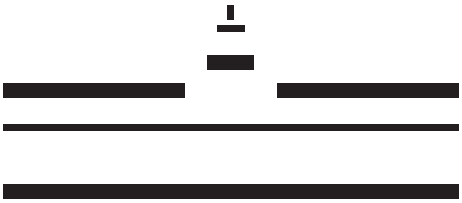

Westfälische
Wilhelms-Universität
Münster

Department of Computer Science
Computer Vision and Pattern Recognition Group

News filtering based on intrinsic user feedback and a modified ESA model

Master's Thesis

submitted by:

Karsten Jeschkies

Matriculation Number: 342058

Supervisor:

Prof. Dr. Xiaoyi Jiang

Münster, January 23, 2013

Abstract

Today's news travels within seconds across earth. While in the past it took messengers days, or a ticker minutes, today it takes just a Tweet to get a news break across the globe. The fast travel of news also allows for the gathering of a lot of news. News websites are cluttered with articles on many topics even if they focus for example on only technology related news. It becomes more confusing to stay informed on the topics of interest. There is just too much noise of information.
In this thesis I will present a news filtering system in form of a website that filters out the noisy uninteresting news and will help the consumer to focus on the news she is interested in.
The filtering is reduced to a classification problem of text documents. In this thesis I compare different classification algorithms with the text features models TF-IDF, LDA, and a modified ESA which is introduced in this thesis.
Tests on the Reuters-21578 data set show that all algorithms yield good results and that the modified ESA model performs best. Tests on user data gathered during a five months period show that simple classification cannot solve the filtering problem alone. It seems that individual user data needs to be enriched with collaborative user data.

Acknowledgements

I'd like to thank Taylor Adams, Dimitri Berh, Achim Dymek, Roman Golda, Philipp Jaquet, Romina Koers, Andreas Löcken, Torge Schiefelbein, Dimitri Wegner, and Thorben Ziemek for their creative and critical input.
I thank Xiaoyi Jiang for his advice and all members of the Computer Vision and Pattern Recognition Group for many years of inspiring lectures and seminars. Also, my thanks go out to the contributors and the community of especially the gensim library, the Scikit-learn framework, and the Topic-models mailing list for creating great Open Source software and offering their support.
Last but not least I thank my family.

Contents

1. Introduction

Today's news travels within seconds across earth. While in the past it took messengers days, or a ticker minutes, today it takes just a Tweet to get a news break across the globe. The fast travel of news also allows for the gathering of a lot of news. News websites are cluttered with articles on many topics even if they focus for example on only technology related news. It becomes more confusing to stay informed on the topics of interest. There is just too much noise of information.
In this thesis I will present a news filtering system in form of a website that filters out the noisy uninteresting news and will help the consumer to focus on the news she is interested in.
The filtering is supposed to classify articles as interesting and uninteresting news. That means I see the filtering problem as a classification problem.
Obviously, if an article is interesting or uninteresting is relative to the user reading the news. That is why I will base the filtering not only on the content of the news but also on the individual feedback given by the user. The feedback will consist only of read and unread news which means the user will not be aware she gives any feedback. That makes the feedback intrinsic as opposed to explicit feedback through for example a Like-button.
I have solved the filtering problem in three steps. At first, an algorithm is used to extract features from text which makes the text more "understandable" for a computer program. I will later give an explanation on what I mean with "understand". Then, a user model is formed from the feedback given earlier. Finally, incoming unknown news is classified as interesting or uninteresting with the help of the extracted features and based on the user model.
In the end, I will analyse the practical use of feature extraction and classification algorithms, and different user models.
The remaining part of the thesis is structured as follows: In Chapter 2, I analyse the problem and describe different methods to solve the given problem. The following Chapter 3 deals with the implementation of the algorithms and the user interface of the news filtering system. In Chapters 4 and 5, I test the system on a well-known data set and actual data collected over the course of this thesis and evaluate the results. The last Chapter 6 presents a conclusion and an outlook to future work.

2. Method

The main goal of this thesis is to filter news for a user. I see this problem as a classification problem. A program is supposed to classify incoming unknown news articles as interesting and uninteresting news depending on the user.
In this chapter I describe the main methods used to create the news filtering system. As stated earlier in the introduction, there are three major steps in filtering news articles. The first step is to convert the text to a format more understandable for the computer. This process is described in Section 2.1. The second step is to model the user's interest. What this means and how it's done are described in Section 2.2. The third and last step is to rank incoming news articles based on the trained user model. This step is covered in Section 2.3.

2.1. From Text to Features

In order to filter news, a computer program needs to "understand" the content of incoming texts. By "understand" I mean that it must be able to tell if two news articles share the same content and to what degree they are similar.
Obviously, it would not be enough to compare the byte representation of two texts, because encodings of text do not give much information about the content. Also, even if words are in a different sequence, the meaning might stay the same while the encoding will change. Comparing texts word by word does not help either, since languages let a writer express the same thought and content in many different forms. Written texts are also cluttered with stop words such as auxiliary verbs and articles. They occur in almost all texts and are not as meaningful as for example nouns. Also, homonyms and synonyms make it very hard for a computer to understand the content of a text by just comparing words as they occur.
In this section I will describe how my program pulls features beyond the occurrence of words from news articles and makes two news articles comparable. In general, a computer will learn from a collection of documents. This collection is called *corpus*. Usually, the computer learns a vector space to which unknown documents can be transformed. Documents can then be transformed from one vector space to another.

2.1.1. The Bag-of-Words Representation

The first step is the transformation a text or document like a news article from a sequence of words to a vector of numbers. Research shows that the

> *Alan Mathison Turing, OBE, FRS (23 June 1912 – 7 June 1954), was a British mathematician, logician, cryptanalyst, and computer scientist. He was highly influential in the development of computer science, giving a formalisation of the concepts of "algorithm" and "computation" with the Turing machine, which can be considered a model of a general purpose computer. Turing is widely considered to be the father of computer science and artificial intelligence.*

Figure 2.1.: The first sentence of the Wikipedia article about Alan Turing.

(4 1 1 3 1 2 2 1 1 4 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 4 1 2 1 4 1 3 2 1 1 1)

Figure 2.2.: Bag-of-Words Representation of Alan Turing quote.

arrangement of words is not too important for defining the content of a text[29]. Thus, a common approach is to associate each word of a language with an id and represent a text by a vector where each row i is the value of the frequency of term with id i in the document. The order of the ids does not change for different documents. Only the analogous values do. In that way any text will have a vector with the same dimension as any other text.

For instance, the first paragraph of the Wikipedia article for Alan Turing[1], displayed in Figure 2.1, will translate to the vector in Figure 2.2. The association between a token and its frequency can be seen in table 2.1. Note that the order has to be the same for every document.

Scanning a large corpus of documents and extracting the occurring words is a common way to teach a computer whether a word belongs to a language or not. A dictionary is built. This dictionary is filtered for the most frequent words. They are regarded as stop words and are dropped from the dictionary. The words are also stemmed or lemmatized, that means they are reduced to their lemma. For instance, "run", "running" and "ran" become "run". More advanced lemmatizing algorithms even tag the lemmas with their word class. Thus "run" is tagged as a verb, while "meeting" is tagged as a verb or noun depending on the context.

Once the process is completed, the dictionary is used to transform a text to the aforementioned Bag-of-Words vector representation. This representation is not very useful to compare the content of two news articles yet. For instance, a news article about the new Apple iPad will have a lot of technological terms and terms related to a product presentation. The word "new" will probably

[1]Wikipedia, *Alan Turing*, http://en.wikipedia.org/wiki/Alan_Turing (as of Dec. 8, 2012, 12:03 GMT).

Id	Token	Frequency	Id	Token	Frequency	Id	Token	Frequency
0	a	4	14	father	1	28	Mathison	2
1	Alan	1	15	formalisation	1	29	model	1
2	algorithm	1	16	general	1	30	of	4
3	and	3	17	giving	1	31	purpose	1
4	artificial	1	18	he	1	32	science	2
5	be	2	19	highly	1	33	scientist	1
6	British	2	20	in	1	34	the	4
7	can	1	21	influential	1	35	to	1
8	computation	1	22	intelligence	1	36	Turing	3
9	computer	4	23	is	1	37	was	2
10	concepts	1	24	June	2	38	which	1
11	considered	2	25	logician	1	39	widely	1
12	cryptanalyst	1	26	machine	1	40	with	1
13	development	1	27	mathematician	1			

Table 2.1.: Tokens and their Frequencies of Alan Turing Paragraph.

have a high frequency. However, "new" is not really meaningful even though it has such a high weight. This can be seen at the Bag-of-Words vector for the Alan Turing sample. The words "a" and "of" occur 4 times. However, they alone do not convey as much meaning as for example "scientist". The vector needs to be transformed again.

2.1.2. Term Frequency - Inverse Document Frequency

The previous section describes how a document, an arrangement of words, is transformed to a vector representation. So far this representation includes just frequencies of words as weights and maybe the word class. However, frequency alone is not a meaningful indicator for the similarity or dissimilarity of texts. As the Turing excerpt shows, words with not a lot of meaning such as "a" and "of" have a high frequency. It seems frequency is not a good weight to measure the importance of a word for the meaning of a text or document.
We need a representation that weighs meaningful words higher than less meaningful words. Picking up the new Apple iPad example we want "iPad" and "tablet" to weigh higher than "new" and "iPad" higher than "tablet" because they tell more about the content of the text.
One of the first and most used algorithms, that yield such a weighing is the Term Frequency - Inverse Document Frequency Algorithm by Salton and McGill[29].

The Term Frequency - Inverse Document Frequency, short TF-IDF, value is calculated by function $tfidf(t, d, D)$ defined in Equation 2.1.

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{2.1}$$

There are a variety of definitions for $idf(t, D)$ and $tf(t, d)$.
Equation 2.2 shows a common definition of $idf(t, D)$ to calculate the inverse document frequency. The number of documents of the learning corpus D that include the term t is given by $|\{d \in D : t \in d\}|$. The total number of documents is given by $|D|$.

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{2.2}$$

Function $tf(t, d)$ just gives the count of term t in document d.
The resulting vector is usually converted to unit length.
The definition 2.1 clearly shows the idea to weigh words or terms — I use both interchangeably — high that have a high frequency in the document and low frequency in the corpus. Thinking about the "iPad" example the word "new" has both a high term frequency and a high document frequency and thus a lower TF-IDF weight than "tablet" which has also a high term frequency but probably a much smaller document frequency. Obviously, the corpus frequency depends heavily on the corpus.
The TF-IDF algorithm is fairly easy to understand and implement and also performs quite well as can be seen in Chapter 4. That is why I chose it for a prototype of the news filtering engine.
However, the algorithm cannot detect homonyms and synonyms very well. Even with a very good lemmatizer it cannot distinguish between short documents about the keys for my room and a document describing the keys of the piano in my room. Two example documents are given in Appendix A. A more advanced algorithm is needed to teach a computer to make this distinction when understanding and comparing news articles.
Also, TF-IDF cannot distinguish between texts and meta-text. For instance an algorithm based on TF-IDF would not be able to distinguish between a text from the book "Lord of the Rings" and an article about the book.

2.1.3. Topic Modelling with Latent Dirichlet Allocation

So far I've presented methods to transform a text to a vector format based on TF-IDF. Even though this format works quite well, it cannot distinguish between for example homonyms and synonyms. In this section I will present Latent Dirichlet Allocation, LDA, which was introduced by Blei et al.[4] and is capable of doing just that and also reducing the dimension of the features

dramatically.
LDA can be used for clustering, collaborative filtering and text feature generation. It is related to Latent Semantic Analysis[10], also called Latent Semantic Indexing, and Probabilistic Semantic Analysis[19]. The idea is to describe a group with a lower dimensional representation of its data. In the case of this thesis a document is seen as group of words.
The authors of LDA introduced topics. Topics are distributions over a fixed vocabulary — every word. Groups, or rather documents, are then seen as distributions over topics. Thus topics are seen as a hidden, also called latent, structures of documents. Their relation to words and documents is best illustrated with the plate notation. I will now give a quick introduction to plate notations.

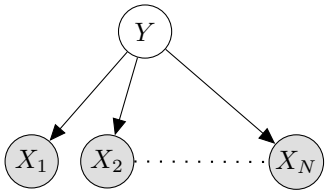

Figure 2.3.: Graphical model

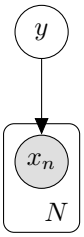

Figure 2.4.: Plate notation

In Figure 2.3, the dependency of the observed variables $X_1, \ldots, X_N$ on the latent variable Y are illustrated with a simple graph. Note that observed variables are placed against a grey background. The same relation is expressed in Figure 2.4. Both graphs are equivalent to Equation 2.3.

$$p(y, x_1, x_2, \ldots, x_n) = p(y) \prod_{n=1}^{N} p(x_n|y) \tag{2.3}$$

The plate notation of the dependency of the latent and the observed variables of LDA are illustrated in Figure 2.5. The variables α and β are Dirichlet distribution parameters. The variable $Z_{d,n}$ assigns observed word $W_{d,n}$ to topic ϕ_k, which is a distribution over words as said before. Variable θ_d is the document-specific topic distribution. Both distributions θ_d and ϕ_k are drawn from Dirichlet distributions. A Dirichlet distribution is — in plainly terms — a distribution of distributions.

Imagine we have the following dictionary: “iPad”, “Apple”, “company”, “new”, “product”, “tree”, “fruit”. The algorithm has learned the following three topics shown in Table 2.2. The topics are distributions across the dictionary. The first topic is somewhat associated with Apple the company, and the iPad. Topic #2 is related to new products. The last topic has a high probability for words related to fruits.
If we again have a document about the presentation of a new iPad the distribution across the topics might look like this: Topic #1 0.5, Topic #2 0.4,

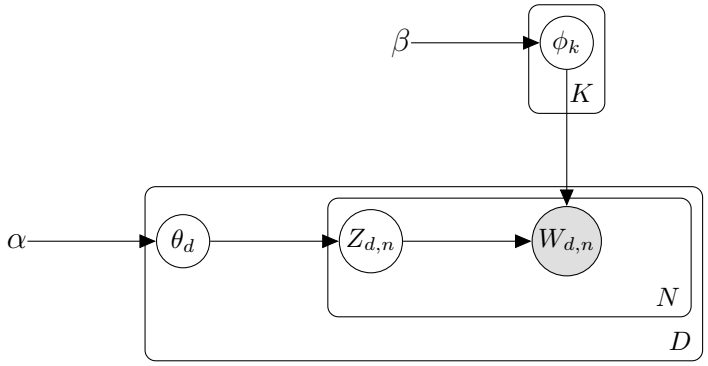

Figure 2.5.: Plate notation of the Latent Dirichlet Allocation model.

Topic #3 0.01. The feature vector will be (0.5 0.4 0.01). The feature vector for a text about apple the fruit might look like this: (0.1 0.001 0.4). So the algorithm is able to distinguish between Apple[2] the company and the fruit.
Over the years since the introduction of LDA several interference algorithms were introduced to determine the distributions by unsupervised learning. Estimation propagation, variational inference and different MCMC (Markov Chain Monte Carlo) algorithms such as Gibbs samplers can be used. Some are simpler to implement, while others allow online or parallel learning or yield better results.

Topic #	Members
1	iPad 0.5, Apple 0.3, company 0.3, new 0.1, product 0.2, tree 0.01, fruit 0.01
2	iPad 0.02, Apple 0.001, company 0.1, new 0.5, product 0.4, tree 0.001, fruit 0.001
3	iPad 0.001, Apple 0.3, company 0.001, new 0.001, product 0.001, tree 0.5, fruit 0.6

Table 2.2.: Latent Dirichlet Allocation example Topics and their Members.

2.1.4. The Concept of Explicit Semantic Analysis

Explicit Semantic Analysis, short ESA, was introduced in 2007 by Evgeniy Gabrilovich et al.[14]. They took a slightly different approach from LDA to help a computer recognize homonyms and synonyms.
The idea is to describe a text not with topics learned from a corpus but with concepts represented by articles from a structured corpus such as Wikipedia. The text of a document is transformed to a vector where each row shows the

[2]Read “Apple” and “apple”. Capitalization is neglected because it is not a reliable indicator for meaning.

similarity of the document to a corresponding concept. It is essential that the corpus is structured in some form.
Taking the new iPad example the news article would have a high similarity with the concepts "Apple Company", "iPad", "tablet" and their Wikipedia articles respectively and a low similarity with for example "Tree" and "Fruit". So the computer is able to understand that the news article is not about apple the fruit but about the company. In the following I will present the learning steps.
At first, each concept from the structured corpus has to be filtered for relevance. The English Wikipedia has more than 4 million articles by now[3], including redirecting and disambiguation pages, which will be ignored. Using the remaining articles as concepts would not only yield a very big feature vector but also create too much noise because a lot of small and negligible articles would "overshadow" meaningful ones. Gabrilovich et al. filtered articles with a word count threshold. Articles which have a word count of less than 500 are dropped. In the next step articles are ranked by their incoming intra-Wikipedia links. Wikipedia articles reference other articles. The article "iPad" e.g. links to "Apple Company" raising the incoming links for "Apple Company" by one. All articles with less than 5 or 10 incoming links are dropped. The thresholds are estimations by the authors. Gabrilovich et al. ended up with 241,393 articles of 1,187,839 articles of an snapshot of the English Wikipedia from March 26, 2006.
Each article is then transformed to TF-IDF space. They form the interpretation matrix shown in Equation 2.4 and eventually the new ESA space. The matrix consists of N concepts with M features each.

$$I_{m,n} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M,1} & c_{M,2} & \cdots & c_{M,N} \end{pmatrix} \tag{2.4}$$

In order to transform a news article from text to ESA space the following steps need to be taken. First, the article is transformed to a Bag-of-Words vector and then to TF-IDF space which was presented in Section 2.1.2. Finally, the vector is multiplied with each concept vector of the interpretation matrix. Let v_i be the weight of the news article vector in TF-IDF space and r the resulting vector in ESA space.

$$r_j = \sum_{i=1}^{M} v_i \cdot c_{i,j} \quad j \in \{1, \ldots, N\} \tag{2.5}$$

[3]November 2012.

