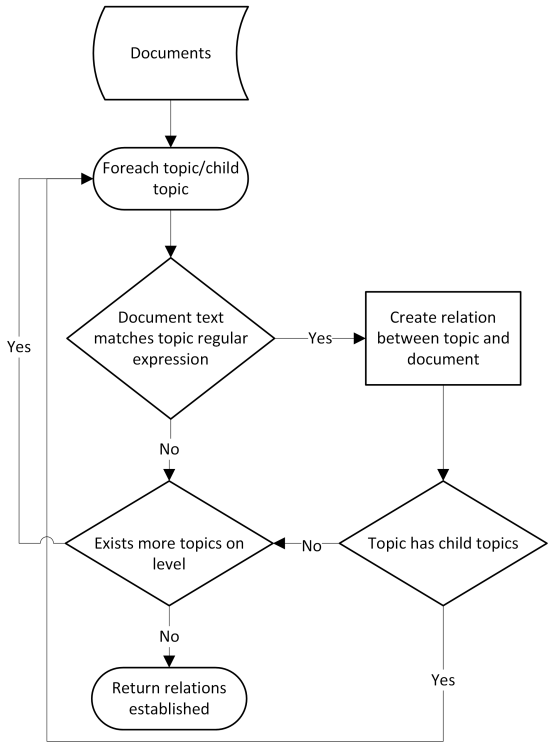

Figure 5.2: Flowchart of the first filtering step

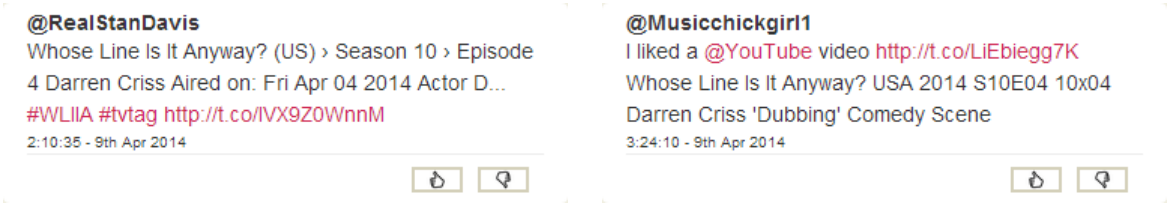

Figure 5.3: Document examples mentioning season and episode

topics are divided into three levels, or categories: TV shows, seasons and episodes. The regular expressions defined for the TV shows checks for a case-insensitive match of the TV show's name, as seen in Table 5.1 for the shows "Suits", "Supernatural" and "Whose Line Is It Anyway? (US)".

Figure 5.3 shows two document examples, where the document text has a regular expression match for season 10 and episode 4 for the TV show, "Whose Line Is It Anyway?". The regular expression for season 10 in the previously mentioned show, matches the document text "Season 10" in the first example. In the same manner the regular expression for episode 4, matches the document text "Episode 4". In the second example season 10 and episode 4 both match the document texts "S10E04" and "10x04".

Topic (show/season/episode)	Regular expression
Suits (show)	(?i)suits
Supernatural (show)	(?i)supernatural
Whose Line Is It Anyway? (US) (show)	(?i)(whose\s*line(\s*is\s*it\s*anyway(\?)*)*)
Season 1 (season of a show)	(?i)(((episode\s*)*(1|01)\.\d*\d)|((season|series |sn|s)\s*(1|01)(\D|$))|((01|1)x\d*\d))
Episode 1 (episode in a season)	(?i)(((episode\s*)*\d*\d\.(1|01)(\D|$))\|(\s\d\d* x(1\|01)(\D\|$))|(((sn|s)\s*\d\d*\s*)*(episode|ep |e)\s*(1|01)(?!\.)(\D|$)))
Episode 11 (episode in a season)	(?i)(((episode\s*)*\d*\d\.11(\D|$))\|(\s\d\d*x11 (\D\|$))|(((sn|s)\s*\d \d* \s*)*(episode|ep|e)\s* 11(?!\.)(\D|$)))

Table 5.1: Regular expression examples

In [1] we expressed an equation for the filtering step, which we reintroduce in Equation 5.1. For a given topic T and document d, the algorithm checks if there exists a match m for T in the document text d_t. A match is found if any part of the regular expression defined for T is found in d_t.

$$\exists m_T \in d_t, R(d,T) \tag{5.1}$$

Equation 5.1: First filtering step

In Algorithm 5.1 the filtering step is shown in pseudo code, where the previously stated equation expresses how the "relation(d, t)" function relates a document to a given topic. The algorithm defines $T[]$ as a list of all topics from the topic tree, where the topic's parent topic does not exist. In other words, $T[]$ is a list of all root topics in the topic tree. All documents passing through the system are put through the "filter(d)" function, which returns a list of relations established between the document and topics from the topic tree. For every root topic, we try to establish a relation between the document and the given topic, using the "relateDocument(d, t)" function. This function recursively checks for regular expression matches for the given document and topic and if a match is found, the same function is performed for all child topics c in the list of children c_t of the given topic t. All proposed relations from the filtering step are returned to the system. Figure 5.2 illustrates the algorithm of the first filtering step, shown in

Algorithm 5.1 First filtering step algorithm

Require: $R[] \leftarrow$ new relation list
Require: $T[] \leftarrow \forall t \in T, \quad \nexists P_t \in t$
  function FILTER(d)
    for all $t \in T[]$ **do**
      $R[] \leftarrow$ relateDocument(d, t)
    end for
    return $R[]$
  end function
  function RELATEDOCUMENT(d, t)
    $r[] \leftarrow$ new relation list
    if $e_t \in d$ **then**
      $r[] \leftarrow$ relation(d, t)
      for all $c \in c_t$ **do**
        $r[] \leftarrow$ relateDocument(d, c)
      end for
    end if
    return $r[]$
  end function

Algorithm 5.1, as a flow chart. The flow chart gives a more simplistic overview, but shows every step of the filter clearly, with all decisions and processes shown with detailed texts.

5.2 The Second Filtering Step

We apply data mining and machine learning techniques to determine if a document is related to a topic. Machine learning is a term used to describe a self-learning computer algorithm, or an algorithm implementing a form of artificial intelligence[17]. By providing reliable outcome of previously processed data, a machine learning algorithm adjusts to improve on its ability to classify incoming data correctly. For using data mining techniques on the documents passing through our system, we use Weka[18]. Weka is a collection of data mining functions and includes a selection of machine learning algorithms, or classifiers. In addition, it allows for data pre-processing, clustering, regression, association rules and visualization. This data mining toolbox runs as an application with a graphical user interface, but also delivers a Java API for doing data mining tasks directly in code. In general, we use Weka to train a model based on feedback to the system and classify new incoming documents using the trained model. We periodically update

Filter	Precision	Recall	F-Measure
Filter #1	0.754	1	0.860
Filter #2			
Naive Bayes	0.822	**0.987**	0.897
Random Forest	**0.825**	0.984	**0.898**
C4.5	0.824	**0.987**	**0.898**

Table 5.2: Cross-validation results using a static data set

the model to incorporate new feedback.