The transformation is shown in Equation 2.5. The weight or rather similarity for the news article with each concept c_j is given by r_j.
All calculations can be done in one step by multiplying v with I.
The advantage of ESA over LDA is that the algorithm is simpler to understand and the results are easier to be interpreted for humans.

2.1.5. Modified ESA

In the previous section I described the ESA model and how a document in TF-IDF space can be transformed to ESA space. Even though the ESA model has some advantages over the LDA model, it has some major disadvantages. Using articles from Wikipedia as concepts will yield a very high number of concepts and thus dimensions. Even if the articles are filtered the number of dimension may lie beyond 1.5 million. Taking into account that each concept itself has around 50.000 dimensions in TF-IDF, space we end up with a 1.5 million times 50.000 matrix. Suppose a weight is a float taking up 32 bit, the matrix will be 600GB[4] huge. That is too much for most computers to handle.
In this section I propose a modified Explicit Semantic Analysis model, short mESA, which tackles this problem.
The matrix can be reduced by reducing the number of concepts used. For example, the Wikipedia articles can be filtered more rigorously. Gabrilovich proposed to filter articles by a PageRank like link ranking[15]. This way, fewer articles are chosen. However, it is important to choose representative articles or substantial information might be lost. Also, the resulting number cannot be set.
I propose another solution. Instead of building a big matrix with TF-IDF weights, a similarity index with fewer but more significant concepts is built with help of a training corpus. The training corpus consists of documents in TF-IDF space and each document is already classified.
First, the cosine similarity of each document d from the training corpus with each concept c_i is calculated. The cosine similarity is just the cosine of the angle between the two feature vectors in unit length. If it is close to 1, the two articles are very similar in content, and if it is closer to -1 they hardly similar at all.
The similarity is used as the weight for a new feature vector v. Its dimension equals the number of concepts. Then, feature selection is done on the training articles using a ranking algorithm for features. Only the top k meaningful features are kept. Once the top features are determined, all corresponding concepts are compiled in a new similarity index. To transform a new document N to mESA space, the cosine similarity of N to each concept c_i in the new similarity index is calculated and then used as the weight for dimension i of the resulting feature vector.

[4]With 1GB = 1.000.000.000 bytes

2.2. Modelling the User

So far I have described in this chapter how to make text for a computer comparable and thus to a certain degree more understandable. I showed ways how a computer can analyze incoming news articles. In this section I will describe how a computer program can decide if an incoming news article is of interest for a user of the news filtering system or not.
To make a decision, the computer needs a model of the user's interests. Somehow the computer needs to learn what is of interest and what is not. The computer needs feedback to learn such a model. As mentioned in the introduction, I decided for individual intrinsic feedback. That means the user does not know that she gives any feedback as opposed to explicit feedback through for example a Like-button when she is aware that she gives feedback.
Also, only her feedback is used for her user model exclusively and no feedback by others is used, as it would be in a collaborative recommendation system.
The problem can be understood as a classification problem. Incoming news articles need to be classified as read/interesting or unread/uninteresting. Thus the user model is just a classifier.
In the stages of the news filtering system I used different user models. I will describe them in the following subsections.

2.2.1. Centroid Based Model

In the early stages of my system I used a centroid based model as mentioned by Han and Karypis[16]. At its core it is a very simple Rocchio classification algorithm to classify new articles as read or unread articles.
If a user reads an article it is marked as read. This is the only feedback needed. To create the centroid based user model a centroid is calculated for all read articles. That means all feature vectors of the read articles are summed up and divided by the total number of read news articles. Equation 2.6 shows the equation. The resulting vector $\vec{C}$ is the centroid of the read articles S.

$$\vec{C} = \frac{1}{|S|} \sum_{d \in S} \vec{d} \tag{2.6}$$

To classify an incoming unknown news article as interesting or not it will be classified as read or unread. This is done by calculating the cosine similarity of the centroid and the incoming article which was transformed to the desired space such as TF-IDF, LDA, or mESA.

2.2.2. Naïve Bayes and SVM Model

The Naïve Bayes Model is, as the name suggests, based on the Bayes classification algorithms. A Naïve Bayes is trained with previously read and unread

articles to be able to classify new articles as read or unread.
In the same way a Support Vector Machine, short SVM, is trained to classify incoming articles as read or unread, interesting or uninteresting respectively and used as the SVM model.

2.2.3. SMOTE and Borderline-SMOTE

The training data will be imbalanced. Users will most likely read only a few of all incoming articles. In the course of this thesis I have received about 14,000 news articles but read only 192 within five months. The unread articles build a majority class and the read articles a minority class. A classifier that classifies every element to be of the majority class will have a very low error rate. As a consequence no item will be classified to be of the minority class. That means such as classifier will not filter out any interesting news article since all articles are regarded as uninteresting. That is quite the opposite of what a filter should do.
There are several techniques to combat such behavior. For starters, I implemented normalization. The data is transformed so that the mean is at the origin and the standard deviation is one.
There are more specialized algorithms[18] to deal with imbalanced data. I decided for the SMOTE algorithm by Chawla et al.[7] and the SMOTE enhancement Borderline-SMOTE by Han et al.[17]. Both are fairly easy to implement and use over-sampling of the minority class with synthetic samples and under-sampling of the majority class.

2.3. Filtering Incoming News Articles

Once a user model is learned, the ranking of new news articles is quite simple. At first, the incoming news article is transformed to a probable space such as the mESA space. It then is classified as read or unread or rather interesting or not interesting.
The centroid based model returns just a likelihood. To definitively decide whether the article is of interest or not, the likelihood has to be over a certain score which needs to be estimated. Obviously, a learning algorithm could be applied to find the perfect threshold. Another way is to create a centroid for both classes and decide whether the incoming article is closer to the interesting or uninteresting class.
The Bayes and SVM model give a definite classification. Either an incoming article is classified as read/interesting or unread/uninteresting. In its core the Bayes model is a Bayes classifier and thus calculates likelihoods of an article to belong to one class or another. Research has shown that this likelihood is useful for classification but a bad estimation[33]. Thus, it should not be used for comparable ranking.

3. Implementation

This chapter covers the user experience of the front end in Section 3.1. Section 3.2 will cover the methods to transform text to TF-IDF and LDA space using the gensim Python library. Section 3.3 focuses on the creation of the mESA model. After that I will give some details on the user model learning algorithms in Section 3.4. The last Section 3.5 is about the structure of the news filtering system.

3.1. Front End

This section will cover step by step the user experience, short UX, of the news filtering system.
In the past the term user interface, short UI, was used to describe how a user interacts with a computer program. Today the term user experience is more common. It emphasizes that the usage of a program is more than clicking buttons — it is in fact an experience.
The main focus for the website was that the user should have a pleasant reading experience and as little trouble as possible in handling the program.
The front end is a minimalistic website and was programmed in Python with the small web framework Flask[1]. Apache[2] is used as the HTTP-Server.
The front end does not start any crawling, training, or ranking actions. It just presents ranked articles and their content. Also, it lets the user subscribe to and unsubscribe from news outlets.
It draws heavily from the HTML5[20] and CSS3[11] standards. The CSS framework Twitter Bootstrap[3] is used to create a good looking cross browser compatible site.
Great care was taken in presenting the news articles overview and articles in reading mode. The web font Open Sans is used as the main typeface. Open Sans was designed to be easily readable on screen and to boost reading speed[4]. Research has shown that typefaces have a great impact on reading experience[1]. Therefore care was taken to deliver such experience.

[1] http://flask.pocoo.org/
[2] http://httpd.apache.org/
[3] http://twitter.github.com/bootstrap/
[4] From the typeface project site: "It was optimized for print, web, and mobile interfaces, and has excellent legibility characteristics in its letterforms.". See: http://www.google.com/webfonts/specimen/Open+Sans

3.1.1. "Top News" Tab

When the user logs in, she will discover five tabs to the left hand side as seen in Figure 3.1. The tab *"Top News"* is highlighted on login. So the first thing displayed to her are the filtered top articles of the day.
The main space is taken by a list of news articles. They are presented in three columns. The titles of the news articles are in bold. The first 140 characters of the news article are displayed underneath to give a short introduction to the article. A click on a headline will direct to the reading view of the corresponding article. The user will not be directed to the news source.
The reading view is described in Section 3.1.5.
At the top of the page there are two buttons which let the user navigate to news of different days.
If there is no news for the day or the user has no subscriptions yet, an appropriate message is displayed.

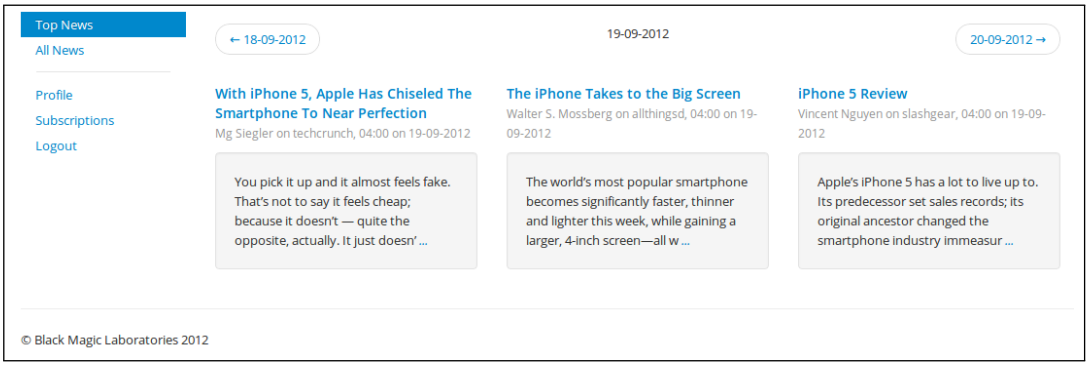

Figure 3.1.: Top News View.

3.1.2. "All News" Tab

The tab *"All News"* will, as the name suggests, display a list of all the news of the day. The articles are grouped by their news outlets. Articles which have already been read are hidden and can be displayed by clicking on "Show Read Articles".
On the right hand side a list of the news outlets is displayed. By clicking on the name the website will scroll to the corresponding news outlet section in the list of all news articles. Note that both navigation lists are fixed and do not scroll as seen in Figure 3.2. This will allow for easy navigation among a long list of articles.

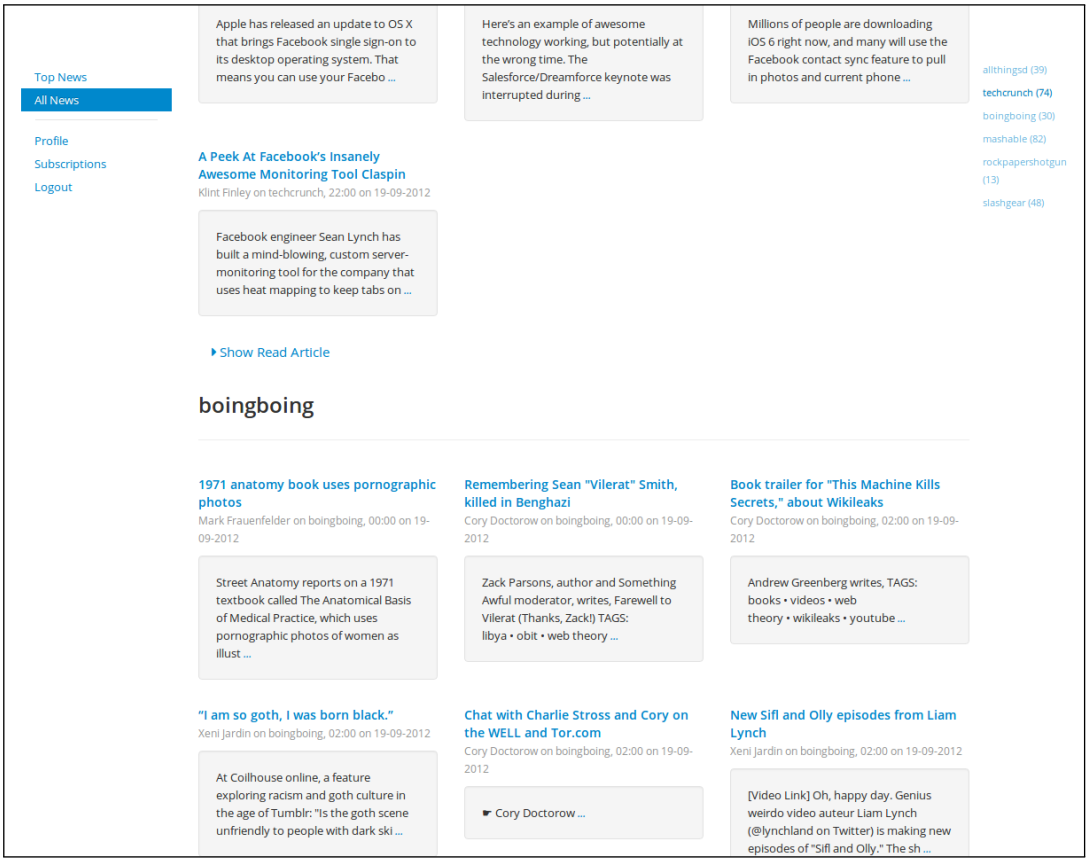

Figure 3.2.: All News View.

3.1.3. "Subscriptions" Tab

The tab *"Subscriptions"* will display a list of all news outlets supported. A small part is shown in 3.3. A news outlet is a website that publishes news articles on a regular basis such as blogs or news sites. So far 24 English technology blogs and news sites are supported.
The user can easily subscribe or unsubscribe by clicking the corresponding buttons. Even though the news filter system uses the RSS feed format internally the user cannot add new news feeds by herself.

Figure 3.3.: Subscriptions Management.

3.1.4. "Profile" Tab

The tab *"Profile"* will display an option to change the user's password. To ensure an easy and fast experience a click on the save button will not reload the page but invoke an AJAX request to the server. When the password was saved successfully a small pop-in at the top will inform the user as can be seen in Figure 3.4.

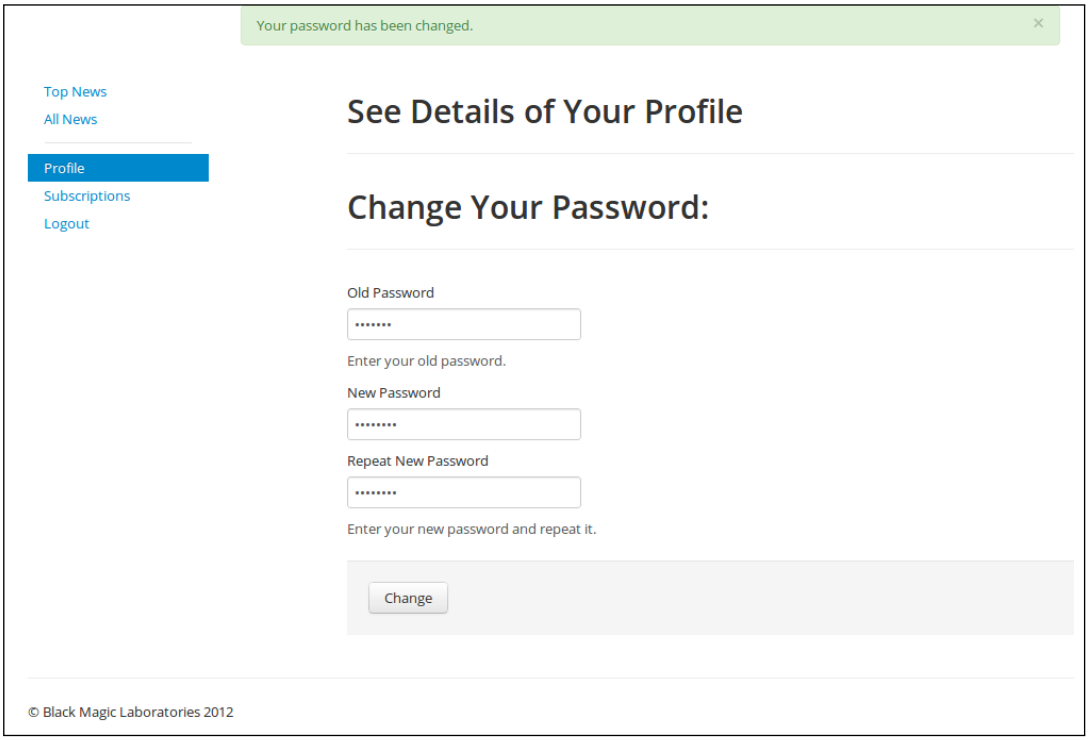

Figure 3.4.: Profile view after changing the user's password.

3.1.5. Reading View

The reading view is accessed by clicking on the headline of an article. Figure 3.5 shows the view. The headline at the top is a link to the original news article. Underneath, the author's name and news outlet's name are displayed next to the publishing date.

Figure 3.5.: Reading View

The content of the article which is scraped from the news site[5] is displayed in a slightly bigger font size with generous spacing to ensure easy reading. So far no pictures and videos are displayed. This is part of future work.
The view is designed for better legibility. In my opinion good presentation is as important as good algorithm for a news filtering system. The best filter algorithms would not be accepted as much if articles were to be presented in small sized typefaces and tight lining. However, designing a legible article pages is subject to designers and related research such as by Beymer et al.[1].

3.2. Transforming Text to TF-IDF and LDA Space

In this section I will briefly describe the implementation for transforming text to TF-IDF and LDA space.
For the transformation of text to a Bag-of-Words representation, TF-IDF and LDA space I used the Python library gesnim by Rehurek[28] which makes heavy use of the Python extensions SciPy[22] and Numpy[26]. Gensim was chosen for mainly for two reasons. On one hand it has an implementation for the most common text feature models and its design allows for an easy extension with new feature models. On another hand it is capable of learning from and transforming big data because it never loads the complete data into

[5]The scraping is done purely for academic purposes.

memory. It is memory independent.
The concepts of gensim are the corpus, vectors and models. A corpus represents a collection of documents. Vectors are simply documents transformed to feature vectors. Models offer transformations between two different document representations. For instance, the LDA model transforms any feature vector to LDA space.
The gensim interface becomes most apparent with the example in Listing 1.

```
1  corpus = CleanCorpus(options.doc_path)
2
3  #create dictionary
4  dictionary = corpora.Dictionary(corpus.get_texts())
5  dictionary.filter_extremes(no_below=20,
6                             no_above=0.1,
7                             keep_n=50000)
8
9  #save dictionary: word <-> token id map
10 dictionary.save(options.prefix + "_wordids.dict")
```

Listing 1: Creating a dictionary in gensim.

At first, a dictionary needs to be created. Gensim expects the corpus to be an iterable over lists of tokens. Each list of tokens represents a document. The function *filter_extremes()* filters out all tokens which appear in more than 1% of all documents and less than 20 times. Of the filtered tokens 50,000 tokens are kept. In this way stop words are automatically filtered out since the main characteristic of a stop word is that it appears in almost all documents.
The function *get_texts()* yields a list of tokens for each document which is lemmatized on the fly. Yielding allows the program to process one document at a time without loading all of them into memory.
Gensim uses library called *pattern*[8] for lemmatization. *Pattern* implements two lemmatization algorithms[5] which can also tag the word with its word class.
The following Listing 2 illustrates the creation of a model. A corpus which was transformed to a specific vector space — Bag-of-Words in this case — and a dictionary is passed to the model which then learns on the corpus. Again, the implementation does not load the complete corpus.

```
#init corpus reader and word -> id map
id2token = corpora.Dictionary.load(options.prefix + "_wordids.dict")
mm_bow = corpora.MmCorpus(options.prefix + '_bow_corpus.mm')

'''TFIDF Model creation'''

#build tfidf model
tfidf = models.TfidfModel(mm_bow, id2word=dictionary, normalize=True)

#save tfidf model
tfidf.save(options.prefix + '_tfidf.model')

#save corpus as tfidf vectors in matrix market format
corpora.MmCorpus.serialize(options.prefix + '_tfidf_corpus.mm',
                           tfidf[mm_bow],
                           progress_cnt=10000)
```

Listing 2: Creating a model in gensim.

3.3. Creating the Modified ESA Model

This section describes the implementation and creation of the mESA model. At first, I will describe how the program masters the big amount of data provided by the English Wikipedia encyclopedia. I will then give a brief overview over the feature selection.

3.3.1. Handling the Big Data of the English Wikipedia

The openness of Wikipedia makes it easy to obtain the 8.8GB[6] compressed snapshot of the English version. The uncompressed snapshot is a well over 32GB large XML file. The XML contains the text of each Wikipedia page in the Wikipedia markup language, different kinds of references among the pages, and external links. There is no media data included.
I took several steps to filter the articles to be used for the learning of the modified ESA model.
At first, I used the Java Wikipedia Library[32], JWPL, to preprocess the snapshot. During the preprocessing the articles are parsed to extract links and to remove any markup and references. Everything is saved in a MySQL[7] database. I created a small Java program which extracts all Wikipedia pages that are not redirects or disambiguation pages, that had more than 500 characters and more than 10 incoming intra-Wikipedia references. Each article was saved to

[6] As of November 2nd, 2012
[7] http://www.mysql.com/

a text file with the article's name as the first line. Saving each article to its own text file and having all 1.8 million files in one folder turned out to be troublesome for Microsoft Windows and Linux file explorers and the NTFS and ext4 file systems. A database should be used in future work.
The text files formed the corpus for the TF-IDF and LDA model training mentioned above and the modified ESA model.
While a lot of papers present algorithms which perform very well on small data sets, they often do not on big data sets because they demand to have all data in RAM. In those cases I had to implement incremental variations of the algorithms to have them memory independent. In some cases I used subsampling to reduce the data sets to a size suitable for in-memory storage.
In cases when a high demand of RAM was unavoidable I used virtual machines with variable system resources such as EC2 instances of the Amazon Web Service[8].

3.3.2. The Modified ESA Model in Gensim

The modified ESA model requires a corpus and a training classification to select features.
I modified the *SelectKBest* class of the Scikit-learn framework[27] to be memory independent and thus be able to handle data without extensive usage of RAM.
The implementation uses the Anova F-Value to sort features. The F-Value is the "ratio of between-group variance to the within-group variance, which represents the classification capability of the feature"[23]. The top k features are selected. In this case k was set to 1000.

3.4. User Model Learning Algorithms

Scikit-learn and its classifier implementations of the Naïve Bayes and SVM were used for the corresponding user models. The Bayes classifier assumes a Gaussian distribution for each feature to work with continuous data.
A custom implementation of the centroid classifier was used. The interface is equal to the classifier interface of Scikit-learn as shown in Listing 3. The method *fit(X,y)* trains the classifier with training data X and the target classes y. The method *predict(X)* returns the predicted classes for data input X. The complete implementation is shown in Appendix B. It illustrates how the input X and y should be handled according to the interface.
The interface allows the classifiers to be exchanged easily.
User models are saved via pickling[9] in the database.

[8] http://aws.amazon.com/
[9] "Pickling" is the Python term for "marshalling". See `http://docs.python.org/2/library/pickle.html` for details.

```
class CentroidClassifier(object):

    def fit(self, X, y):
        '''
        Parameters
        ----------
        """Fit Centroid classifier according to X, y

        Parameters
        ----------
        X : array-like, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of samples
            and n_features is the number of features.

        y : array-like, shape = [n_samples]
            Target values.
        '''
        pass

    def predict(self, X):
        """
        Perform classification on an array of test vectors X.

        Parameters
        ----------
        X : array-like, shape = [n_samples, n_features]

        Returns
        -------
        C : array, shape = [n_samples]
            Predicted target values for X
        """
        pass
```

Listing 3: Scitkit-learn interface for the Centroid classifier.

3.5. Architecture

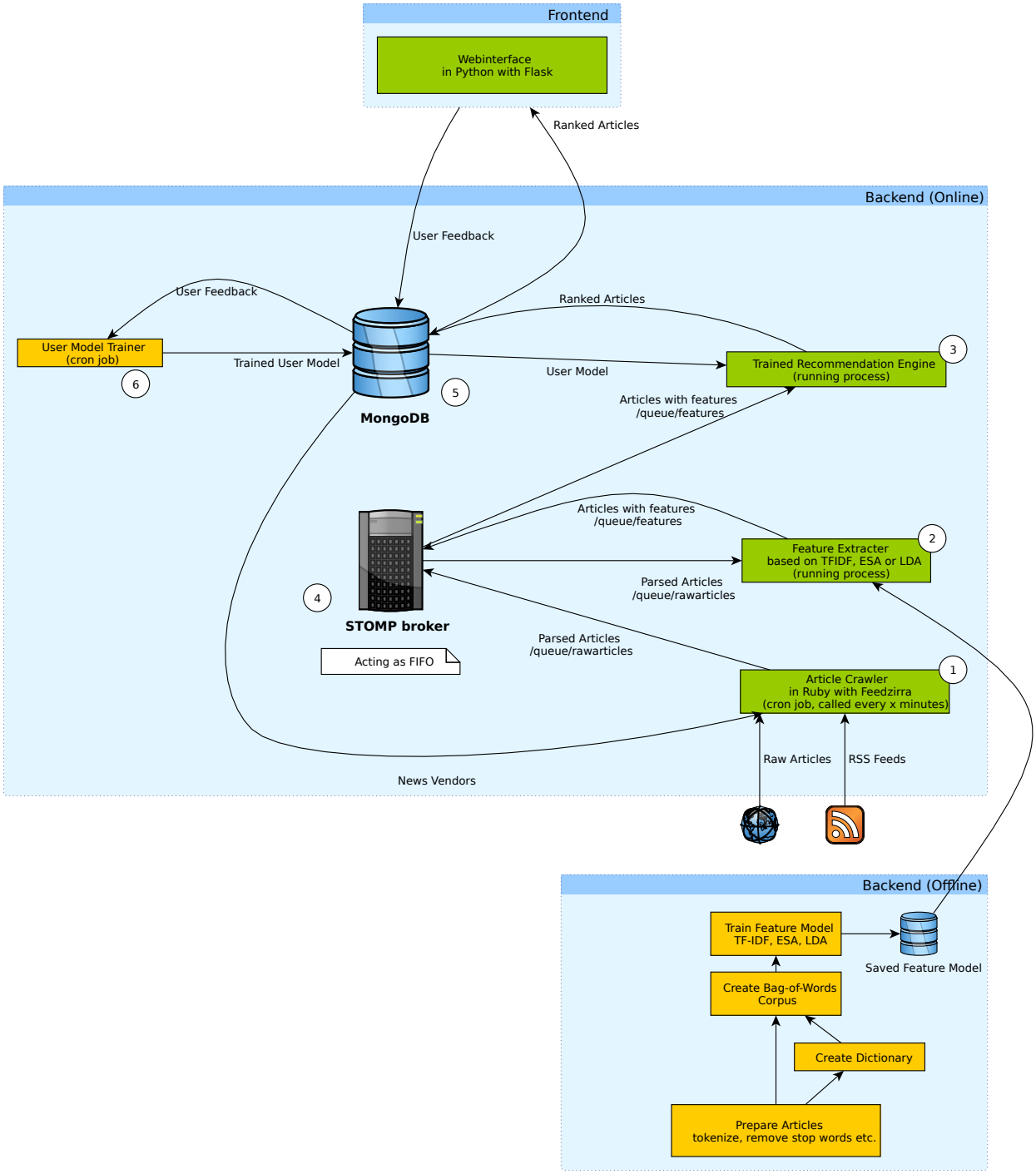

Figure 3.6.: Server Architecture.

So far I covered the implementation of the algorithms and the interface of the front end. In this section I will talk about the backend architecture employed.

That means I will give some details how the news filtering system is set up and on how the implemented algorithms work with each other.

3.5.1. The Data Model

Originally, a MySQL database was used to save user credentials, news articles and so forth. However, the rigid setup of a relational database was not suitable for the frequent changing data structures during development. That is why I decided to use the non-relational, or NoSQL, database MongoDB[10] which has dynamic schemas. Figure 3.7 shows the data models in JSON notation and their relations. MongoDB does not support relations and joins directly, but one can still define ids as references to other models. For instance, RankedArticle has a user ID and an article ID. So every RankedArticle instance references to one User and one Article. Joins have to be performed "manually" in the client of the database.
The structure is quite simple. Every User instance has a reference to its UserModel, e.g. a pickled Bayes model, and references to the news outlets subscribed. When a user reads an article a ReadArticleFeedback with a reference to the user and the article is saved. All incoming articles are ranked for each user and the ranking is saved in RankedArticle.

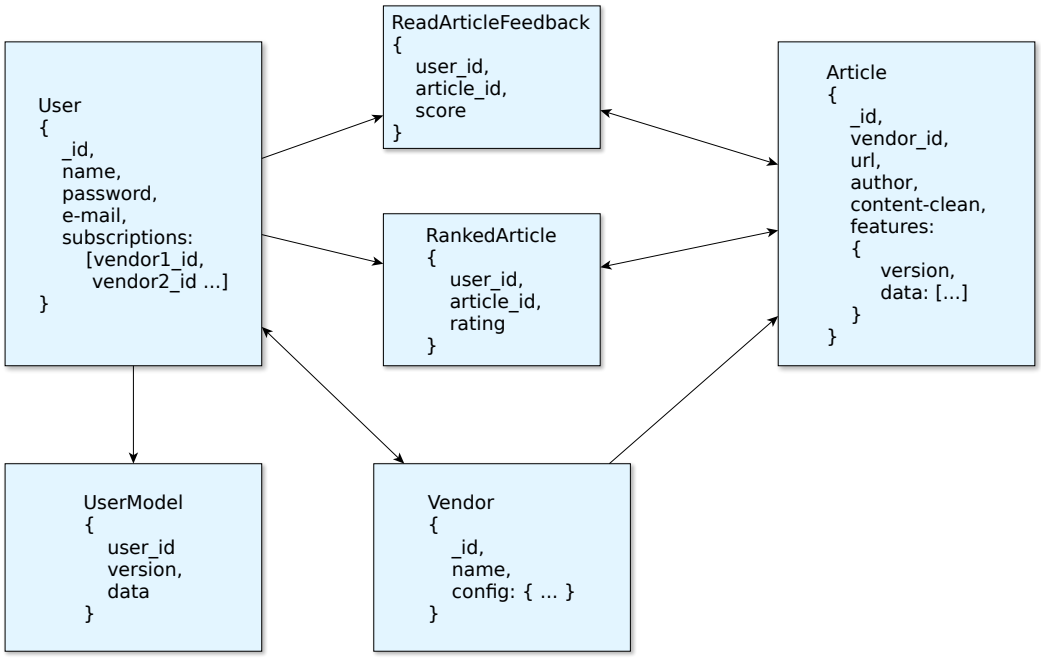

Figure 3.7.: JSON notation of the data models and their relations.

[10] `http://www.mongodb.org/`

3.5.2. The Crawler

The crawler is a standalone-program and written in Ruby[30]. For each news outlet it crawls every two hours the associated RSS-Feeds for new articles. New articles are downloaded and their author, headline, content, and URL are send to a message query.
The crawler is marked with number *1* in Figure 3.6.

3.5.3. The Feature Extractor

The feature extractor (number *2* in Figure 3.6) is a daemon program written in Python that reads incoming articles from a message queue and transforms the articles' texts to the modified ESA space. It then sends the article and its feature vector to another message queue.

3.5.4. The Article Ranker

The article ranker (number *3* in Figure 3.6) is a daemon program in Python that receives transformed articles from the feature extractor via a message queue and ranks the articles with one of the ranking algorithms described above. It then saves the articles, their features, and their ranking to the MongoDB database.

3.5.5. The User Model Trainer

The user model trainer (number *6* in Figure 3.6) is a program written in Python that is started every other day to update the user models of the registered users. It uses one of the implemented models described above.
The learned models are saved to the database.

3.5.6. Tying Everything Together

All parts of the news filtering system run in their own programs to provide great flexibility. They communicate either by saving to and reading from the MongoDB database (number *5* in Figure 3.6) or the Streaming Text Oriented Messaging Protocol[11], short STOMP, message queue protocol.
For instance, the article ranker saves incoming articles to the database. The front end reads articles from the database when a users wants to view them. In this way they work independently and can be on different machines.
The crawler, feature extractor and article ranker send and read messages through STOMP message queues. CoilMQ[12] is used as the message queue

[11] `http://stomp.github.com/`
[12] `https://github.com/hozn/coilmq/`

server also called broker (number *5* in Figure 3.6). All programs can be on different machines. There can be even many instances of the same programs working parallel. This provides a great and robust scalability. Figure 3.6 shows the whole setup.

4. Results

This chapter presents the effectiveness of the classifiers in combination with different feature spaces.
Section 4.1 introduces the metrics used to compare the results. After that, Section 4.2 presents the results for tests on the Reuters-21578 data set and Section 4.3 the results for actual data.
The results are discussed in Chapter 5.

4.1. Precision, Recall, and F-Measure

This section is about the metrics to measure the performance of the news article recommendation system.
Calculating just the error rates of the classifier and using them as performance metrics for the whole system would not lead to meaningful data. My assumption is that there are many more uninteresting articles than interesting ones by roughly a ratio of 100 to 1 respectively. A classifier which just declares all articles as uninteresting will have an error rate of less than 1%. Thus, a different metric is needed.
What I am interested in is the performance of the system in regard to the actual and predicted interesting articles. A good classifier and thus a good recommendation system will detect almost all actual interesting articles, called True Positive (TP in Table 4.1), and have very few uninteresting articles predicted as interesting, called False Positive (FP in Table 4.1).

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 4.1.: Confusion Matrix.

The Precision is the percentage of actual interesting articles of all predicted as interesting articles. Recall is the percentage of actual interesting articles which where classified as interesting. They are calculated as defined in Equation (4.1).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1a}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.1b}$$

Since both metrics have to be evaluated, a single score was introduced by Lews and Gale[24]. It is called F-measure and is defined in Equation 4.2. It ranges between 0 and 1.

$$\mathrm{F}_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.2}$$

4.2. Tests on Reuters-21578

Before I present my experiments and an evaluation of the news filtering system I will present an evaluation of the classifiers in combination with various feature models and settings, and a well known data set.
I used the Reuters-21578 dataset for training and testing because it is well studied and came to be one of the de facto standard datasets for natural language processing tasks. The dataset consists of short news wires by Reuters from 1987. Each news wire is considered a document and was assigned to specific topics. There are documents which are assigned to no topics at all and others to more than one topic.
Despite being a somewhat standard dataset concerning natural language processing and text classification, many scholars use different subsets, making comparisons between results from different authors difficult. I used the R8 subset which is based off of the R10 subset as described and proposed as a standard subset by Debole and Sebastiani[9]. The R8 subset has the following definition: The classes with the highest positive training samples from the "modApté" training/test split as described by David Lewis in the README[1] of the dataset are chosen. Only the documents with a single topic and the classes which still have at least one training and one test example are considered. This leaves us with 8 classes, hence the name R8. The classes are named after their topics names: *acq*, *crude*, *earn*, *grain*, *interest*, *money-fx*, *ship*, and *trade*.
I tested the SVM, Bayes, and centroid based classifiers on documents in TF-IDF space, LDA space with 20 and 50 topics trained on the corpus in TF-IDF space, LDA with 20, 30 and 50 topics trained on the corpus in Bag-of-Words space, and in modified ESA space. The TF-IDF space has a dictionary with 2894 tokens and thus 2894 dimensions. The modified ESA space was reduced to 1000 concepts and thus has 1000 dimensions. All LDA models were trained with 4 passes.