The second filtering step is designed as a machine learning filter and as we measured in [1], the classifier, Random Forest[19], is the most suitable classifier for our system. In [1] we evaluated three different classifiers, doing a cross-validation on a trained data set to measure the precision, recall and F-measure of the different classifiers. Table 5.2 shows the results of the cross-validation test, where we tested the Random Forest[19], Naive Bayes[20] and C4.5[21] classifiers. Random Forest, as described in Definition 1.1 in [19], is a classifier consisting of a collection of tree-structured classifiers $\{h(x,\Theta_k), k = 1,...\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x.

We need to define a threshold for when the second filter should start to classify relations established by the first filter. In [1] we expressed the equation shown in Equation 5.2 for defining the threshold for when the system actively starts using the second filter.

$$\sum_{0<i<n} F_i(T,d) > t \tag{5.2}$$

Equation 5.2: Threshold for using the second filtering step

If the number of feedbacks $F_i(T,d)$ registered in the system exceeds the threshold t, we deem the feedback count to be high enough, so that we can train a model for the classifier. The threshold we use in this project is set to 500. This gives us a fair amount of feedback instances to build a model.

In addition to defining a threshold for when to start using the second filtering step, we also need to establish a model retraining interval, i. This interval defines how frequently the system

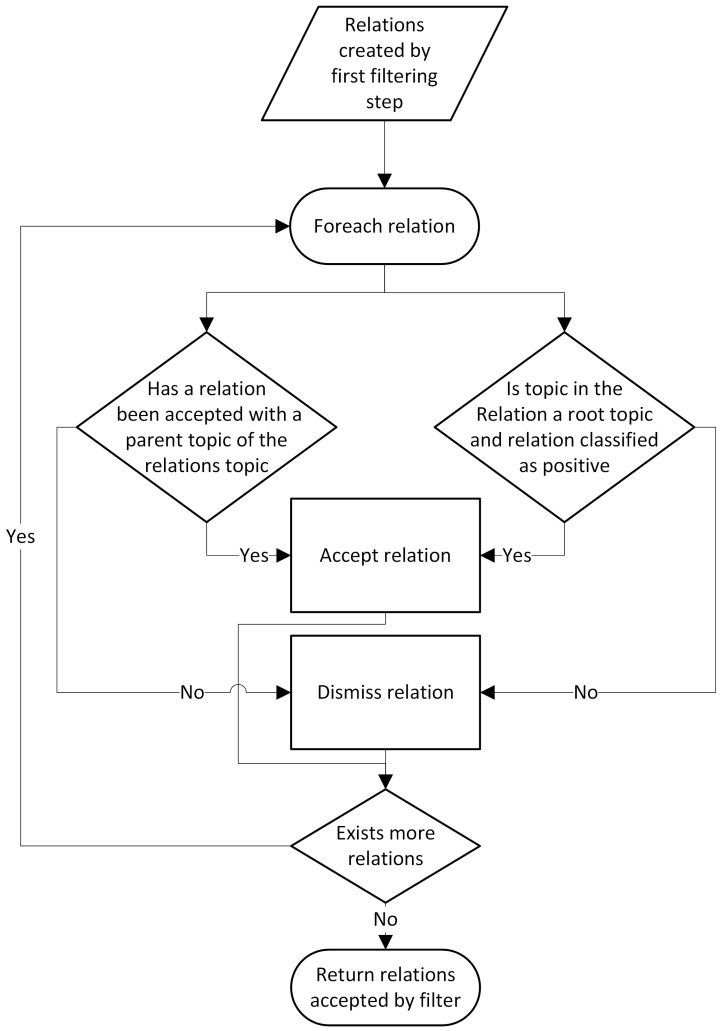

Figure 5.4: Flowchart of the second filtering step

retrains the model and replaces the model present for our classifier, with the new and updated model. Defining i should be based on the end user feedback frequency f, meaning the higher the frequency, the lower the interval. The interval is expressed in Equation 5.3,

$$i = \frac{k}{f} \tag{5.3}$$

Equation 5.3: Model retraining interval for the second filtering step

where k is a constant. For this project however, the end user frequency is not high due to only a handful of end users giving feedback to the system. The interval is set to 24 hours, which is suitable for the experimental setup explained in Section 6.1.

Algorithm 5.2 Second filtering step algorithm

Require: $M \leftarrow$ trained model based on feedback
Require: $R[] \leftarrow$ new relation list
  function FILTER($r[]$)
    for all $r \in r[]$ **do**
      if $(\nexists P_{t_r} \in t_r \wedge \text{classify}(d_r, t_r)) \vee (\exists P_{t_r} \in t_r \wedge \exists r_i \in R[], \quad T(t_r, t_{r_i}))$ **then**
        $R[] \leftarrow r$
      end if
    end for
    return $R[]$
  end function
  function CLASSIFY(d, t)
    classify d, t relation using M
    return is d and t related according to classifier?
  end function

Now, using Random Forrest as a classifier, we design a filtering method that eliminates false positive matches created by the first filter. By using end user feedback for the topic-document relations and a semantic deduction of the feedback data, the classifier gives a probable classification of any live relation as $positive$ or $negative$. The filter receives a list of the proposed topic-document relations created by the first filtering step. For every relation that has a root topic as its relational topic, we conduct a classification for that relation. If the relation is classified as $positive$ the relation itself and any relation with a relational topic set as a sub-topic of the relation's topic passes the filter. To give an example, we only classify relations that relates a tweet to a TV show (e.g. Agents of S.H.I.E.L.D, Doctor Who or Suits). If the relation then classifies as $positive$, any relation that has been established by the first filter and contain a sub-topic of that show, is also considered as $positive$ (e.g. a specific season or episode). If the relation is classified as $negative$, the relation is dropped, as is the relations containing a sub-topic of the relations topic. The process can be seen in Figure 5.4.

Algorithm 5.2 displays pseudo code for the algorithm used in our project. If either of the cases previously explained and shown in Figure 5.4 occurs, the relation is accepted and added to the return result. The algorithm accepts a relation r if either of the following cases occur:

1. Does the parent topic P_{t_r} not exist for the current relation's topic t_r and does the classifier classify the relation r as $positive$?

2. Does the parent topic P_{t_r} exist for the current relation's topic t_r and does there exist a previously accepted relation r_i, where r_i's topic t_{r_i} is the parent topic of t_r?

The "classify(d, t)" function returns a boolean value of whether the document d is classified as $positive$ or not. To do this classification we need to organize the end user feedback as a data set structured for Weka. This structure is explained in the following subsection.