[1] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

4.2.1. Results for Balanced data

At first, I picked the documents from two classes with roughly the same number of training and test documents. The classifiers were trained and tested on these documents. All results can be seen in C.1 in the appendix. Table 4.2 shows the result for the documents of topics *crude* and *interest*. The F1 scores are highlighted in grey. The best results for each classifier are in bold type. The average column shows the weighted average of the F1 scores. The numbers following the topic/class names indicate training size and test size.

		crude (389/189)			*interest* (347/131)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.56769	0.68783	0.62201	0.35165	0.24427	0.28829	0.48539
	LDA-20	0.77193	0.69841	0.73333	0.61745	0.70229	0.65714	0.70214
	LDA-50	0.84971	0.77778	0.81215	0.71429	0.80153	0.75540	0.78892
	LDA-BOW-20	0.90419	0.79894	0.84831	0.75163	0.87786	0.80986	0.83257
	LDA-BOW-30	0.77619	0.86243	0.81704	0.76364	0.64122	0.69710	0.76794
	LDA-BOW-50	0.77232	0.91534	0.83777	0.83333	0.61069	0.70485	0.78336
	mESA-1000	1.00000	0.84656	**0.91691**	0.81875	1.00000	**0.90034**	0.91013
Centroid	TFIDF	0.57277	0.64550	0.60697	0.37383	0.30534	0.33613	0.49609
	LDA-20	0.76705	0.71429	0.73973	0.62500	0.68702	0.65455	0.70486
	LDA-50	0.73729	0.92063	0.81882	0.82143	0.52672	0.64186	0.74638
	LDA-BOW-20	0.78070	0.94180	0.85372	0.88043	0.61832	0.72646	0.80162
	LDA-BOW-30	0.76126	0.89418	0.82238	0.79592	0.59542	0.68122	0.76460
	LDA-BOW-50	0.78667	0.93651	0.85507	0.87368	0.63359	0.73451	0.80572
	mESA-1000	1.00000	0.84656	**0.91691**	0.81875	1.00000	**0.90034**	0.91013
Bayes	TFIDF	0.58974	0.24339	0.34457	0.40909	0.75573	0.53083	0.42082
	LDA-20	0.69737	0.84127	0.76259	0.67391	0.47328	0.55605	0.67804
	LDA-50	0.71429	0.89947	0.79625	0.76829	0.48092	0.59155	0.71245
	LDA-BOW-20	0.75893	0.89947	0.82324	0.80208	0.58779	0.67841	0.76395
	LDA-BOW-30	0.75490	0.81481	0.78372	0.69828	0.61832	0.65587	0.73138
	LDA-BOW-50	0.78505	0.88889	0.83375	0.80189	0.64885	0.71730	0.78608
	mESA-1000	0.99383	0.85185	**0.91738**	0.82278	0.99237	**0.89965**	0.91012

Table 4.2.: Balanced training data.

The SVM classifier performed best on most cases. The modified ESA model performed best of most models. In some cases the TF-IDF model was the best but usually in cases when all feature models had a high F1 score. LDA with 50 topics on the corpus in TF-IDF space performed best among the LDA models.

4.2.2. Results for Imbalanced data without SMOTE

Next topic combinations with a great imbalance were chosen. So far no SMOTE has been applied to increase the Precision. All results can be seen in C.2 of the appendix.
Table 4.3 shows the results for classes *acq* and *interest*. As can be seen the modified ESA model scores high F1 scores while other models fall behind. Again the SVM model performs best. Note that the scores for the majority class, *acq* in this case, usually stay high. The results for the minority class, *interest* here, are more important because it is the class we pretend to be the "interesting-news"-class. Here the modified ESA model performs much better than all the others. Reasons for the results will be discussed in Chapter 5.

		acq (1650/719)			*interest* (347/131)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.90404	0.99583	0.94772	0.94828	0.41985	0.58201	0.89135
	LDA-20	0.84588	1.00000	0.91651	0.00000	0.00000	0.00000	0.77526
	LDA-50	0.87871	0.94715	0.91165	0.49333	0.28244	0.35922	0.82651
	LDA-BOW-20	0.88198	0.96662	0.92236	0.61290	0.29008	0.39378	0.84090
	LDA-BOW-30	0.90811	0.93463	0.92118	0.57273	0.48092	0.52282	0.85978
	LDA-BOW-50	0.89831	0.95828	0.92732	0.63855	0.40458	0.49533	0.86074
	mESA-1000	0.95867	1.00000	**0.97890**	1.00000	0.76336	**0.86580**	0.96147
Centroid	TFIDF	0.94379	0.88734	0.91470	0.53448	0.70992	0.60984	0.86771
	LDA-20	0.87234	0.68428	0.76695	0.20629	0.45038	0.28297	0.69236
	LDA-50	0.92734	0.74548	0.82652	0.32721	0.67939	0.44169	0.76721
	LDA-BOW-20	0.91725	0.72462	0.80963	0.29787	0.64122	0.40678	0.74755
	LDA-BOW-30	0.97222	0.68150	0.80131	0.33815	0.89313	0.49057	0.75342
	LDA-BOW-50	0.92233	0.79277	0.85266	0.35776	0.63359	0.45730	0.79172
	mESA-1000	0.95429	0.92907	**0.94151**	0.66000	0.75573	**0.70463**	0.90500
Bayes	TFIDF	0.91520	0.87065	0.89237	0.43976	0.55725	0.49158	0.83060
	LDA-20	0.86207	0.73018	0.79066	0.19502	0.35878	0.25269	0.70775
	LDA-50	0.93774	0.69124	0.79584	0.30625	0.74809	0.43459	0.74016
	LDA-BOW-20	0.90560	0.78720	0.84226	0.32000	0.54962	0.40449	0.77479
	LDA-BOW-30	0.96673	0.68707	0.80325	0.33628	0.87023	0.48511	0.75422
	LDA-BOW-50	0.94536	0.72184	0.81861	0.33555	0.77099	0.46759	0.76451
	mESA-1000	0.94414	0.96384	**0.95389**	0.77586	0.68702	**0.72874**	0.91919

Table 4.3.: Imbalanced training data *without* SMOTE.

4.2.3. Results for Imbalanced data with SMOTE

In this section I present the results for imbalanced data with SMOTE. Only the SVM classifier was tested because it performed best earlier. All results can be seen in C.3.
As can be seen in Table 4.4 and Figure 4.1, the Recall of almost all minority classes and feature models increased and thus also increased the F1 scores. The F1 scores of the majority class did not decrease much.
The modified ESA model performed best on all test pairs.

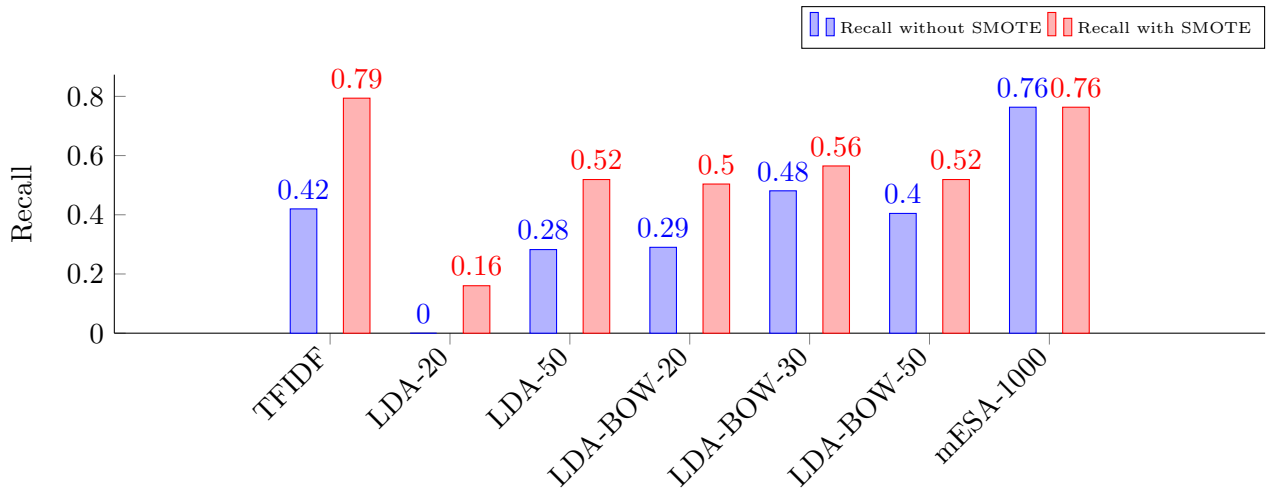

Figure 4.1.: Recall of minority class *interest* and majority class *acq* with and without SMOTE tested with SVM classifier.

	acq			*interest*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.95897	0.87761	0.91649	0.54167	0.79389	0.64396	600 / 400
LDA-20	0.86129	0.94993	0.90344	0.36842	0.16031	0.22340	100 / 400
LDA-50	0.91102	0.89708	0.90399	0.47887	0.51908	0.49817	100 / 600
LDA-BOW-20	0.90780	0.89013	0.89888	0.45517	0.50382	0.47826	100 / 600
LDA-BOW-30	0.91915	0.90125	0.91011	0.51034	0.56489	0.53623	100 / 500
LDA-BOW-50	0.91226	0.91099	0.91162	0.51515	0.51908	0.51711	100 / 400
mESA-1000	0.95861	0.99861	**0.97820**	0.99010	0.76336	**0.86207**	100 / 500

Table 4.4.: Imbalanced training data *with* SMOTE and SVM classifier.

4.3. Test on actual data

In this section I present the evaluation on "actual" data specific to the stated problem of this thesis. Over a five month period I collected my own news reading history by using a very early version of the news filtering system. In this time 14339 news articles were presented to me. Of these I read 192 news articles. Thus I end up with 14147 unread and 192 read articles. They make up the corpus.
For cross validation 200 unread and 10 read articles were selected randomly as test data. This split is supposed to simulate the incoming news articles of a day. All other articles were used for training. The training data has a great imbalance of 13947 to 182.
The tests were done on four feature models: modified ESA trained on all articles (mESA-1000), LDA with 100 topics trained on all articles in TF-IDF space (LDA-100), LDA with 50 and 100 topics trained on all articles in Bag-of-Words space (LDA-BOW-50 and LDA-BOW-100).
The test also highlights differences between training on the whole dataset and training on a subset created with the SMOTE algorithm. The results are from the best SMOTE parameters which are shown in Table 4.5. The parameter N denotes how many new synthetic minority samples, read samples in this case, should be created. For instance, $N = 400$ means that for 20 read samples 80 synthetic will be created. Parameter P denotes how many majority samples, unread samples in this case, are used. $P = 200$ means that $2 \cdot NumberOfReadSamples$ randomly selected unread samples will be used for training.

Model	Parameter N	Parameter P
mESA	400	100
LDA-100	100	500
LDA-BOW-50	100	400
LDA-BOW-100	400	100

Table 4.5.: SMOTE parameters for different models with best results.

Table 4.6 shows the average results after 10 iterations. The best results are featured in bold face.
The modified ESA model performs best on the minority class. As expected SMOTE increases the Recall of the minority class in return for a lower Precision. However, it does improve the F1 score as illustrated in Figures 4.2 and 4.3.
At large, the scores are much worse than on the Reuters-21578 dataset. Possible reasons are discussed in 5.1.3.

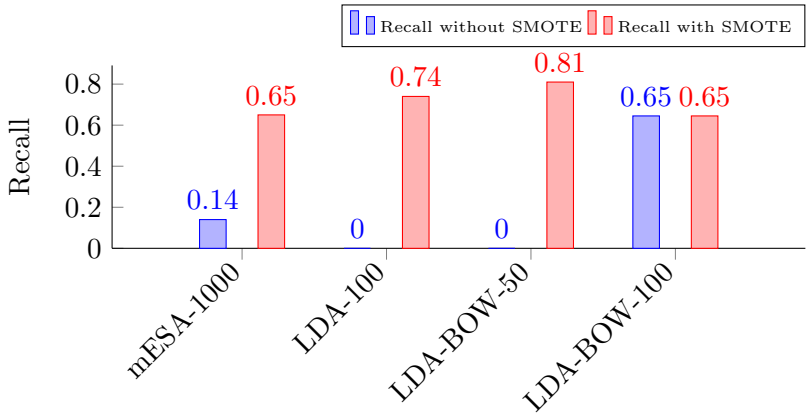

Figure 4.2.: Recall of minority class *interesting* with and without SMOTE tested with SVM classifier.

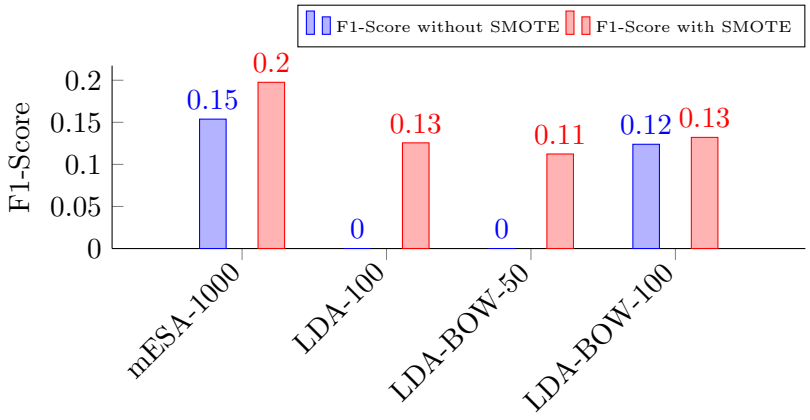

Figure 4.3.: F1-Score of minority class *interesting* with and without SMOTE tested with SVM classifier.

		Unread / Uninteresting			Read / Interesting		
	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score
without SMOTE	mESA-1000	0.95735	0.96500	0.96108	0.22505	0.14000	**0.15379**
	LDA-100	0.95238	1.00000	**0.97561**	0.00000	0.00000	0.00000
	LDA-BOW-50	0.95238	1.00000	**0.97561**	0.00000	0.00000	0.00000
	LDA-BOW-100	0.96968	0.56300	0.71177	0.06857	0.64500	0.12391
with SMOTE	mESA-1000	0.97691	0.74500	**0.84459**	0.11715	0.65000	**0.19755**
	LDA-100	0.97405	0.49450	0.65540	0.06864	0.74000	0.12560
	LDA-BOW-50	0.97409	0.36600	0.53079	0.06039	0.81000	0.11237
	LDA-BOW-100	0.97103	0.59200	0.73432	0.07369	0.64500	0.13209

Table 4.6.: Training on actual data with and without SMOTE.

5. Discussion

In this chapter I will discuss the results of the previous chapter and discuss some other aspects of news filtering and the feature models. I will also discuss the advantages and disadvantages of collaborative filtering and news filtering in general. At the end I will give a brief interpretation of the models presented in this thesis.

5.1. Results

The F1 scores for most feature models and classifiers are good compared to Debole and Sabistiani[9]. In general they were higher than 0.8. Really surprising is the performance of the combination of the TF-IDF model with the centroid classifier in comparison with more sophisticated algorithms. Both are really simple to implement. Thus, they form a great start for a "vanilla" implementation to test assumptions and collect more data as was done in this thesis.
The TF-IDF model preformed also quite well with the SVM classifier. The explanation is that SVM is very good with high dimensional data such as the one created by TF-IDF models. Naïve Bayes and the centroid classifier can have bad results from too much noise. Imagine two documents differ only in very few dimensions and all other dimensions have somewhat random values. Both Bayes and centroid classifier cannot handle such cases while the kernel of a SVM can reduce the noise a little.
The score for the LDA models scored lower than expected when one takes into account its sophistication and wide adoption. However, one has to keep in mind that LDA reduces the number of dimensions immensely and thus reduces the memory usage in comparable manner. The reduction also greatly improved the classifiers' performance.
It seems that LDA is more suitable for information retrieval and comparison tasks.

5.1.1. Evaluation of Modified ESA

The modified ESA model scored mostly the highest F1 scores or was near to the top score. Also, the model was very good on imbalanced data which will be important when working with actual data.
The reason lies within the feature selection which is performed via learning.

Only the most distinctive features are kept. So the noise is reduced as much as possible. However, this results in vectors that are not really sparse.
The feature selection has two main disadvantages. For one thing, training data for the classification case is needed. While TF-IDF and LDA can be trained on almost any corpus modified ESA requires the corpus to be classified. This makes the model unsuitable for non-classification cases such as information retrieval for which no classification of the training data is available. The process also makes the model inflexible for new classes that might appear.
The feature selection might also yield overfitting if the training corpus is not diverse enough. This is subject for future research.