5.2.1 Features

As we previously stated, the system periodically rebuilds the machine learning model with an interval of 24 hours. This model is used by our classifier to predict if an incoming document classifies its relevance to our given topic as $positive$ or $negative$. To be able to do this we need to create a viable model design, which Weka is able to interpret and use to do a statistical prediction using machine learning algorithms.

The machine learning model contains five different attributes, which tell us something about a given document. Before we begin examining these attributes, we need to add one more definition to our topic tree. We define the terms "positive trackers" and "negative trackers" for each root topic in the topic tree. These trackers are regular expressions, containing words or phrases we expect to find or not find in a document related to the given topic.

Table 5.3 shows the five attributes we define for our model, or relation, "document". The attribute "positive_trackers" is a numeric attribute, telling us how many positive trackers of a given topic, the document contains. "positive_trackers_as_hashtags" is a numeric attribute telling us how many positive trackers are written as hashtags. "negative_trackers" is a numeric attribute, counting the number of negative trackers present in the document. "children_trackers" is a numeric attribute, which tells us if a document relates to a season or episode. If a document men-

```
@relation document
@attribute positive_trackers numeric
@attribute positive_trackers_as_hashtags numeric
@attribute negative_trackers numeric
@attribute children_trackers numeric
@attribute relation {positive, negative}
```

Table 5.3: Data definition for Weka and numeric data entry examples

3, 3, 0, 0, positive
2, 2, 0, 1, positive
1, 1, 1, 0, positive
1, 2, 0, 2, positive

Table 5.4: Document data entry examples for Weka

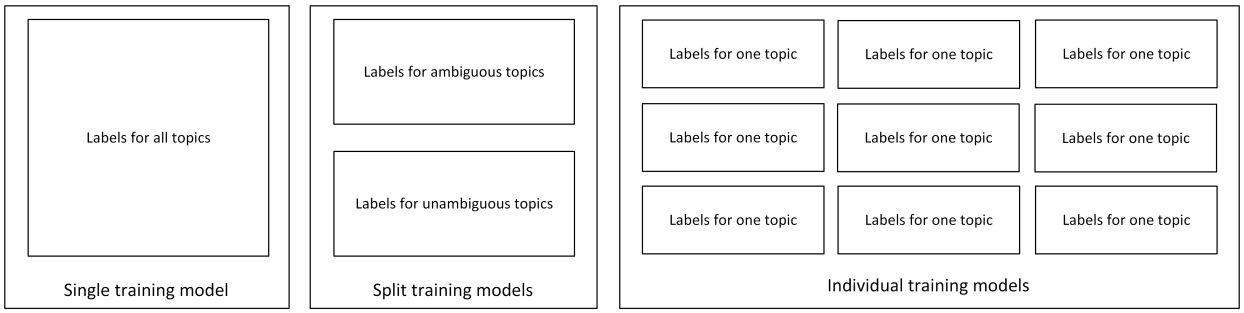

Figure 5.5: Graphical representation of the model training strategies

tions a season but not an episode, the attribute value would be "1" and if it mentions both, "2". In a more generic sense, the attribute counts the number of topic tree levels below the root topic that contain a match for the given document. Lastly, the attribute “relation” tells us whether the document has been marked as $positive$ or $negative$ by an end user and this is the attribute we want to predict for incoming documents. Some data entry examples can be seen in Table 5.4.

5.3 Model Training Strategies

A classifier depend on a good model to be able to classify documents correctly. As the number of instances increase in a model, a classifier’s ability to classify documents should improve. We want to implement three different model training strategies to see how they perform in comparison to each other. The first model training strategy creates a single training model, using labels created from feedback for all topics. We group all labels to a single model, independent of what topic a given feedback is related to. The second model training strategy use two models, where we distribute the labels created from user feedback based on a topic’s title being ambiguous or unambiguous. The last model training strategy uses individual models, one for each topic, where each model contains labels related to a specific topic.

The different model training strategies are displayed in Figure 5.5. We refer to the differ-

ent strategies as single training model, split training models and individual training models, respectively. The single training model is implemented into the system, while we evaluate the effectiveness of the different strategies in Section 6.3.

Chapter 6

Experimental Evaluation

In this chapter we do an evaluation of the proposed system, focusing on how well the filtering system accept relevant documents (true positives) and reject irrelevant documents (false positives and true negatives). First, we look at some key features of the experimental setup, followed by statistics from the live system test. Then, we evaluate the classifiers used for predicting the relation of incoming documents to the system and compare the filter design to some alternative designs. Our focus is evaluating the precision of the classifiers and their evolution as more feedback is added to the training model. Seeing our goal is to create an event summarization system, we want the system to eventually only display correct document-topic relations. Recall is therefore not our main concern and we would rather miss some information, than displaying incorrect information. We can consider the system to be structured with a goal of having an eventually perfect precision, with little concern of missing false negative filtering results. One could argue that false negative results in the filtering system architecture are acceptable, seeing microblog posts in general contain very little information and documents with false negative results contain even less information. Lastly, we propose new trackers for the system, based on feedback the system has received over the live system test period.

6.1 Experimental Setup

This section focuses on describing parts of the system, which allow the filtering system to use end user feedback to improve on its document classifications. First, we list all topics used in

Ambiguous	Unambiguous
Firefly	Marvel's Agents of S.H.I.E.L.D.
Suits	The Big Bang Theory
Supernatural	Doctor Who (2005)
Vikings	Hannibal
	Whose Line Is It Anyway?

Table 6.1: Topics divided into ambiguous and unambiguous groups

the live system test and show which topics are ambiguous and unambiguous. Then, we explain how the system registers end user feedback, using the website interface described in Section 3.4. Lastly, we explain the evaluation measure used for calculating the classifier precisions, based on the models created during the live system test.

6.1.1 Topics Used in the Live System Test

As expressed in the introduction to the thesis, we use 9 popular TV shows as topics when performing the live system test. The topics are the TV shows: "Marvel's Agents of S.H.I.E.L.D.", "The Big Bang Theory", "Doctor Who (2005)", "Firefly", "Hannibal", "Suits", "Supernatural", "Vikings" and "Whose Line Is It Anyway?". Table 6.1 displays the topics as they are separated by the topic title being ambiguous or unambiguous. The TV shows "Firefly", "Suits", "Supernatural" and "Vikings" are defined as ambiguous topics, while "Marvel's Agents of S.H.I.E.L.D.", "The Big Bang Theory", "Doctor Who (2005), "Hannibal" and "Whose Line Is It Anyway?" are defined as unambiguous.

6.1.2 Registering Positive and Negative Feedback

To be able to create a machine learning model, we first need a method for the end user to tell the system if a document is related to a topic. Figure 6.1 shows two examples of documents displayed for a topic. The "thumbs up" and "thumbs down" buttons represents a way for the end user to give feedback to the system, which registers the document-topic relation as $positive$ or $negative$ as a button is pushed, respectively.