5.1.2. Evaluation of SMOTE

As expected SMOTE, improved the Recall score of the minority classes and thus improved the F1 scores even though the Precision suffered a little bit. The reasons for this are explained in detail by Chawla[7]. The parameters for the best output for SMOTE differ a little from each other. It is important to do some testing to find the best fit.
In some rare cases, the application of SMOTE actually resulted in worse results. Therefore it has to be used with caution.
SMOTE also proved to be much more important on the actual data. The effects are more dramatic because the imbalance is much bigger on the data set gathered for this thesis. Again, parameters need to be tested for each model. The results also show that significant under-sampling the majority class does not lower the scores too much. This means a classifier can produce good results even if trained on a fraction of the whole class (1,000 compared to 10,000).

5.1.3. Results for Actual Data

In contrast to the tests on the Reuters-21578 dataset, the tests on actual data seem to be disappointing. In the best case the modified ESA model scores a Precision of 0.11. That means only a little over a 10th of all articles rated as top articles *are* actually top articles. At first glance this might be not satisfying since the whole point of a news filtering system is to distinguish between interesting and uninteresting news to give a better overview for the user.
So why is the score so weak?
One reason is the lack of enough training data. 182 articles do not seem to be enough to capture all of the user's interests. A solution can be to use collaborative filtering to enhance the training data. Section 5.2 gives more details on this.
Another much more important reason is the nature of the data. So far the news filtering system offers 24 news sources which report on mostly internet technology. That does not only mean that all articles are more related than, for example, a news on a football match and one on a presidential election.

The news sources also report often on the same news. If Google acquires a company, almost all news sources will report the event, but I will probably read only one article on it. Thus only one article on the news will be marked as interesting while other articles about exactly the same topic will be marked as uninteresting. A classifier will then learn that the news is not important since the "uninteresting" articles outweigh the one article marked as interesting. The application of SMOTE counters this effect only a little bit.

5.2. Collaborative VS. Individual Filtering

The filtering done for the thesis is based on intrinsic individual features. That means the profile of a user consists only of her feedback. The filtering is individual to each user. In contrast, collaborative filtering or collaborative recommendation tries to find users with similar interests and recommend items based on their interests. For instance, user A, B, and C listen to band X. A and B also listen to band Y. Following a collaborative approach, the chances are high that user C would also listen to band Y. I call A, B, and C neighbours.
This approach proved very successful for music and movies recommendations and can be easily applied to news filtering. Each news article would be seen as an atomic item. If user A, B, and C read news article Y and if A and B read news article X, chances are again high that C would be interested in news article X. Its content does not matter. However, users have time to discover music and movies. Both do not have such a short life span as news does. A collaborative news filter system would recommend news to me that is maybe one or two days old and thus not news any more. Such a system will also have trouble gathering data.
It seems this naïve approach would not work. That is why I decided for the individual approach in the thesis. However, a collaborative system can help to enhance the training data for a user model. If a user has read only 200 articles it is possible to discover more "read" articles by looking at the articles her neighbours have read. An algorithm can then add these discovered articles to the training data. Weighing the articles for learning might be wise.

5.3. Filtering News in General, Challenges, and Criticism

In this section I will talk about news filtering in general.
For one thing, the presentation is very important as I mentioned before. If news articles are presented in an unpleasant and unreadable formats it might not be read. This applies also to the order of the news selected. If news appears at the top of a page the likelihood that a user reads it or at least views it is high.

In general, design can generate noisy input for a recommendation engine. For instance, if a search engine would rank websites by their click rate, top results would get more clicks and stay at the top while more significant pages might not get clicks at all because they are at the end of the results.
Another risk of a news filter system is that it might "train" the user. If a user is told that some news articles are her personal top news, her habits and tastes might actually develop so that these articles and topics replace her prior "true" preferences. Whether this is possible will be the subject of further research.
Another challenge lies in the nature of news. As the name implies news covers new incidents. A user might be interesting in new topics and not reports on topics she has already read. For instance, if I read only news about Facebook, the social network, and Apple's iPhone, a news filtering system will only filter news around the topics "social network" and "smartphones". The system is not able to filter news about a new upcoming disruptive product by a new company. This problem can be faced with a collaborative approach detailed above and by using more features such as authorship and news source.
Also, it is really hard to define similarity between documents or news at all. When are two documents equal? Even for humans it is difficult to express the similarity between documents in numbers. However, surveys have shown that humans do not differ much from each others' estimations[6, 21, 12].

5.4. Philosophical View

This section will give a short Philosophical interpretation of the understanding a computer can get about the content of text.
It is widely believed that we can get a better understanding of the human thinking process by rebuilding it for example with a computer. This opinion is certainly true. However, I am not very optimistic in regard to teaching a computer the meaning of words. Philosophers have tried to give a definition of *meaning* of a word for centuries and have not come up with a complete model yet. However, a few models have prevailed. I'd like to use them to interpret the feature models I presented in 2.1.
The TF-IDF model essentially lets a computer program rank words by their "meaningfulness" in regard to a document. It can answer the question "How important is a word for the understanding of this text?" As I have shown this approach enables a computer to compare documents by their content. However, the computer does not gain a knowledge of the meaning of the words or even of the whole document.

Topics #	Members
1	project/NN, open/JJ, source/NN, development/NN, community/NN, software/NN, develop/VB, problem/NN, designer/NN, code/NN
12	summit/VB, president/NN, ceo/NN, founder/NN, chief/JJ, vice/NN, marketing/NN, executive/NN, officer/NN, manager/NN
25	email/NN, security/NN, account/NN, promo/NN, message/NN, send/VB, name/NN, hacker/NN, internet/NN, password/NN

Table 5.1.: Latent Dirichlet Allocation topics and their members. The complete table is in Appendix D.

Looking at the three topics in Table 5.1 generated by an LDA model one might think that each word is expressing some concept through its related topics. For instance, topic 1 and its words are clearly about the open source community. This would relate to traditional Proposition Theories of meaning[25, Chapter 5]. In these theories the meaning of a word or sentence is thought of as a proposition which is expressed by the word or sentence.
However, as Blei explains[3] there is no definite explanation yet of why related words have high probabilities for one topic. There are some theories of why related words seem to be grouped together but they have yet to be proved. Thus it is a very broad interpretation to say that topics give a computer program a way to deal with propositions.
Explicit Semantic Analysis is easier to interpret. Words are related to concepts represented by, for instance, Wikipedia articles. A computer program can actually point to concepts of a word. This approach is similar to enhancing text searches with knowledge graphs or knowledge bases where queries are not only answered with matching texts but facts. For instance, the query "Who was Alan Turing?" will not only be answered with texts containing the name Alan Turing but facts such as his date of birth and death and related concepts such as "John von Neumann". This approach is somewhat similar to different Referential Theories where the the meaning of a word is understood as the thing the word points to. However, these theories cannot explain the meaning of more abstract words such as "freedom". To what thing does "freedom" point?
The modified ESA model lacks the easy interpretation of the ESA model since features are filtered according to certain scores that do not consider the meanings of words.
In the end the proposed models do not come close to modern models. Modern speech theories also demand a response by the speaker such as the Turing Test does. The presented models alone cannot give such responses. So far they are passive.

6. Conclusion and Future Work

In this thesis I presented a news filtering system based on individual intrinsic user feedback.
I used several feature models to represent documents for a computer program, namely TF-IDF, LDA, and a modified ESA. These models were combined with three different classifiers to create a news filtering system. Tests on the Reuters-21578 dataset have shown that all combinations lead to good results. The modified ESA model performed best of the models and the SVM classifier best of the classifiers.
Tests on actual data gathered during the making of this thesis have shown that this is not yet sufficient to filter news. The bad performance by the featured models and the SVM classifier can partially be explained by the nature of the test data. However, in this case "bad" data cannot be an excuse since there cannot be such a thing as "bad" data when it was gathered with an actual system. Thus the problem itself proves to be still very hard to solve for computers. Moreover, the question arises if the problem of filtering interesting news for humans can be solved by using just individual intrinsic features. Can news be filtered by its content alone?
It is also questionable whether a user's taste in news will be consistent enough to make judgements on new content based on the performance of old news.
Overall the results indicate how close today's algorithm come to understand natural language. Such an understanding is the basis of passing the Turing Test which Alan Turing proposed in 1950 to exhibit the intelligence of computers[31].
I am optimistic that computers will be able to completely understand natural language some day but I am doubtful that they will be able to answer on arbitrary input.

6.1. Future Work

For future work the individual filtering should be combined with collaborative filtering approaches. Also, a long-term and short-term user model might improve the overall performance as described by Billsus and Pazzani[2].
So far only the click on a headline was used as feedback. A feedback weighted by the reading time seems promising to improve the user model. However, it is difficult to measure this reliably.
Obviously, 24 technology news sites do not cover all news. More news sources

should be added. To capture different interests of a user, clustering of the user model into separate clusters might be suitable. Franek describes in his Ph.D. thesis a suitable algorithm to find the optimal number of clusters[13].
The ESA and the modified ESA model could be used to compare documents across languages. This seems to an interesting approach to filtering for example German and English news with a language-independent user model.
Processing the huge Wikipedia corpus proved to be a great challenge. Cluster computing frameworks such as Hadoop[1] in combination with Mahout[2], or Storm[3] seem to be a great approach which should be further analysed.

[1] http://hadoop.apache.org/
[2] http://mahout.apache.org/
[3] http://storm-project.net/

Appendices

A. Short Documents

This is an example of two documents that cannot be well distinguished in TF-IDF space but in LDA space. It is taken from "Latent Dirichlet Allocation (LDA) and Google's Rankings are Remarkably Well Correlated"[1].

Dropping his meeting notes at the door, he jiggled the keys into the lock but found it wouldn't budge.

Figure A.1.: Document A.

Her hands mercilessly pounded the keys, notes cascading into the surrounding stairway.

Figure A.2.: Document B.

[1]See: http://www.seomoz.org/blog/lda-and-googles-rankings-well-correlated

B. Complete Centroid Classifier Implementation

The complete implementation of the centroid classifier.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''
Created on 12.12.2012

@author: karsten jeschkies

Implementation of a centroid classifier
'''
from gensim.similarities import MatrixSimilarity
import numpy as np
from sklearn.utils import array2d

class CentroidClassifier(object):

    def fit(self, X, y):
        '''
        Parameters
        ----------
        """Fit Centroid classifier according to X, y

        Parameters
        ----------
        X : array-like, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of samples
            and n_features is the number of features.

        y : array-like, shape = [n_samples]
            Target values.
        '''

        if X.shape[0] != y.shape[0]:
            raise ValueError("X and y have incompatible shapes")
```

```
34
35          n_features = X.shape[1]
36          self.classes = unique_y = np.unique(y)
37          n_classes = unique_y.shape[0]
38          centroids = np.zeros(shape=(n_classes, n_features))
39
40          #Calculate mean for each class
41          for i, y_i in enumerate(unique_y):
42              centroids[i, :] = np.mean(X[y == y_i, :], axis=0)
43
44          #Build similarity index from centroids
45          self.similarity_index = MatrixSimilarity(corpus = centroids,
46                                                   num_features = n_features)
47
48      def predict(self, X):
49          """
50          Perform classification on an array of test vectors X.
51
52          Parameters
53          ----------
54          X : array-like, shape = [n_samples, n_features]
55
56          Returns
57          -------
58          C : array, shape = [n_samples]
59              Predicted target values for X
60          """
61          X = array2d(X)
62
63          n_samples = X.shape[0]
64          predictions = np.empty(shape=(n_samples))
65          for i, sample in enumerate(X):
66              similarities = self.similarity_index[sample]
67              class_id = np.argmax(similarities)
68              predictions[i] = self.classes[class_id]
69
70          return predictions
```

C. Evaluation Results

The F1 scores are highlighted in grey. The best results for each classifier are in bold. The average column shows the weighted average of the F1 scores. The numbers following the topic/class names indicate training size and test size.

C.1. Results for Balanced Data Sets

		crude (389/189)			*interest* (347/131)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.56769	0.68783	0.62201	0.35165	0.24427	0.28829	0.48539
	LDA-20	0.77193	0.69841	0.73333	0.61745	0.70229	0.65714	0.70214
	LDA-50	0.84971	0.77778	0.81215	0.71429	0.80153	0.75540	0.78892
	LDA-BOW-20	0.90419	0.79894	0.84831	0.75163	0.87786	0.80986	0.83257
	LDA-BOW-30	0.77619	0.86243	0.81704	0.76364	0.64122	0.69710	0.76794
	LDA-BOW-50	0.77232	0.91534	0.83777	0.83333	0.61069	0.70485	0.78336
	mESA-1000	1.00000	0.84656	**0.91691**	0.81875	1.00000	**0.90034**	0.91013
Centroid	TFIDF	0.57277	0.64550	0.60697	0.37383	0.30534	0.33613	0.49609
	LDA-20	0.76705	0.71429	0.73973	0.62500	0.68702	0.65455	0.70486
	LDA-50	0.73729	0.92063	0.81882	0.82143	0.52672	0.64186	0.74638
	LDA-BOW-20	0.78070	0.94180	0.85372	0.88043	0.61832	0.72646	0.80162
	LDA-BOW-30	0.76126	0.89418	0.82238	0.79592	0.59542	0.68122	0.76460
	LDA-BOW-50	0.78667	0.93651	0.85507	0.87368	0.63359	0.73451	0.80572
	mESA-1000	1.00000	0.84656	**0.91691**	0.81875	1.00000	**0.90034**	0.91013
Bayes	TFIDF	0.58974	0.24339	0.34457	0.40909	0.75573	0.53083	0.42082
	LDA-20	0.69737	0.84127	0.76259	0.67391	0.47328	0.55605	0.67804
	LDA-50	0.71429	0.89947	0.79625	0.76829	0.48092	0.59155	0.71245
	LDA-BOW-20	0.75893	0.89947	0.82324	0.80208	0.58779	0.67841	0.76395
	LDA-BOW-30	0.75490	0.81481	0.78372	0.69828	0.61832	0.65587	0.73138
	LDA-BOW-50	0.78505	0.88889	0.83375	0.80189	0.64885	0.71730	0.78608
	mESA-1000	0.99383	0.85185	**0.91738**	0.82278	0.99237	**0.89965**	0.91012

Table C.1.: Results for balanced training data sets *crude* and *interest*.

		crude (389/189)			*money-fx* (538/179)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.58125	0.49206	0.53295	0.53846	0.62570	0.57881	0.55526
	LDA-20	0.84397	0.62963	0.72121	0.69163	0.87709	0.77340	0.74660
	LDA-50	0.83815	0.76720	0.80110	0.77436	0.84358	0.80749	0.80421
	LDA-BOW-20	0.90741	0.77778	0.83761	0.79612	0.91620	0.85195	0.84458
	LDA-BOW-30	0.73973	0.85714	0.79412	0.81879	0.68156	0.74390	0.76969
	LDA-BOW-50	0.92857	0.75661	0.83382	0.78505	0.93855	0.85496	0.84410
	mESA-1000	0.99383	0.85185	**0.91738**	0.86408	0.99441	**0.92468**	0.92093
Centroid	TFIDF	0.53061	0.68783	0.59908	0.52033	0.35754	0.42384	0.51384
	LDA-20	0.67521	0.83598	0.74704	0.76866	0.57542	0.65815	0.70380
	LDA-50	0.74892	0.91534	0.82381	0.88321	0.67598	0.76582	0.79560
	LDA-BOW-20	0.75546	0.91534	0.82775	0.88489	0.68715	0.77358	0.80140
	LDA-BOW-30	0.72727	0.88889	0.80000	0.84672	0.64804	0.73418	0.76798
	LDA-BOW-50	0.77826	0.94709	0.85442	0.92754	0.71508	0.80757	0.83163
	mESA-1000	0.82533	1.00000	**0.90431**	1.00000	0.77654	**0.87421**	0.88967
Bayes	TFIDF	0.50730	0.73545	0.60043	0.46809	0.24581	0.32234	0.46517
	LDA-20	0.69406	0.80423	0.74510	0.75168	0.62570	0.68293	0.71486
	LDA-50	0.68675	0.90476	0.78082	0.84874	0.56425	0.67785	0.73074
	LDA-BOW-20	0.75217	0.91534	0.82578	0.88406	0.68156	0.76972	0.79851
	LDA-BOW-30	0.72368	0.87302	0.79137	0.82857	0.64804	0.72727	0.76019
	LDA-BOW-50	0.76856	0.93122	0.84211	0.90647	0.70391	0.79245	0.81795
	mESA-1000	0.99383	0.85185	**0.91738**	0.86408	0.99441	**0.92468**	0.92093

Table C.2.: Resluts for balanced training data sets *crude* and *money-fx*.

		crude (389/189)			*trade* (369/116)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.95918	0.99471	0.97662	0.99083	0.93103	0.96000	0.97030
	LDA-20	0.74766	0.84656	0.79404	0.68132	0.53448	0.59903	0.71988
	LDA-50	0.83333	0.92593	0.87719	0.85263	0.69828	0.76777	0.83558
	LDA-BOW-20	0.84343	0.88360	0.86305	0.79439	0.73276	0.76233	0.82474
	LDA-BOW-30	0.85922	0.93651	0.89620	0.87879	0.75000	0.80930	0.86315
	LDA-BOW-50	0.88119	0.94180	0.91049	0.89320	0.79310	0.84018	0.88375
	mESA-1000	0.96891	0.98942	**0.97906**	0.98214	0.94828	**0.96491**	0.97368
Centroid	TFIDF	0.97253	0.93651	**0.95418**	0.90244	0.95690	**0.92887**	0.94455
	LDA-20	0.80723	0.70899	0.75493	0.60432	0.72414	0.65882	0.71838
	LDA-50	0.83920	0.88360	0.86082	0.79245	0.72414	0.75676	0.82124
	LDA-BOW-20	0.87978	0.85185	0.86559	0.77049	0.81034	0.78992	0.83681
	LDA-BOW-30	0.87565	0.89418	0.88482	0.82143	0.79310	0.80702	0.85523
	LDA-BOW-50	0.94079	0.75661	0.83871	0.69935	0.92241	0.79554	0.82229
	mESA-1000	0.96914	0.83069	0.89459	0.77622	0.95690	0.85714	0.88035
Bayes	TFIDF	0.95105	0.71958	0.81928	0.67284	0.93966	0.78417	0.80593
	LDA-20	0.70760	0.64021	0.67222	0.49254	0.56897	0.52800	0.61737
	LDA-50	0.79096	0.74074	0.76503	0.61719	0.68103	0.64754	0.72034
	LDA-BOW-20	0.85641	0.88360	0.86979	0.80000	0.75862	0.77876	0.83517
	LDA-BOW-30	0.84393	0.77249	0.80663	0.67424	0.76724	0.71774	0.77282
	LDA-BOW-50	0.86082	0.88360	0.87206	0.80180	0.76724	0.78414	0.83862
	mESA-1000	0.96875	0.98413	**0.97638**	0.97345	0.94828	**0.96070**	0.97041

Table C.3.: Results for balanced training data sets *crude* and *trade*.

		interest (347/131)			*money-fx* (538/179)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.31169	0.18321	0.23077	0.54077	0.70391	0.61165	0.45070
	LDA-20	0.66667	0.01527	0.02985	0.57980	0.99441	0.73251	0.43558
	LDA-50	0.67857	0.29008	0.40642	0.63386	0.89944	0.74365	0.60114
	LDA-BOW-20	0.58824	0.07634	0.13514	0.58703	0.96089	0.72881	0.47794
	LDA-BOW-30	0.42857	0.09160	0.15094	0.57801	0.91061	0.70716	0.47211
	LDA-BOW-50	0.61364	0.20611	0.30857	0.60902	0.90503	0.72809	0.55081
	mESA-1000	0.68085	0.48855	**0.56889**	0.68981	0.83240	**0.75443**	0.67602
Centroid	TFIDF	0.42553	0.61069	0.50157	0.58197	0.39665	0.47176	0.48436
	LDA-20	0.50350	0.54962	0.52555	0.64671	0.60335	0.62428	0.58256
	LDA-50	0.53521	0.58015	0.55678	0.67262	0.63128	**0.65130**	0.61135
	LDA-BOW-20	0.51562	0.50382	0.50965	0.64286	0.65363	0.64820	0.58965
	LDA-BOW-30	0.50820	0.47328	0.49012	0.63298	0.66480	0.64850	0.58157
	LDA-BOW-50	0.48148	0.59542	0.53242	0.64189	0.53073	0.58104	0.56050
	mESA-1000	0.56150	0.80153	**0.66038**	0.78862	0.54190	0.64238	0.64999
Bayes	TFIDF	0.38308	0.58779	0.46386	0.50459	0.30726	0.38194	0.41656
	LDA-20	0.45794	0.37405	0.41176	0.59606	0.67598	0.63351	0.53980
	LDA-50	0.57778	0.39695	0.47059	0.64091	0.78771	**0.70677**	0.60696
	LDA-BOW-20	0.52564	0.31298	0.39234	0.61207	0.79330	0.69100	0.56479
	LDA-BOW-30	0.41905	0.33588	0.37288	0.57561	0.65922	0.61458	0.51244
	LDA-BOW-50	0.47778	0.32824	0.38914	0.60000	0.73743	0.66165	0.54650
	mESA-1000	0.56069	0.74046	**0.63816**	0.75182	0.57542	0.65190	0.64609

Table C.4.: Results for balanced training data sets *interest* and *money-fx*.

		interest (347/131)			*trade* (369/116)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	1.00000	0.45802	0.62827	0.62032	1.00000	0.76568	0.69280
	LDA-20	0.73984	0.69466	0.71654	0.67742	0.72414	0.70000	0.70877
	LDA-50	0.77027	0.87023	0.81720	0.82828	0.70690	0.76279	0.79165
	LDA-BOW-20	0.70629	0.77099	0.73723	0.71154	0.63793	0.67273	0.70694
	LDA-BOW-30	0.75969	0.74809	0.75385	0.72034	0.73276	0.72650	0.74100
	LDA-BOW-50	0.77372	0.80916	0.79104	0.77273	0.73276	0.75221	0.77281
	mESA-1000	0.93382	0.96947	**0.95131**	0.96396	0.92241	**0.94273**	0.94728
Centroid	TFIDF	0.95098	0.74046	0.83262	0.76552	0.95690	0.85057	0.84105
	LDA-20	0.73469	0.54962	0.62882	0.60403	0.77586	0.67925	0.65250
	LDA-50	0.82075	0.66412	0.73418	0.68794	0.83621	0.75486	0.74389
	LDA-BOW-20	0.67778	0.46565	0.55204	0.55414	0.75000	0.63736	0.59211
	LDA-BOW-30	0.77064	0.64122	0.70000	0.65942	0.78448	0.71654	0.70777
	LDA-BOW-50	0.82022	0.55725	0.66364	0.63291	0.86207	0.72993	0.69477
	mESA-1000	0.89916	0.81679	**0.85600**	0.81250	0.89655	**0.85246**	0.85434
Bayes	TFIDF	0.92045	0.61832	0.73973	0.68553	0.93966	0.79273	0.76462
	LDA-20	0.59000	0.45038	0.51082	0.51020	0.64655	0.57034	0.53878
	LDA-50	0.66667	0.50382	0.57391	0.56081	0.71552	0.62879	0.59968
	LDA-BOW-20	0.59341	0.41221	0.48649	0.50641	0.68103	0.58088	0.53082
	LDA-BOW-30	0.70000	0.53435	0.60606	0.58503	0.74138	0.65399	0.62857
	LDA-BOW-50	0.69512	0.43511	0.53521	0.55152	0.78448	0.64769	0.58803
	mESA-1000	0.88235	0.91603	**0.89888**	0.90090	0.86207	**0.88106**	0.89051

Table C.5.: Results for balanced training data sets *interest* and *trade*.

		money-fx (538/179)			*trade* (369/116)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.95722	1.00000	**0.97814**	1.00000	0.93103	**0.96429**	0.97269
	LDA-20	0.75263	0.79888	0.77507	0.65714	0.59483	0.62443	0.71584
	LDA-50	0.78804	0.81006	0.79890	0.69369	0.66379	0.67841	0.75152
	LDA-BOW-20	0.72727	0.89385	0.80201	0.74667	0.48276	0.58639	0.71722
	LDA-BOW-30	0.74479	0.79888	0.77089	0.65049	0.57759	0.61187	0.70836
	LDA-BOW-50	0.82632	0.87709	0.85095	0.79048	0.71552	0.75113	0.81170
	mESA-1000	0.92021	0.96648	0.94278	0.94393	0.87069	0.90583	0.92825
Centroid	TFIDF	0.96970	0.89385	**0.93023**	0.85385	0.95690	**0.90244**	0.91930
	LDA-20	0.77863	0.56983	0.65806	0.53049	0.75000	0.62143	0.64366
	LDA-50	0.84314	0.48045	0.61210	0.51813	0.86207	0.64725	0.62592
	LDA-BOW-20	0.75000	0.65363	0.69851	0.55396	0.66379	0.60392	0.66131
	LDA-BOW-30	0.74107	0.46369	0.57045	0.47541	0.75000	0.58194	0.57497
	LDA-BOW-50	0.87705	0.59777	0.71096	0.58382	0.87069	0.69896	0.70624
	mESA-1000	0.90323	0.46927	0.61765	0.52970	0.92241	0.67296	0.63940
Bayes	TFIDF	0.95541	0.83799	**0.89286**	0.78986	0.93966	**0.85827**	0.87926
	LDA-20	0.68293	0.46927	0.55629	0.44767	0.66379	0.53472	0.54781
	LDA-50	0.75781	0.54190	0.63192	0.50898	0.73276	0.60071	0.61965
	LDA-BOW-20	0.71901	0.48603	0.58000	0.47126	0.70690	0.56552	0.57431
	LDA-BOW-30	0.68376	0.44693	0.54054	0.44382	0.68103	0.53741	0.53931
	LDA-BOW-50	0.82645	0.55866	0.66667	0.54598	0.81897	0.65517	0.66215
	mESA-1000	0.91566	0.84916	0.88116	0.79070	0.87931	0.83265	0.86209

Table C.6.: Results for balanced training data sets *money-fx* and *trade*.

		acq (1650/719)			*earn* (2877/1087)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.98179	0.82476	0.89645	0.89517	0.98988	0.94015	0.92275
	LDA-20	0.70517	0.32267	0.44275	0.67028	0.91076	0.77223	0.64106
	LDA-50	0.69799	0.43394	0.53516	0.70052	0.87580	0.77841	0.68157
	LDA-BOW-20	0.70778	0.51878	0.59872	0.72948	0.85833	0.78867	0.71305
	LDA-BOW-30	0.66857	0.32545	0.43779	0.66690	0.89328	0.76366	0.63393
	LDA-BOW-50	0.78661	0.52295	0.62824	0.74172	0.90616	0.81573	0.74109
	mESA-1000	0.97097	0.83727	**0.89918**	0.90135	0.98344	**0.94061**	0.92411
Centroid	TFIDF	0.90883	0.88734	**0.89796**	0.92663	0.94112	**0.93382**	0.91954
	LDA-20	0.62408	0.59110	0.60714	0.73867	0.76449	0.75136	0.69394
	LDA-50	0.68801	0.58275	0.63102	0.74937	0.82521	0.78546	0.72398
	LDA-BOW-20	0.61538	0.61196	0.61367	0.74427	0.74701	0.74564	0.69310
	LDA-BOW-30	0.63839	0.63839	0.63839	0.76081	0.76081	0.76081	0.71207
	LDA-BOW-50	0.43704	0.74826	0.55179	0.68522	0.36247	0.47413	0.50505
	mESA-1000	0.80493	0.99861	0.89137	0.99891	0.83993	0.91254	0.90411
Bayes	TFIDF	0.82672	0.86926	0.84746	0.91048	0.87948	0.89471	0.87590
	LDA-20	0.62891	0.58693	0.60719	0.73833	0.77093	0.75428	0.69572
	LDA-50	0.64931	0.58971	0.61808	0.74415	0.78933	0.76607	0.70715
	LDA-BOW-20	0.65196	0.55494	0.59955	0.73199	0.80405	0.76633	0.69993
	LDA-BOW-30	0.60676	0.54937	0.57664	0.71948	0.76449	0.74130	0.67575
	LDA-BOW-50	0.65751	0.63282	0.64493	0.76302	0.78197	0.77238	0.72164
	mESA-1000	0.81528	0.99444	**0.89599**	0.99569	0.85097	**0.91766**	0.90903

Table C.7.: Results for balanced training data sets *acq* and *earn*.

C.2. Different Results for Imbalanced Data Sets without SMOTE

		acq (1650/719)			*crude* (389/189)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.93693	0.99166	**0.96351**	0.95918	0.74603	0.83929	0.93766
	LDA-20	0.79185	1.00000	0.88384	0.00000	0.00000	0.00000	0.69987
	LDA-50	0.91532	0.94715	0.93096	0.76829	0.66667	0.71388	0.88578
	LDA-BOW-20	0.90247	0.96523	0.93280	0.82014	0.60317	0.69512	0.88332
	LDA-BOW-30	0.89495	0.93602	0.91502	0.70513	0.58201	0.63768	0.85729
	LDA-BOW-50	0.90909	0.94576	0.92706	0.75625	0.64021	0.69341	0.87843
	mESA-1000	0.95168	0.98609	0.96858	0.93865	0.80952	**0.86932**	0.94792
Centroid	TFIDF	0.97121	0.89152	0.92966	0.68548	0.89947	0.77803	0.89810
	LDA-20	0.90926	0.66898	0.77083	0.37203	0.74603	0.49648	0.71373
	LDA-50	0.93740	0.79138	0.85822	0.50166	0.79894	0.61633	0.80787
	LDA-BOW-20	0.93000	0.77608	0.84610	0.47727	0.77778	0.59155	0.79311
	LDA-BOW-30	0.95616	0.63700	0.76461	0.39161	0.88889	0.54369	0.71862
	LDA-BOW-50	0.97452	0.63839	0.77143	0.40503	0.93651	0.56550	0.72856
	mESA-1000	0.95898	0.94298	**0.95091**	0.79602	0.84656	**0.82051**	0.92377
Bayes	TFIDF	0.89912	0.85535	0.87669	0.53571	0.63492	0.58111	0.81517
	LDA-20	0.90943	0.67038	0.77182	0.37302	0.74603	0.49735	0.71469
	LDA-50	0.95606	0.75661	0.84472	0.48378	0.86772	0.62121	0.79820
	LDA-BOW-20	0.95455	0.75939	0.84586	0.48512	0.86243	0.62095	0.79904
	LDA-BOW-30	0.93381	0.72601	0.81690	0.43553	0.80423	0.56506	0.76448