When an end user has registered feedback for a document, no other end user can edit or change the value of the feedback for that document. The first screenshot shown in Figure 6.1

Figure 6.1: Document examples for giving feedback to the system

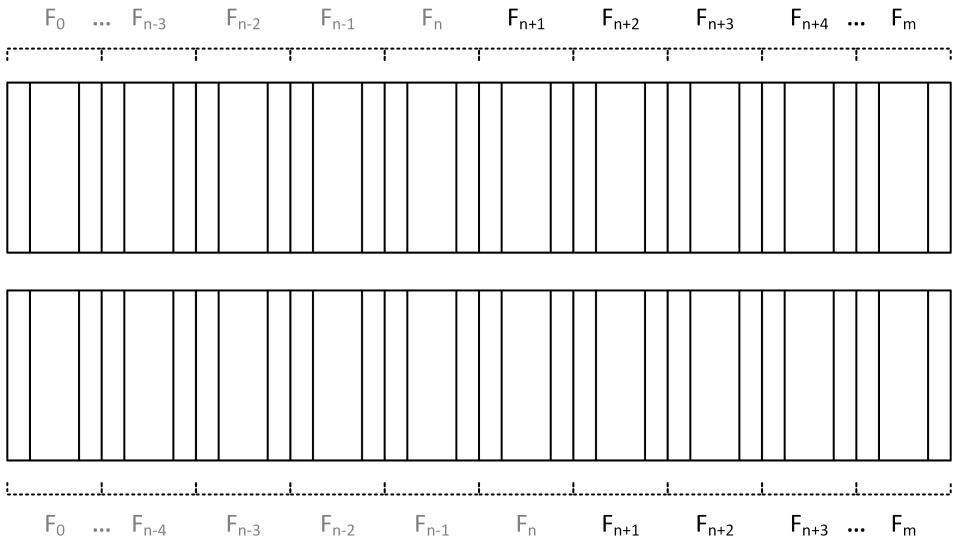

Figure 6.2: Sliding window method for precision calculation

shows a document that does not have any registered feedback. The second screenshot shown in Figure 6.1 shows a document that has been registered as $negative$. Any document displayed by the system that has a registered feedback has a disabled button, which displays the registered vote.

6.1.3 Evaluation Measures

In Section 6.3 we evaluate the classifier used in the second filtering step. Before we do so, we briefly examine the methods we use for evaluating the classifier, Random Forest and its competing classifiers, namely Naive Bayes and C4.5. First, we look at different ways of calculating the precision of a classifier and their pros and cons. Then, we describe how we create the training model and test data for evaluating the precision of a classifier.

We evaluate the classifiers using both micro and macro-averaged precision[22]. Micro-averaged

precision is calculated as the precision of a trained data set, $S_{training}$, tested on a test data set, $S_{testing}$. The test data set contains data instances from all labels of the system, meaning feedback gathered in the testing period for all topics present in the system. We can then denote $S_{testing}$ as $\sum_{1i}^{n} F_i(T,d)$. Macro-averaged precision on the other hand, calculates the precision of $S_{training}$ by calculating the precision for every topic individually and then averaging the result, leaving a label's impact on the final precision uniformly influential amongst the topics.

Micro-averaged precision gives us a more realistic view of the precision of the classifier, using our system design. The filtering step explained in Section 5.2 use a global set of training data and not individual training data sets for each topic. However, by using this precision calculation, we do not compensate for feedback frequency differences between the different topics. To compensate for the feedback frequency macro-averaged precision can be used.

When calculating the precision of a classifier we need to create a training model and a set of test data, which we use to test the training model. Our variation build on a sliding window technique, which is shown in Figure 6.2. If F_i is the feedback for a given day, we use $\sum_{1i}^{n} F_i$ to build a training model and create testing data based on feedback from the rest of the live system test data, $\sum_{n+1i}^{m} F_i$.

6.2 Live Testing Statistics

The system has been running live from April 2nd to May 15th and in this time we have gathered some statistics, showing the activity of the system. Due to a logging error, the statistics from the third through the sixth day were lost, leaving us with data from 40 out of 44 days. Figure 6.3 shows how many documents passes the first and second filter in a day by day perspective. We see that documents periodically pass the system at higher rates than other times and that the second filter on average accept 28.95 percent of the documents accepted by the first filter. Figure 6.4 shows the percent of documents that passes the filtering system. For each day, the percent of documents passing the filter is expressed as $\frac{\sum_{1i}^{n} f(d_i)}{\sum_{1i}^{n} d_i}$, where $\sum_{1i}^{n} f(d_i)$ is the number of documents that has passed the filter and $\sum_{1i}^{n} d_i$ is the total number of documents passing through the system.

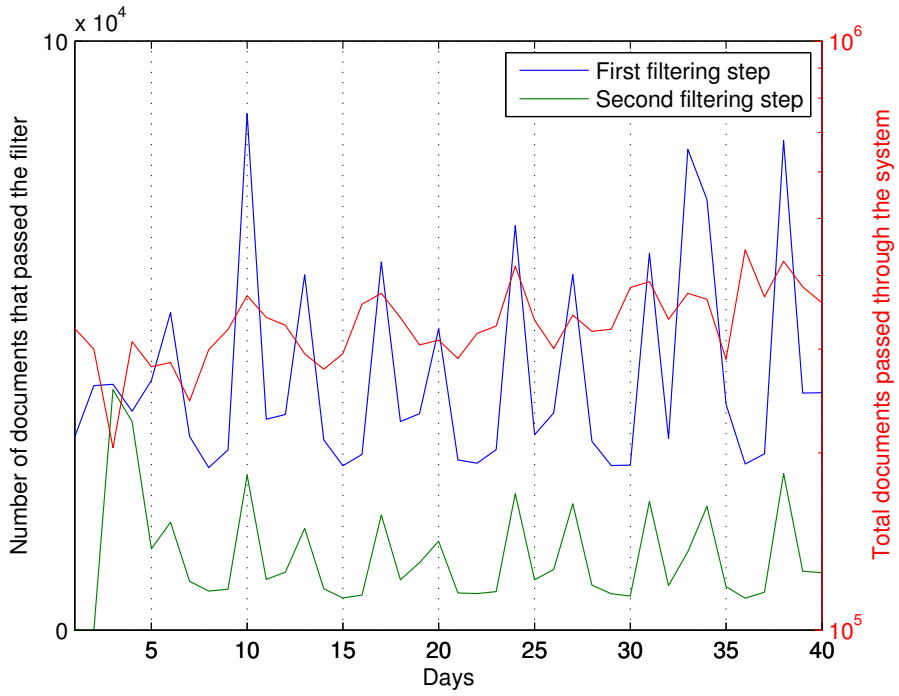

Figure 6.3: Documents passed filtering steps, day by day

6.3 Classifier Evaluation

In Section 5.2 we chose to use Random Forest as the second filtering step's classifier during the live system test, based on the results we presented in Section VIII in [1]. We evaluate the precision of this classifier, as well as Naive Bayes and C4.5, comparing their results on the daily-created training models from the live system test. As we argued in the beginning of this chapter, we focus on evaluating the precision of the classifiers, using the daily-created training models.

The classifiers are evaluated using both micro and macro-averaged precision. First, we evaluate the classifiers using the single training model, where we use a single training model to predict all document-topic relations. Then, we evaluate the classifiers using split training models, which differentiates the feedback data into two categories: ambiguous and unambiguous topics. We continue to evaluate the classifiers using individual training models for all root topics and lastly, we compare the results of the macro-averaged precision for the different model training strategies. The micro-averaged precision results shown in the following subsections also show how the feedback varies in the testing period. If F_p is the positive feedback, F_n is the negative feedback and F_t is the total number of feedback for a given period, the feedback shown

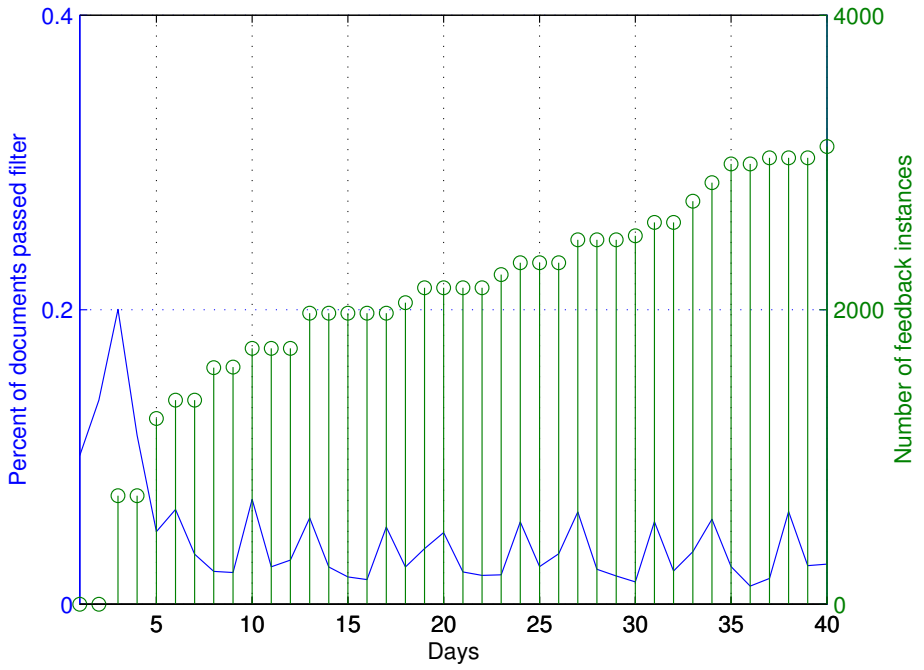

Figure 6.4: Percent of documents passed both filters compared to feedback count, day by day

in the graphs is defined as $\frac{F_p}{F_t} - \frac{F_n}{F_t}$. This represents the percent difference between positive and negative feedback, where a positive percentage means a higher frequency of positive feedback, and vice versa.

The precision score results showed that the classifier C4.5 proved to either have a precision score equal to Random Forest or score lower than Random Forest at times, for the individual training models. To make the graphs more readable we exclude C4.5 from these graphs, though the precision scores were generated. Averages written for the different training model strategies are calculated by creating a single vector containing the averaged precisions of the classifiers for each model, and calculate the average of the period. This does not accurately describe the precision of a classifier, but gives us an idea of the overall precision development.

6.3.1 Evaluating Classifiers Using Single Training Model

Figure 6.5 displays the micro-averaged precision of the three classifiers, Random Forest, Naive Bayes and C4.5, using the sliding window method previously explained. The graph shows the classifier precision evolution over the duration of the live system test. Ideally, we would expect

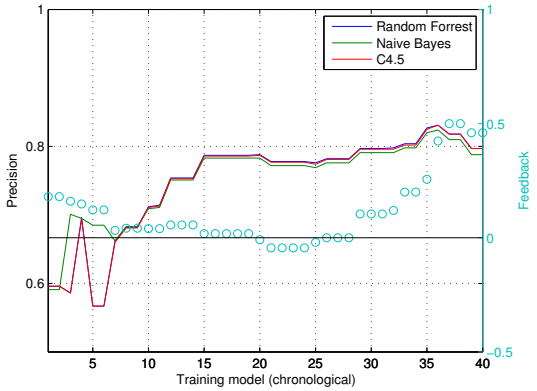

Figure 6.5: Classifier precisions over time using micro-average

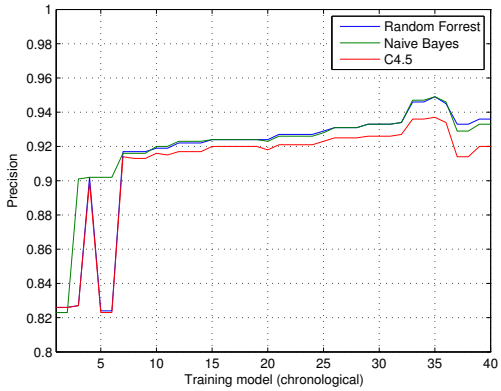

Figure 6.6: Classifier precisions over time using macro-average

the precision to become better over time, as more feedback is registered to the system. For the first 6 training models, or days, we see the precision fluctuate, with an average of 0.152 more positive feedback than negative. Overall, the precision do improve over time, with the exception of the first days. There are some minor precision drops, which can be explained by having a marginally higher rate of negative feedback than positive at the given time and the high difference rate later on. Over the course of the testing period, we see a precision improvement of 0.1997, a total precision average of 0.7502 and an average of 0.8053 for the last 10 days. The figure shows Random Forest and C4.5 having marginally better precision than Naive Bayes overall. If we now look at Figure 6.6, we can see Random Forest and Naive Bayes having better precision than C4.5, with Random Forest scoring marginally better than Naive Bayes. The macro-averaged precision has an average precision of 0.9342 for the last 10 days, with an overall average of 0.915.