	LDA-BOW-50	0.94040	0.78999	0.85865	0.50329	0.80952	0.62069	0.80912
	mESA-1000	0.95132	0.95132	**0.95132**	0.81481	0.81481	**0.81481**	0.92291

Table C.8.: Results for imbalanced training data sets *acq* and *crude without* SMOTE.

		acq (1650/719)			*grain* (433/149)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.95816	0.98748	0.97260	0.92913	0.79195	0.85507	0.95243
	LDA-20	0.86582	0.95132	0.90656	0.55128	0.28859	0.37885	0.81598
	LDA-50	0.90000	0.95132	0.92495	0.67593	0.48993	0.56809	0.86369
	LDA-BOW-20	0.88948	0.92907	0.90884	0.56410	0.44295	0.49624	0.83802
	LDA-BOW-30	0.91851	0.92490	0.92169	0.62500	0.60403	0.61433	0.86893
	LDA-BOW-50	0.94521	0.95967	0.95238	0.78986	0.73154	0.75958	0.91929
	mESA-1000	0.97548	0.99583	**0.98555**	0.97761	0.87919	**0.92580**	0.97529
Centroid	TFIDF	0.96991	0.94159	**0.95554**	0.75294	0.85906	**0.80251**	0.92927
	LDA-20	0.91213	0.72184	0.80590	0.33110	0.66443	0.44196	0.74343
	LDA-50	0.94388	0.77191	0.84927	0.41429	0.77852	0.54079	0.79632
	LDA-BOW-20	0.95085	0.61892	0.74979	0.31500	0.84564	0.45902	0.69988
	LDA-BOW-30	0.96694	0.65090	0.77805	0.34635	0.89262	0.49906	0.73016
	LDA-BOW-50	0.98384	0.67733	0.80231	0.37802	0.94631	0.54023	0.75732
	mESA-1000	1.00000	0.84562	0.91635	0.57308	1.00000	0.72861	0.88412
Bayes	TFIDF	0.97360	0.82058	0.89057	0.50763	0.89262	0.64720	0.84879
	LDA-20	0.88354	0.79138	0.83492	0.33036	0.49664	0.39678	0.75971
	LDA-50	0.93309	0.69819	0.79873	0.34242	0.75839	0.47182	0.74261
	LDA-BOW-20	0.92424	0.67872	0.78268	0.32059	0.73154	0.44581	0.72485
	LDA-BOW-30	0.94403	0.70376	0.80637	0.35843	0.79866	0.49480	0.75289
	LDA-BOW-50	0.97173	0.76495	0.85603	0.44040	0.89262	0.58980	0.81033
	mESA-1000	0.97775	0.97775	**0.97775**	0.89262	0.89262	**0.89262**	0.96313

Table C.9.: Results for imbalanced training data sets *acq* and *grain without* SMOTE.

		acq (1650/719)			*interest* (347/131)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.90404	0.99583	0.94772	0.94828	0.41985	0.58201	0.89135
	LDA-20	0.84588	1.00000	0.91651	0.00000	0.00000	0.00000	0.77526
	LDA-50	0.87871	0.94715	0.91165	0.49333	0.28244	0.35922	0.82651
	LDA-BOW-20	0.88198	0.96662	0.92236	0.61290	0.29008	0.39378	0.84090
	LDA-BOW-30	0.90811	0.93463	0.92118	0.57273	0.48092	0.52282	0.85978
	LDA-BOW-50	0.89831	0.95828	0.92732	0.63855	0.40458	0.49533	0.86074
	mESA-1000	0.95867	1.00000	**0.97890**	1.00000	0.76336	**0.86580**	0.96147
Centroid	TFIDF	0.94379	0.88734	0.91470	0.53448	0.70992	0.60984	0.86771
	LDA-20	0.87234	0.68428	0.76695	0.20629	0.45038	0.28297	0.69236
	LDA-50	0.92734	0.74548	0.82652	0.32721	0.67939	0.44169	0.76721
	LDA-BOW-20	0.91725	0.72462	0.80963	0.29787	0.64122	0.40678	0.74755
	LDA-BOW-30	0.97222	0.68150	0.80131	0.33815	0.89313	0.49057	0.75342
	LDA-BOW-50	0.92233	0.79277	0.85266	0.35776	0.63359	0.45730	0.79172
	mESA-1000	0.95429	0.92907	**0.94151**	0.66000	0.75573	**0.70463**	0.90500
Bayes	TFIDF	0.91520	0.87065	0.89237	0.43976	0.55725	0.49158	0.83060
	LDA-20	0.86207	0.73018	0.79066	0.19502	0.35878	0.25269	0.70775
	LDA-50	0.93774	0.69124	0.79584	0.30625	0.74809	0.43459	0.74016
	LDA-BOW-20	0.90560	0.78720	0.84226	0.32000	0.54962	0.40449	0.77479
	LDA-BOW-30	0.96673	0.68707	0.80325	0.33628	0.87023	0.48511	0.75422
	LDA-BOW-50	0.94536	0.72184	0.81861	0.33555	0.77099	0.46759	0.76451
	mESA-1000	0.94414	0.96384	**0.95389**	0.77586	0.68702	**0.72874**	0.91919

Table C.10.: Results for imbalanced training data sets *acq* and *interest without* SMOTE.

		acq (1650/719)			*money-fx* (538/179)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.94040	0.98748	0.96336	0.93706	0.74860	0.83230	0.93724
	LDA-20	0.80854	0.97497	0.88398	0.41935	0.07263	0.12381	0.73246
	LDA-50	0.87760	0.93741	0.90652	0.65385	0.47486	0.55016	0.83549
	LDA-BOW-20	0.90896	0.88873	0.89873	0.58974	0.64246	0.61497	0.84217
	LDA-BOW-30	0.89424	0.92907	0.91132	0.66225	0.55866	0.60606	0.85047
	LDA-BOW-50	0.90212	0.94854	0.92475	0.73944	0.58659	0.65421	0.87082
	mESA-1000	0.94730	1.00000	**0.97294**	1.00000	0.77654	**0.87421**	0.95326
Centroid	TFIDF	0.96391	0.89152	0.92630	0.66524	0.86592	**0.75243**	0.89164
	LDA-20	0.86924	0.65647	0.74802	0.30423	0.60335	0.40449	0.67954
	LDA-50	0.90290	0.73713	0.81164	0.39228	0.68156	0.49796	0.74911
	LDA-BOW-20	0.91892	0.70932	0.80063	0.39067	0.74860	0.51341	0.74338
	LDA-BOW-30	0.91798	0.80946	0.86031	0.48106	0.70950	0.57336	0.80311
	LDA-BOW-50	0.91922	0.79138	0.85052	0.46237	0.72067	0.56332	0.79327
	mESA-1000	0.94101	0.93185	**0.93641**	0.73656	0.76536	0.75068	0.89939
Bayes	TFIDF	0.92868	0.81502	0.86815	0.50187	0.74860	0.60090	0.81488
	LDA-20	0.86082	0.69680	0.77018	0.31013	0.54749	0.39596	0.69558
	LDA-50	0.91139	0.70097	0.79245	0.37681	0.72626	0.49618	0.73340
	LDA-BOW-20	0.96614	0.67455	0.79443	0.40909	0.90503	0.56348	0.74839
	LDA-BOW-30	0.92416	0.72879	0.81493	0.41088	0.75978	0.53333	0.75880
	LDA-BOW-50	0.93675	0.76217	0.84049	0.45367	0.79330	0.57724	0.78802
	mESA-1000	0.93531	0.96523	**0.95003**	0.83974	0.73184	**0.78209**	0.91656

Table C.11.: Results for imbalanced training data sets *acq* and *money-fx without* SMOTE.

		acq (1650/719)			*trade* (369/116)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.98489	0.99722	0.99102	0.98131	0.90517	0.94170	0.98417
	LDA-20	0.88703	0.96106	0.92256	0.50000	0.24138	0.32558	0.83963
	LDA-50	0.93154	0.96523	0.94809	0.72222	0.56034	0.63107	0.90405
	LDA-BOW-20	0.89708	0.89708	0.89708	0.36207	0.36207	0.36207	0.82275
	LDA-BOW-30	0.94110	0.95549	0.94824	0.69524	0.62931	0.66063	0.90829
	LDA-BOW-50	0.95125	0.94993	0.95059	0.69231	0.69828	0.69528	0.91512
	mESA-1000	0.99034	0.99861	**0.99446**	0.99091	0.93966	**0.96460**	0.99031
Centroid	TFIDF	0.99270	0.94576	**0.96866**	0.74000	0.95690	**0.83459**	0.95004
	LDA-20	0.94008	0.63282	0.75644	0.24786	0.75000	0.37259	0.70312
	LDA-50	0.95029	0.69124	0.80032	0.28846	0.77586	0.42056	0.74756
	LDA-BOW-20	0.94687	0.69402	0.80096	0.28571	0.75862	0.41509	0.74736
	LDA-BOW-30	0.96546	0.81641	0.88470	0.41850	0.81897	0.55394	0.83875
	LDA-BOW-50	0.97431	0.68567	0.80490	0.31307	0.88793	0.46292	0.75739
	mESA-1000	0.98949	0.91655	0.95162	0.64497	0.93966	0.76491	0.92569
Bayes	TFIDF	0.96708	0.85814	0.90936	0.48223	0.81897	0.60703	0.86736
	LDA-20	0.89648	0.63839	0.74574	0.19505	0.54310	0.28702	0.68201
	LDA-50	0.93085	0.73018	0.81839	0.28413	0.66379	0.39793	0.75998
	LDA-BOW-20	0.91667	0.68846	0.78634	0.24068	0.61207	0.34550	0.72510
	LDA-BOW-30	0.93366	0.80250	0.86313	0.34562	0.64655	0.45045	0.80580
	LDA-BOW-50	0.95017	0.79555	0.86601	0.36910	0.74138	0.49284	0.81417
	mESA-1000	0.97343	0.96801	**0.97071**	0.80833	0.83621	**0.82203**	0.95006

Table C.12.: Results for imbalanced training data sets *acq* and *trade without* SMOTE.

		crude (389/189)			*earn* (2877/1087)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.23000	0.24339	0.23650	0.86710	0.85833	0.86269	0.76994
	LDA-20	0.78125	0.26455	0.39526	0.88531	0.98712	0.93345	0.85373
	LDA-50	0.84354	0.65608	0.73810	0.94243	0.97884	0.96029	0.92738
	LDA-BOW-20	0.59140	0.29101	0.39007	0.88673	0.96504	0.92423	0.84511
	LDA-BOW-30	0.80909	0.47090	0.59532	0.91424	0.98068	0.94629	0.89431
	LDA-BOW-50	0.81935	0.67196	0.73837	0.94469	0.97424	0.95924	0.92652
	mESA-1000	0.98710	0.80952	**0.88953**	0.96789	0.99816	**0.98279**	0.96898
Centroid	TFIDF	0.21685	0.92593	0.35141	0.97015	0.41858	0.58483	0.55026
	LDA-20	0.39942	0.72487	0.51504	0.94427	0.81049	0.87228	0.81936
	LDA-50	0.57143	0.82540	0.67532	0.96710	0.89236	**0.92823**	0.89077
	LDA-BOW-20	0.44412	0.79894	0.57089	0.95940	0.82613	0.88779	0.84085
	LDA-BOW-30	0.19424	0.82011	0.31408	0.92887	0.40846	0.56741	0.52989
	LDA-BOW-50	0.21302	0.95238	0.34816	0.97912	0.38822	0.55599	0.52521
	mESA-1000	0.52500	1.00000	**0.68852**	1.00000	0.84269	0.91463	0.88114
Bayes	TFIDF	0.18605	0.63492	0.28777	0.89065	0.51702	0.65425	0.59997
	LDA-20	0.41197	0.61905	0.49471	0.92742	0.84637	0.88504	0.82723
	LDA-50	0.22781	0.91005	0.36441	0.96737	0.46366	0.62687	0.58799
	LDA-BOW-20	0.20823	0.88360	0.33703	0.95359	0.41582	0.57912	0.54326
	LDA-BOW-30	0.18299	0.75132	0.29430	0.90600	0.41674	0.57089	0.52992
	LDA-BOW-50	0.21131	0.83069	0.33691	0.93996	0.46090	0.61852	0.57681
	mESA-1000	0.75234	0.85185	**0.79901**	0.97363	0.95124	**0.96231**	0.93812

Table C.13.: Results for imbalanced training data sets *crude* and *earn without* SMOTE.

		earn (2877/1087)			*grain* (433/149)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.95668	0.95492	0.95580	0.67550	0.68456	0.68000	0.92255
	LDA-20	0.90886	0.97240	0.93956	0.58904	0.28859	0.38739	0.87299
	LDA-50	0.93369	0.97148	0.95221	0.70476	0.49664	0.58268	0.90766
	LDA-BOW-20	0.91945	0.98712	0.95209	0.79710	0.36913	0.50459	0.89814
	LDA-BOW-30	0.90537	0.97700	0.93982	0.60317	0.25503	0.35849	0.86974
	LDA-BOW-50	0.95699	0.98252	0.96959	0.84167	0.67785	0.75093	0.94323
	mESA-1000	0.98370	0.99908	**0.99133**	0.99242	0.87919	**0.93238**	0.98422
Centroid	TFIDF	0.97661	0.80681	0.88363	0.37870	0.85906	0.52567	0.84048
	LDA-20	0.94456	0.83073	0.88399	0.34286	0.64430	0.44755	0.83138
	LDA-50	0.96440	0.84729	0.90206	0.40925	0.77181	0.53488	0.85779
	LDA-BOW-20	0.94794	0.40202	0.56460	0.16129	0.83893	0.27056	0.52915
	LDA-BOW-30	0.96529	0.79301	0.87071	0.34402	0.79195	0.47967	0.82357
	LDA-BOW-50	0.97333	0.40294	0.56994	0.17430	0.91946	0.29305	0.53656
	mESA-1000	0.99239	0.83993	**0.90982**	0.44937	0.95302	**0.61075**	0.87376
Bayes	TFIDF	0.91594	0.58142	0.71131	0.16667	0.61074	0.26187	0.65713
	LDA-20	0.94008	0.83717	0.88564	0.33955	0.61074	0.43645	0.83149
	LDA-50	0.93038	0.40570	0.56502	0.15223	0.77852	0.25467	0.52761
	LDA-BOW-20	0.94468	0.83257	0.88509	0.34532	0.64430	0.44965	0.83259
	LDA-BOW-30	0.95153	0.83073	0.88703	0.35889	0.69128	0.47248	0.83706
	LDA-BOW-50	0.95316	0.43054	0.59316	0.16913	0.84564	0.28188	0.55563
	mESA-1000	0.99483	0.88592	**0.93723**	0.53731	0.96644	**0.69065**	0.90750

Table C.14.: Results for imbalanced training data sets *earn* and *grain without* SMOTE.

		earn (2877/1087)			*interest* (347/131)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.89088	0.91628	0.90340	0.09000	0.06870	0.07792	0.81462
	LDA-20	0.89245	1.00000	0.94317	0.00000	0.00000	0.00000	0.84173
	LDA-50	0.91976	0.98068	0.94924	0.64407	0.29008	0.40000	0.89017
	LDA-BOW-20	0.89245	1.00000	0.94317	0.00000	0.00000	0.00000	0.84173
	LDA-BOW-30	0.90396	0.98712	0.94371	0.54839	0.12977	0.20988	0.86479
	LDA-BOW-50	0.93509	0.98068	0.95734	0.73077	0.43511	0.54545	0.91304
	mESA-1000	0.96967	1.00000	**0.98460**	1.00000	0.74046	**0.85088**	0.97022
Centroid	TFIDF	0.92137	0.42042	0.57738	0.12742	0.70229	0.21571	0.53849
	LDA-20	0.94486	0.80405	0.86879	0.27304	0.61069	0.37736	0.81593
	LDA-50	0.95582	0.87580	0.91407	0.39189	0.66412	0.49292	0.86877
	LDA-BOW-20	0.94526	0.82613	0.88169	0.29478	0.60305	0.39599	0.82945
	LDA-BOW-30	0.95412	0.84177	0.89443	0.33591	0.66412	0.44615	0.84621
	LDA-BOW-50	0.95253	0.86753	0.90804	0.36842	0.64122	0.46797	0.86071
	mESA-1000	1.00000	0.84545	**0.91625**	0.43813	1.00000	**0.60930**	0.88324
Bayes	TFIDF	0.89844	0.52898	0.66589	0.11419	0.50382	0.18618	0.61430
	LDA-20	0.92545	0.79945	0.85785	0.21864	0.46565	0.29756	0.79759
	LDA-50	0.94188	0.43238	0.59269	0.14186	0.77863	0.24000	0.55475
	LDA-BOW-20	0.93848	0.82797	0.87977	0.27799	0.54962	0.36923	0.82486
	LDA-BOW-30	0.94673	0.86661	0.90490	0.34978	0.59542	0.44068	0.85497
	LDA-BOW-50	0.94309	0.42686	0.58771	0.14187	0.78626	0.24037	0.55036
	mESA-1000	1.00000	0.90156	**0.94823**	0.55042	1.00000	**0.71003**	0.92261

Table C.15.: Results for imbalanced training data sets *earn* and *interest without* SMOTE.

		earn (2877/1087)			*money-fx* (538/179)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.87837	0.87029	0.87431	0.25397	0.26816	0.26087	0.78757
	LDA-20	0.85861	1.00000	0.92393	0.00000	0.00000	0.00000	0.79329
	LDA-50	0.91965	0.96872	0.94355	0.71901	0.48603	0.58000	0.89215
	LDA-BOW-20	0.91202	0.96320	0.93691	0.66102	0.43575	0.52525	0.87871
	LDA-BOW-30	0.92895	0.96228	0.94532	0.70714	0.55307	0.62069	0.89942
	LDA-BOW-50	0.93292	0.97240	0.95225	0.77444	0.57542	0.66026	0.91097
	mESA-1000	0.96444	0.99816	**0.98101**	0.98582	0.77654	**0.86875**	0.96514
Centroid	TFIDF	0.97436	0.41950	0.58650	0.20927	0.93296	0.34186	0.55191
	LDA-20	0.94033	0.78289	0.85442	0.34626	0.69832	0.46296	0.79907
	LDA-50	0.95035	0.86293	0.90453	0.46595	0.72626	0.56769	0.85691
	LDA-BOW-20	0.94869	0.83349	0.88737	0.41801	0.72626	0.53061	0.83692
	LDA-BOW-30	0.94851	0.88132	0.91369	0.49609	0.70950	0.58391	0.86706
	LDA-BOW-50	0.94900	0.87305	0.90944	0.48120	0.71508	0.57528	0.86219
	mESA-1000	1.00000	0.84821	**0.91787**	0.52035	1.00000	**0.68451**	0.88488
Bayes	TFIDF	0.92160	0.48666	0.63697	0.19364	0.74860	0.30769	0.59041
	LDA-20	0.91164	0.80681	0.85603	0.30921	0.52514	0.38923	0.79003
	LDA-50	0.92621	0.43882	0.59551	0.18775	0.78771	0.30323	0.55418
	LDA-BOW-20	0.93524	0.81049	0.86841	0.36420	0.65922	0.46918	0.81196
	LDA-BOW-30	0.92478	0.88224	0.90301	0.44105	0.56425	0.49510	0.84534
	LDA-BOW-50	0.93874	0.43698	0.59636	0.19474	0.82682	0.31523	0.55661
	mESA-1000	0.96481	0.95860	**0.96170**	0.75806	0.78771	**0.77260**	0.93496

Table C.16.: Results for imbalanced training data sets *earn* and *money-fx without* SMOTE.

		earn (2877/1087)			*trade* (369/116)			
Classifier	Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Average
SVM	TFIDF	0.97285	0.98896	0.98084	0.87755	0.74138	0.80374	0.96376
	LDA-20	0.93838	0.98068	0.95906	0.68657	0.39655	0.50273	0.91506
	LDA-50	0.95845	0.97608	0.96718	0.72917	0.60345	0.66038	0.93760
	LDA-BOW-20	0.91304	0.98528	0.94779	0.46667	0.12069	0.19178	0.87489
	LDA-BOW-30	0.94818	0.95952	0.95382	0.57282	0.50862	0.53881	0.91380
	LDA-BOW-50	0.96664	0.98620	0.97632	0.84043	0.68103	0.75238	0.95473
	mESA-1000	0.99451	1.00000	**0.99725**	1.00000	0.94828	**0.97345**	0.99495
Centroid	TFIDF	0.99394	0.75437	0.85774	0.29365	0.95690	0.44939	0.81837
	LDA-20	0.96677	0.82981	0.89307	0.31481	0.73276	0.44041	0.84942
	LDA-50	0.97720	0.86753	0.91910	0.39496	0.81034	0.53107	0.88169
	LDA-BOW-20	0.97158	0.84913	0.90623	0.35178	0.76724	0.48238	0.86536
	LDA-BOW-30	0.97801	0.85925	0.91479	0.38306	0.81897	0.52198	0.87691
	LDA-BOW-50	0.97619	0.45262	0.61848	0.14878	0.89655	0.25521	0.58345
	mESA-1000	1.00000	0.84453	**0.91571**	0.40702	1.00000	**0.57855**	0.88320
Bayes	TFIDF	0.93775	0.85925	0.89678	0.26087	0.46552	0.33437	0.84255
	LDA-20	0.89135	0.40754	0.55934	0.08782	0.53448	0.15085	0.51995