6.3.2 Evaluating Classifiers Using Split Training Models

Figure 6.7 and Figure 6.8 show the micro-averaged precisions for the unambiguous and ambiguous topics, respectively. We can see that the precision for unambiguous topics get worse over time, however it correlates to increasing negative feedback. For a model containing almost only positive instances, adding negative instances negatively affect the precision more than for a model containing the same amount of positive and negative feedback. Over the test period, the unambiguous model averages a precision of 0.9791 and the last 10 days, 0.9758. The preci-

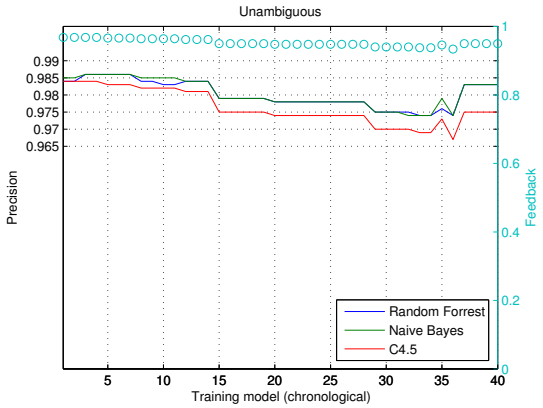

Figure 6.7: Classifier micro-averaged precisions for unambiguous topics

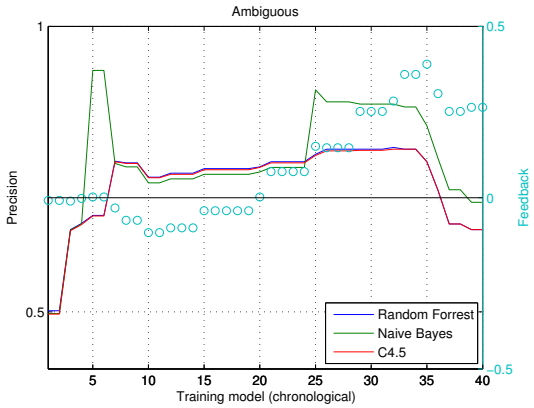

Figure 6.8: Classifier micro-averaged precisions for ambiguous topics

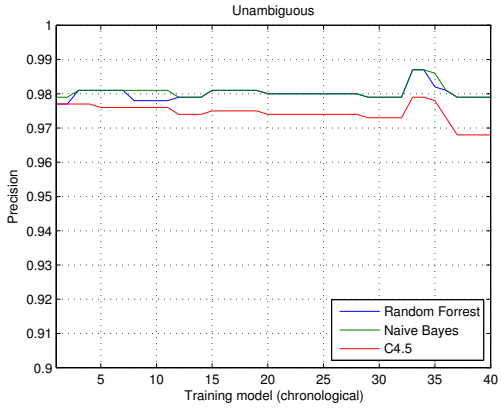

Figure 6.9: Classifier macro-averaged precisions for unambiguous topics

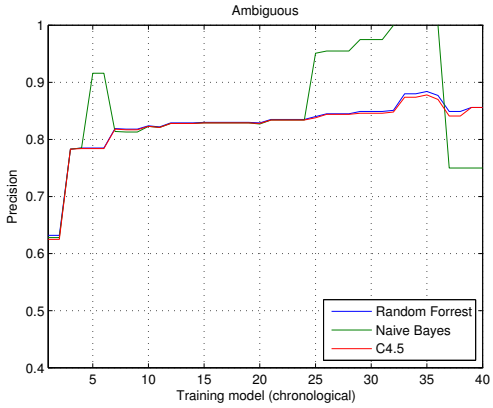

Figure 6.10: Classifier macro-averaged precisions for ambiguous topics

sion for ambiguous topics fluctuate and there appears to be a correlation between the feedback ratio and precision. As feedback change to contain more negative feedback, the precision scores lower and as the testing data gain more feedback that is positive, the precision scores higher. For the total test time the precision averages 0.7394 and for the last 10 days, 0.7488.

Figure 6.9 and Figure 6.10 show the macro-averaged precision for unambiguous and ambiguous topics, respectively. We see the precision for unambiguous topics fluctuate and slightly declines in the laps of the testing period. The precision for ambiguous topics improves over time for Random Forest and C4.5, but Naive Bayes fluctuates as the feedback ratio change.

Compared to the single model, the split models do not always improve upon the classifier precisions and the scores fluctuate. Naive Bayes scores better than Random Forest and C4.5, but also fluctuates more when dealing with ambiguous topics.

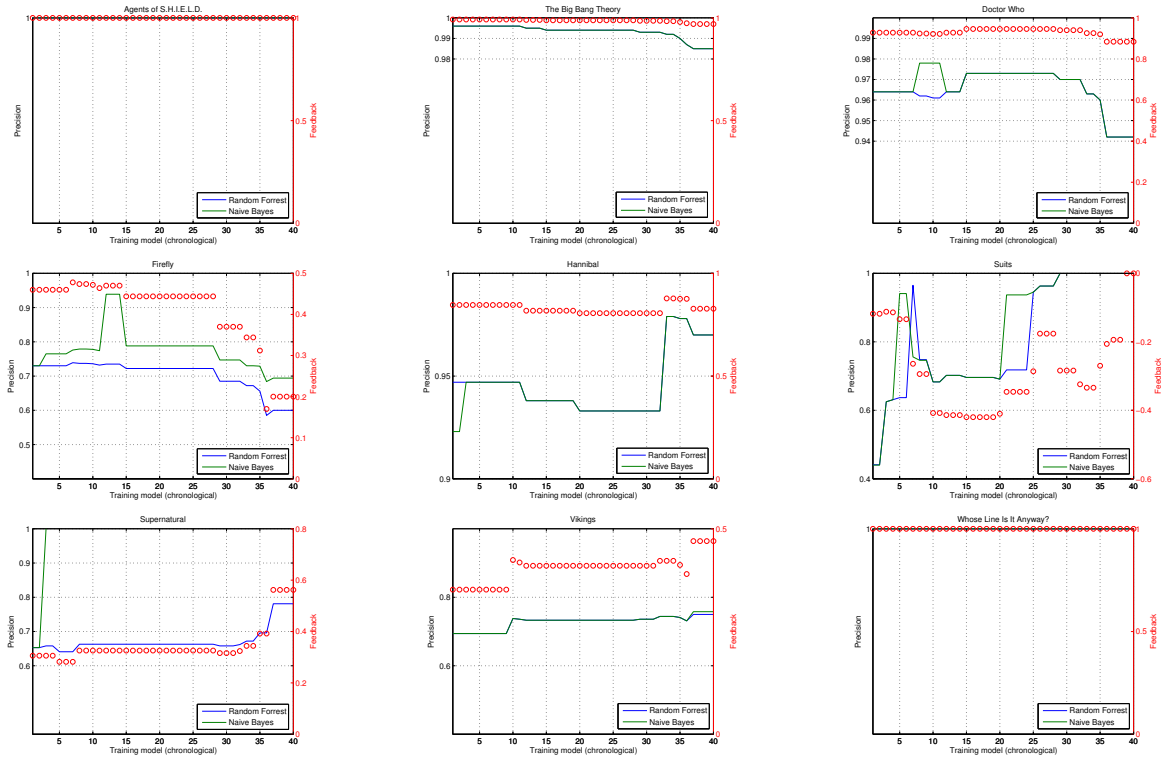

Figure 6.11: Classifier evaluation for individual topics

6.3.3 Evaluating Classifiers Using Individual Training Models

Figure 6.11 displays the classifier precisions for the individual topics. The precision for both "Agents of S.H.I.E.L.D." and "Whose Line Is It Anyway?" is 100 percent. This is due to all feedback given to the system for these topics are *positive*. Both "Agents of S.H.I.E.L.D." and "Whose Line Is It Anyway?" are unambiguous topics and we continue discussing the remaining unambiguous topics, namely, "The Big Bang Theory", "Doctor Who" and "Hannibal". Their precision scores lower over time, as more negative feedback is present in the test data and affects it. We can also see the models improving as more positive feedback is added to the test data set. Looking at the ambiguous topics, the precision scores for "Firefly" and "Suits" fluctuate over time, where the model for "Firefly" scores lower precision over time and "Suits" scores higher. "Supernatural" and "Vikings" both have precision improving models, with some deviation, as the test data increase its ratio of negative feedback.

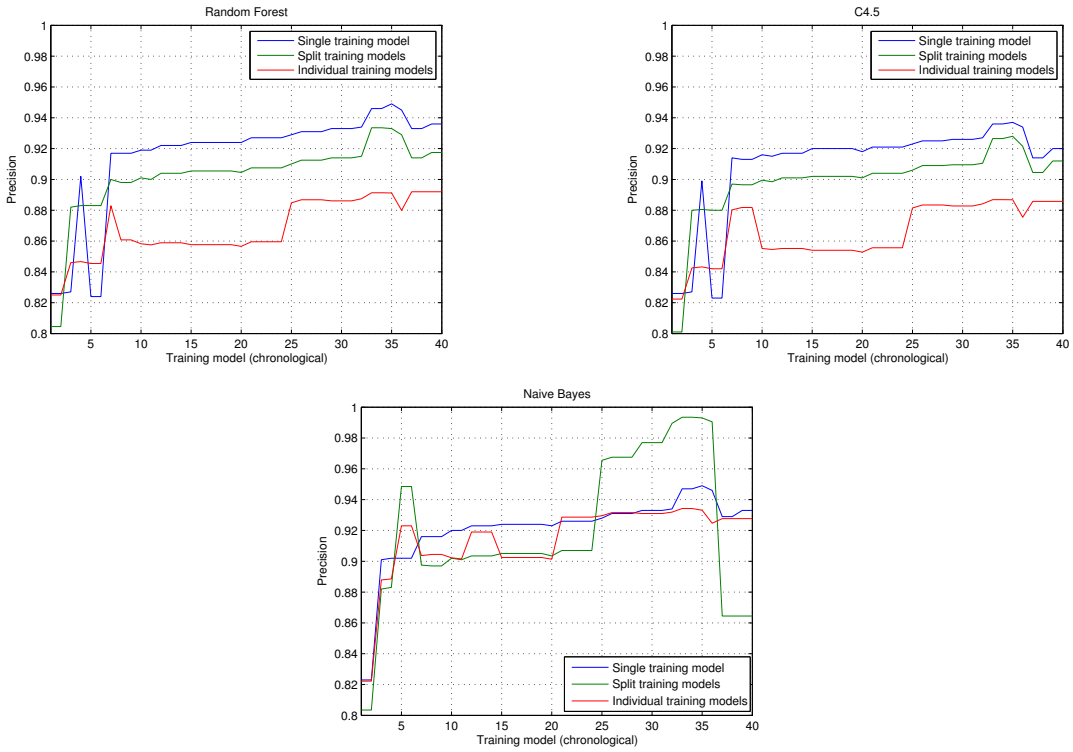

Figure 6.12: Plot of model training strategies over time

Model Strategy	Random Forest	Naive Bayes	C4.5
Single model	**0.933**	**0.933**	0.926
Split models	0.914	**0.977**	0.910
Individual models	0.804	0.849	0.801

Table 6.2: Comparing model training strategies

6.3.4 Comparing Macro-averaged Precision of the Model Training Strategies

Figure 6.12 shows the macro-averaged precision plots for the different classifiers, comparing how well the model training strategies perform. Using the single training model strategy generally has a better improvement curve and slightly better precision than the other strategies.

Table 6.2 compares the macro-averaged precisions using the different model training strategies. The precisions are calculated using $n = 30$ and $m = 40$, so that the training model contains the feedback $\sum_1^{30} {}_i F_i$ and the test data contains the feedback $\sum_{31}^{40} {}_i F_i$. This leaves a sizeable training model and sufficient test data. The table shows the split models strategy to score highest precision using Naive Bayes, however, as Figure 6.8, 6.10 and 6.12 show, the Naive Bayes precision in this period fluctuates and deviates from its prior development. The precision at this

time may be a natural result due to the nature of the feedback present in the trained model and test data at the given time and as seen for $n = 37$ in Figure 6.10 and 6.12, the precision drops drastically for Naive Bayes using the split models training strategy.

The precision scores using the single model strategy and Random Forest as classifier proves to maintain a stable and precision-improving development over time and scores second best at $n = 30$. If we compare the average precision for Random Forest and Naive Bayes, using the single model strategy and in the period between $n = 10$ and $n = 40$ (disregarding the irregularities for $n = [1, 9]$), we get the following results: the average macro-averaged precision for Random Forest is 0.9302 and for Naive Bayes it is 0.9298. Using the average micro-averaged precision Random Forest scores 0.7848 and Naive Bayes scores 0.7792. Overall Random Forest perform slightly better than Naive Bayes using the single model strategy, even though the macro-averaged precision at $n = 30$ are the same.

6.4 Proposing Topic Trackers Based on Feedback

In this section, we try to suggest some words or phrases that can be added to the system as either positive or negative trackers to the topics, based on feedback registered to the system. We use log likelihood[23] to estimate the popularity of words in documents related to a topic. Given a, b, c and d, where a is the frequency of a given word in documents related to a topic, b is the frequency of a given word in documents related to any topic, c is the frequency of all words in documents related to a topic and d is the frequency of all words in documents related to any topic, we can estimate the log likelihood of a given word. The log likelihood is calculated as $LL = 2 * ((a * ln(a/E_1)) + (b * ln(b/E_2)))$, where $E_1 = c * (ab)/(cd)$ and $E_2 = d * (ab)/(cd)$.

Table 6.3 and Table 6.4 display the log likelihood scores of words found in documents related to the ambiguous topics, “Firefly”, “Suits”, “Supernatural” and “Vikings”. The scores are separated into two tables, based on a document’s feedback to be either $positive$ or $negative$, respectively. Both tables display the highest scoring words. The crossed out words are words either found in every day speech or not relevant towards the given topic. The plain words are words relevant to the topic and already used as a tracker for the given topic, e.g. “serenity” is already used as a positive tracker for the topic “Firefly”. The bolded words are words relevant

Firefly		Suits		Supernatural		Vikings	
firefly	392	**suits**	321	**supernatural**	423	**vikings**	412
serenity	131	~~firefly~~	114	spn_updates	73	**historyvikings**	150
nathanfillion	76	~~agentsofshield~~	96	mishacollins	51	ragnar	88
browncoats	44	harvey	77	jensenackles	51	~~gblagden~~	45
~~thee~~	34	~~bang~~	75	dean	47	**history**	34
~~s1e1~~	34	~~vikings~~	75	**spn**	39	~~forward~~	19
~~release~~	34	~~theory~~	73	~~set~~	33	~~done~~	19
flyin	34	~~big~~	62	crowley	24	~~2~~	18
fireflyrpg	34	~~doctor~~	59	**bts**	24	**season**	14
~~bouletcorp~~	34	**suits_usa**	50	~~saying~~	19	~~blood~~	14
whedonesque	34	mike	42	~~than~~	18	~~liked~~	14

Table 6.3: Log likelihood scores for $positive$ feedback

Firefly		Suits		Supernatural		Vikings	
firefly	280	**suits**	22	**supernatural**	185	**vikings**	328
game	41	~~games~~	10	~~he~~	18	**bridgewater**	88
sales	29	**purple**	8	~~know~~	18	**qb**	57
~~2~~	29	~~holding~~	8	~~3~~	14	**teddy**	51
toothbrushes	24	~~game~~	8	~~the~~	14	~~back~~	14
light	23	~~the~~	8	~~us~~	10	~~and~~	14
games	19	~~play~~	6	~~its~~	10	~~at~~	13
solar	19	~~led~~	6	~~by~~	7	~~the~~	12
play	19	~~grow~~	6	~~what~~	7	~~into~~	10
~~9~~	19	**glass**	6	~~from~~	6	~~in~~	8
~~the~~	14	**freddie**	6	~~to~~	5	~~no~~	7

Table 6.4: Log likelihood scores for $negative$ feedback

for the topic, but not used as trackers in the system. This displays the newly suggested trackers, both positive and negative. We can see that the topic title is a high scoring word for both positive and negative tracker suggestions. This implies that topic titles should not be set as trackers, for ambiguous topics.

Chapter 7

Conclusion

Through the course of this thesis, we have presented a viable method for doing tweet-based event summarization. Our initial training model design, grouping feedback for all topics in a single model provide a mean precision of 0.7502 cross-classifier. The design proved to improve the classifier precision over time. By dividing the training model into two different models, separating ambiguous and unambiguous topics, we see an improvement in the classifier precision for unambiguous topics, though the precision get marginally lower over time, while the ambiguous topics still prove challenging to classify correctly. By creating individual training models for each topic, we see high precisions for unambiguous topics and fluctuating precisions for ambiguous topics. Overall, the split and individual training model strategies prove to have lower precision than the single training model strategy. Using a single model with the Random Forest classifier, prove to be better suited for the proposed system design.

In Section 6.4, we proposed topic trackers based on feedback provided to the system. The topic title for the ambiguous shows scores high for both $positive$ and $negative$ feedback, implying the title should not be used as a tracker for these shows, but rather rely on other trackers. The newly suggested trackers are based on the content of documents registered as $positive$ or $negative$ for a topic and can be added to the second filtering-step to improve upon the training model. Doing so, requires that the system rebuild the entire training model from scratch, meaning all data entries related to a document need to be adjusted to coincide with the new tracker sets.

7.1 Future Work

Some more time should be spent on researching the precision challenges for ambiguous topics, as we faced with “Firefly” and “Suits”. The system as it is designed counts all trackers and all instances of trackers, meaning if a document contains the text “Suits, suits, suits!” and “suits” is a positive tracker, the document get three positive matches. Reducing this to check for a tracker and not the frequency of that tracker could be applied to the system to see if it improves the precision rates for ambiguous topics.

The system accepts all documents streamed from Twitter. Reducing the accepted documents to only originally published documents could be applied and studied to disallow duplicate data entries. This reduction would not accept any document that is either a re-tweet or a reply to an already published document, whereas the last statement is negotiable.

An implementation for editing the trackers defined for the topics should be developed, where a user can use the log likelihood method to determine new $positive$ and $negative$ trackers for a topic, given enough feedback. Doing this would also require the second filtering step to rebuild the data entries in the training model as trackers are changing.

The thesis has focused on building a machine learning filter for the root topics. The filter can be expanded to incorporate training models for lower tier topics as well. E.g. a model for seasons and a model for episodes.

References

[1] Chris Haaland. Introduction to tweet-based event summarization. 2013.

[2] M-Dyaa Albakour, Craig Macdonald, and Iadh Ounis. On sparsity and drift for effective real-time filtering in microblogs. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 419–428, New York, NY, USA, 2013. ACM.

[3] Edward Benson, Aria Haghighi, and Regina Barzilay. Event discovery in social media feeds. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 389–398, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[4] Marco Pennacchiotti and Ana-Maria Popescu. A machine learning approach to twitter user classification. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[5] Edgar Meij, Wouter Weerkamp, and Maarten de Rijke. Adding semantics to microblog posts. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 563–572, New York, NY, USA, 2012. ACM.

[6] Tariq Mahmood, Tasmiyah Iqbal, Farnaz Amin, Wajeeta Lohanna, and Atika Mustafa. Mining twitter big data to predict 2013 pakistan election winner. In *Multi Topic Conference (INMIC), 2013 16th International*, pages 49–54. IEEE.

[7] Miles Efron. Information search and retrieval in microblogs. *J. Am. Soc. Inf. Sci. Technol.*, 62(6):996–1008, June 2011.

[8] Gideon S. Mann and David Yarowsky. Multi-field information extraction and cross-document fusion. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 483–490, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[9] Nick Dimiduk and Amandeep Khurana. *HBase in Action*. Manning.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[11] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[12] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[13] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[14] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[15] Todd Fredrich. *RESTful Service Best Practices - Recommendations for Creating Web Services*. Pearson eCollege, 2013.

[16] Aditya Agarwal, Mark Slee, and Marc Kwiatkowski. Thrift: Scalable cross-language services implementation. Technical report, Facebook, 4 2007.

[17] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[18] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[19] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[20] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*.

[21] Devashsish Thakur, Nisarga Markandaiah, and Sharan Raj D. Re optimization of id3 and c4.5 decision tree. In *Computer and Communication Technology (ICCCT), 2010 International Conference on*, pages 448–450. IEEE.

[22] Vincent Van Asch. Macro- and micro-averaged evaluation measures.

[23] Paul Rayson and Roger Garside. *The Workshop on Comparing Corpora*, chapter Comparing Corpora using Frequency Profiling. 2000.

Appendix A

Attachments

1. Webpage containing the project
2. Source code