	LDA-50	0.94027	0.44894	0.60772	0.12427	0.73276	0.21250	0.56961
	LDA-BOW-20	0.96480	0.85741	0.90794	0.34599	0.70690	0.46459	0.86519
	LDA-BOW-30	0.93116	0.47286	0.62721	0.11982	0.67241	0.20339	0.58634
	LDA-BOW-50	0.95094	0.46366	0.62338	0.13373	0.77586	0.22814	0.58527
	mESA-1000	0.99519	0.95216	**0.97320**	0.68098	0.95690	**0.79570**	0.95609

Table C.17.: Results for imbalanced training data sets *earn* and *trade without* SMOTE.

C.3. Different Results for SMOTE with Variable N Parameter and SVM Classifier

	acq			*crude*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.95839	0.92907	0.94350	0.75829	0.84656	0.80000	100 / 400
LDA-20	0.87246	0.89430	0.88324	0.55556	0.50265	0.52778	100 / 500
LDA-50	0.92000	0.92768	0.92382	0.71585	0.69312	0.70430	100 / 600
LDA-BOW-20	0.91884	0.92907	0.92393	0.71823	0.68783	0.70270	100 / 500
LDA-BOW-30	0.91061	0.90682	0.90871	0.65104	0.66138	0.65617	100 / 500
LDA-BOW-50	0.92033	0.93185	0.92605	0.72778	0.69312	0.71003	100 / 500
mESA-1000	0.94667	0.98748	**0.96664**	0.94304	0.78836	**0.85879**	200 / 500

Table C.18.: Results for imbalanced training data sets *acq* and *crude with* SMOTE and SVM classifier.

	acq			*grain*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.96721	0.90264	0.93381	0.64467	0.85235	0.73410	200 / 400
LDA-20	0.90028	0.87900	0.88951	0.47590	0.53020	0.50159	100 / 500
LDA-50	0.91460	0.92350	0.91903	0.61268	0.58389	0.59794	100 / 500
LDA-BOW-20	0.90504	0.87483	0.88967	0.47977	0.55705	0.51553	100 / 500
LDA-BOW-30	0.92910	0.87483	0.90115	0.52880	0.67785	0.59412	100 / 500
LDA-BOW-50	0.95359	0.94298	0.94825	0.73885	0.77852	0.75817	100 / 500
mESA-1000	0.97811	0.99444	**0.98621**	0.97080	0.89262	**0.93007**	100 / 300

Table C.19.: Results for imbalanced training data sets *acq* and *grain with* SMOTE and SVM classifier.

	acq			*interest*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.95897	0.87761	0.91649	0.54167	0.79389	0.64396	600 / 400
LDA-20	0.86129	0.94993	0.90344	0.36842	0.16031	0.22340	100 / 400
LDA-50	0.91102	0.89708	0.90399	0.47887	0.51908	0.49817	100 / 600
LDA-BOW-20	0.90780	0.89013	0.89888	0.45517	0.50382	0.47826	100 / 600
LDA-BOW-30	0.91915	0.90125	0.91011	0.51034	0.56489	0.53623	100 / 500
LDA-BOW-50	0.91226	0.91099	0.91162	0.51515	0.51908	0.51711	100 / 400
mESA-1000	0.95861	0.99861	**0.97820**	0.99010	0.76336	**0.86207**	100 / 500

Table C.20.: Results for imbalanced training data sets *acq* and *interest with* SMOTE and SVM classifier.

	acq			*trade*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.99247	0.91655	0.95300	0.64912	0.95690	0.77352	100 / 600
LDA-20	0.93130	0.84840	0.88792	0.39444	0.61207	0.47973	100 / 500
LDA-50	0.94266	0.93741	0.94003	0.62500	0.64655	0.63559	100 / 500
LDA-BOW-20	0.92296	0.84979	0.88487	0.37572	0.56034	0.44983	100 / 400
LDA-BOW-30	0.95077	0.94019	0.94545	0.65323	0.69828	0.67500	100 / 500
LDA-BOW-50	0.95455	0.93463	0.94448	0.64122	0.72414	0.68016	100 / 600
mESA-1000	0.99033	0.99722	**0.99376**	0.98198	0.93966	**0.96035**	100 / 600

Table C.21.: Results for imbalanced training data sets *acq* and *trade with* SMOTE and SVM classifier.

	crude			*earn*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.22660	0.48677	0.30924	0.88851	0.71113	0.78998	100 / 600
LDA-20	0.71111	0.50794	0.59259	0.91849	0.96412	0.94075	100 / 600
LDA-50	0.79769	0.73016	0.76243	0.95376	0.96780	0.96073	100 / 600
LDA-BOW-20	0.58525	0.67196	0.62562	0.94145	0.91720	0.92917	100 / 500
LDA-BOW-30	0.68831	0.56085	0.61808	0.92602	0.95584	0.94070	100 / 600
LDA-BOW-50	0.80117	0.72487	0.76111	0.95294	0.96872	0.96077	100 / 600
mESA-1000	0.97419	0.79894	**0.87791**	0.96610	0.99632	**0.98098**	100 / 500

Table C.22.: Results for imbalanced training data sets *crude* and *earn with* SMOTE and SVM classifier.

	earn			*grain*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.97551	0.87948	0.92501	0.48828	0.83893	0.61728	100 / 500
LDA-20	0.93993	0.93560	0.93776	0.54545	0.56376	0.55446	200 / 400
LDA-50	0.94374	0.95676	0.95021	0.64925	0.58389	0.61484	100 / 500
LDA-BOW-20	0.95379	0.94940	0.95159	0.64286	0.66443	0.65347	100 / 500
LDA-BOW-30	0.94236	0.90248	0.92199	0.45641	0.59732	0.51744	100 / 600
LDA-BOW-50	0.96451	0.97516	0.96981	0.80292	0.73826	0.76923	100 / 500
mESA-1000	0.98636	0.99816	**0.99223**	0.98529	0.89933	**0.94035**	200 / 600

Table C.23.: Results for imbalanced training data sets *earn* and *grain with* SMOTE and SVM classifier.

	earn			*trade*			
Features	Precision	Recall	F1 Score	Precision	Recall	F1 Score	N/P
TFIDF	0.98874	0.88868	0.93605	0.46460	0.90517	0.61404	300 / 600
LDA-20	0.95837	0.95308	0.95572	0.58197	0.61207	0.59664	100 / 600
LDA-50	0.96425	0.96780	0.96602	0.68750	0.66379	0.67544	100 / 600
LDA-BOW-20	0.94845	0.93100	0.93965	0.44853	0.52586	0.48413	100 / 600
LDA-BOW-30	0.96352	0.94756	0.95547	0.57463	0.66379	0.61600	100 / 600
LDA-BOW-50	0.97083	0.97976	0.97527	0.79245	0.72414	0.75676	100 / 600
mESA-1000	0.99360	0.99908	**0.99633**	0.99091	0.93966	**0.96460**	100 / 300

Table C.24.: Results for imbalanced training data sets *earn* and *trade with* SMOTE and SVM classifier.

D. LDA Topics generated from articles

Table D.1 shows the LDA topics learned from all news articles. The ten tokens with the biggest likelihood are displayed. They are tagged with their respective word class.

Table D.1.: Latent Dirichlet Allocation Topics and their Members.

Topics #	Members
0	studio/NN, blackberry/NN, ea/NN, rim/NN, photographer/NN, guild/NN, mmo/NN, trademark/NN, honor/NN, french/JJ
1	project/NN, open/JJ, source/NN, development/NN, community/NN, software/NN, develop/VB, problem/NN, designer/NN, code/NN
2	report/NN, issue/NN, law/NN, government/NN, statement/NN, report/VB, policy/NN, public/JJ, internet/NN, privacy/NN
3	percent/NN, quarter/NN, sale/NN, revenue/NN, third/JJ, report/VB, percent/VB, growth/NN, stock/NN, earning/NN
4	keyboard/NN, battery/NN, audio/JJ, wireless/JJ, touch/NN, headphone/NN, speaker/NN, control/NN, sound/NN, cover/NN
5	sony/NN, xbox/NN, student/NN, ps/NN, playstation/NN, school/NN, education/NN, pc/NN, live/NN, university/NN
6	steam/NN, sept/NN, valve/NN, combat/NN, mini/VB, gameplay/NN, pc/NN, half/NN, title/NN, gaming/NN
7	courtesy/NN, tweet/NN, pin/NN, youtube/NN, view/NN, comment/NN, obama/NN, image/VB, popular/JJ, reddit/NN
8	io/NN, browser/NN, option/NN, email/NN, application/NN, download/VB, update/VB, interface/NN, text/NN, list/NN
9	hp/NN, art/NN, disney/NN, artist/NN, flood/VB, op/NN, artist/VB, smith/NN, concept/NN, film/NN
10	samsung/NN, galaxy/NN, note/NN, iii/VB, vs/NN, apple/VB, pen/VB, tab/NN, trial/NN, forstall/NN

Continued on next page

Topic #	Members
11	cloud/NN, nook/NN, software/NN, enterprise/NN, server/NN, application/NN, storage/NN, barne/NN, hr/VB, solution/NN
12	summit/VB, president/NN, ceo/NN, founder/NN, chief/JJ, vice/NN, marketing/NN, executive/NN, officer/NN, manager/NN
13	book/NN, star/NN, wars/NN, edition/NN, final/JJ, fantasy/NN, borderland/NN, title/NN, fan/NN, dragon/NN
14	country/NN, china/NN, uk/NN, europe/NN, japan/NN, united/NN, london/NN, internet/NN, european/NN, chinese/NN
15	wii/NN, nintendo/NN, gaming/NN, console/NN, super/JJ, japanese/NN, href/NN, game/VB, gamer/NN, mario/NN
16	feel/VB, don/NN, didn/VB, ve/NN, enough/RB, probably/RB, quite/RB, if/VB, never/RB, doesn/NN
17	life/NN, woman/NN, man/NN, love/VB, guy/NN, talk/VB, like/VB, family/NN, learn/VB, fun/NN
18	car/NN, pm/NN, driver/NN, lightning/NN, winner/NN, warren/NN, ticket/NN, uber/NN, vehicle/NN, contest/NN
19	patent/NN, court/NN, rt/NN, mashable/NN, judge/NN, patent/VB, file/VB, legal/JJ, lawsuit/NN, claim/NN
20	amazon/NN, fire/NN, com/NN, http/NN, kindle/NN, allthingsd/NN, kindle/JJ, atd/VB, amazon/VB, book/NN
21	halloween/NN, assassin/NN, nba/NN, normally/JJ, saving/VB, coupon/NN, gb/NN, iii/NN, code/NN, curiosity/NN
22	map/NN, fi/NN, wi/JJ, travel/NN, view/NN, map/VB, location/NN, pic/NN, flight/NN, street/NN
23	character/NN, enemy/NN, robot/NN, comic/JJ, adventure/NN, man/NN, weapon/NN, monster/NN, trailer/NN, war/NN
24	yahoo/NN, question/NN, mayer/NN, word/NN, soul/NN, ask/VB, bad/JJ, batman/NN, lord/NN, write/VB
25	email/NN, security/NN, account/NN, promo/NN, message/NN, send/VB, name/NN, hacker/NN, internet/NN, password/NN
26	card/NN, payment/NN, pay/VB, credit/NN, money/NN, square/NN, fee/NN, wallet/NN, transaction/NN, paypal/JJ
27	html/NN, weather/NN, notebook/NN, jerk/NN, wind/NN, electric/JJ, bike/NN, smart/JJ, emergency/NN, alexi/NN
28	nokia/NN, lte/NN, htc/NN, carrier/NN, lg/NN, verizon/NN, lumia/NN, wireless/JJ, motorola/NN, handset/NN

Continued on next page

Topic #	Members
29	power/NN, speed/NN, energy/NN, cost/NN, use/NN, solution/NN, industry/NN, deliver/VB, improve/VB, develop/VB
30	sport/NN, venturebeat/NN, nfl/NN, football/NN, island/NN, season/NN, espn/NN, america/NN, latin/NN, fbi/NN
31	zynga/NN, debate/NN, bill/NN, house/NN, state/NN, fail/NN, convention/NN, speech/NN, gaming/NN, party/NN
32	drive/NN, mac/NN, pc/NN, laptop/NN, ssd/NN, computer/NN, macbook/NN, machine/NN, hard/JJ, desktop/NN
33	ipad/NN, mini/NN, apple/VB, ipod/NN, inch/NN, display/NN, retina/NN, generation/NN, cook/NN, touch/NN
34	tv/NN, music/NN, movie/NN, stream/VB, watch/VB, show/NN, live/NN, channel/NN, itune/NN, netflix/NN
35	window/NN, microsoft/NN, phone/VB, surface/NN, rt/VB, window/VB, pc/NN, os/NN, tile/NN, october/VB
36	water/NN, kickstarter/NN, nano/NN, project/NN, previous/NN, light/NN, sandy/NN, blue/JJ, carbon/NN, double/JJ
37	gb/NN, hd/NN, inch/NN, display/NN, core/NN, processor/NN, performance/NN, camera/NN, chip/NN, resolution/NN
38	gallery/NN, kid/NN, px/NN, img/NN, margin/NN, child/NN, left/VB, width/NN, tumblr/NN, auto/NN
39	city/NN, reception/NN, new/NN, instagram/NN, york/NN, conference/NN, san/NN, francisco/NN, theme/NN, location/NN
40	mashable/JJ, new/NN, york/NN, editor/NN, digital/JJ, magazine/NN, lab/NN, talk/NN, league/NN, editor/VB
41	friend/NN, family/NN, gun/NN, logo/NN, bond/NN, font/NN, tag/VB, tag/NN, farm/NN, friend/VB
42	order/NN, pre/NN, retailer/NN, retail/JJ, sale/NN, originally/JJ, com/NN, purchase/VB, item/NN, digital/JJ
43	player/NN, game/VB, level/NN, mode/NN, zombie/NN, multiplayer/NN, play/NN, rpg/NN, effect/NN, action/NN
44	post/NN, post/VB, question/NN, profile/NN, blog/NN, change/NN, party/NN, important/JJ, third/JJ, ask/VB
45	startup/NN, founder/NN, venture/NN, investor/NN, raise/VB, funding/NN, partner/NN, investment/NN, round/NN, capital/NN
46	camera/NN, color/NN, shot/NN, picture/NN, hurricane/NN, capture/VB, music/NN, cat/NN, lens/NN, view/NN

Continued on next page

Topic #	Members
47	nexus/NN, update/NN, bean/NN, update/VB, sprint/NN, mod-/JJ, jelly/NN, update/JJ, ice/NN, official/NN
48	ad/NN, search/NN, brand/NN, advertising/NN, result/NN, marketing/NN, target/VB, campaign/NN, engagement/NN, audience/NN
49	space/NN, science/NN, researcher/NN, university/NN, health/NN, nasa/NN, scientist/NN, research/NN, solar/JJ, earth/NN

List of Figures

List of Tables

List of listings

Bibliography

[1] D. Beymer, D. Russell, and P. Orton. "An eye tracking study of how font size and type influence online reading." In: *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction-Volume 2*. British Computer Society. 2008, pp. 15–18.

[2] D. Billsus and M.J. Pazzani. "User modeling for adaptive news access." In: *User modeling and user-adapted interaction* 10.2 (2000), pp. 147–180.

[3] David Blei. *Topic Models*. Sept. 2009. URL: `http://videolectures.net/mlss09uk_blei_tm/`.

[4] D.M. Blei, A.Y. Ng, and M.I. Jordan. "Latent dirichlet allocation." In: *The Journal of Machine Learning Research* 3 (2003), pp. 993–1022.

[5] E. Brill. "A simple rule-based part of speech tagger." In: *Proceedings of the Workshop on Speech and Natural Language*. Association for Computational Linguistics. 1992, pp. 112–116.

[6] A. Budanitsky and G. Hirst. "Evaluating wordnet-based measures of lexical semantic relatedness." In: *Computational Linguistics* 32.1 (2006), pp. 13–47.

[7] N.V. Chawla et al. "SMOTE: synthetic minority over-sampling technique." In: *arXiv preprint arXiv:1106.1813* (2011).

[8] T. De Smedt and W. Daelemans. "Pattern for Python." In: *The Journal of Machine Learning Research* 98888 (2012), pp. 2063–2067.

[9] F. Debole and F. Sebastiani. "An analysis of the relative hardness of Reuters-21578 subsets." In: *Journal of the American Society for Information Science and Technology* 56.6 (2005), pp. 584–596.

[10] S. Deerwester et al. "Indexing by latent semantic analysis." In: *Journal of the American society for information science* 41.6 (1990), pp. 391–407.

[11] Elika J. Etemad. *CSS*. W3C Working Draft. W3C, 2010. URL: `http://www.w3.org/TR/2011/NOTE-css-2010-20110512/`.

[12] L. Finkelstein et al. "Placing search in context: The concept revisited." In: *Proceedings of the 10th international conference on World Wide Web*. ACM. 2001, pp. 406–414.

[13] Lucas Franek. "Ensemble algorithms with applications to clustering and image segmentation." PhD thesis. University of Münster, 2012.

[14] E. Gabrilovich and S. Markovitch. "Computing semantic relatedness using wikipedia-based explicit semantic analysis." In: *Proceedings of the 20th international Joint Conference on Artifical Intelligence*. 2007, pp. 1606–1611.

[15] E. Gabrilovich and S. Markovitch. "Wikipedia-based semantic interpretation for natural language processing." In: *Journal of Artificial Intelligence Research* 34.2 (2009), p. 443.

[16] E. H. Han and G. Karypis. "Centroid-based document classification: Analysis and experimental results." In: *Principles of Data Mining and Knowledge Discovery* (2000), pp. 116–123.

[17] H. Han, W. Y. Wang, and B. H. Mao. "Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning." In: *Advances in Intelligent Computing* (2005), pp. 878–887.

[18] H. He and E.A. Garcia. "Learning from imbalanced data." In: *Knowledge and Data Engineering, IEEE Transactions on* 21.9 (2009), pp. 1263–1284.

[19] T. Hofmann. "Unsupervised learning by probabilistic latent semantic analysis." In: *Machine Learning* 42.1 (2001), pp. 177–196.

[20] David Hyatt and Ian Hickson. *HTML 5*. W3C Working Draft. `http://www.w3.org/TR/2009/WD-html5-20090825/`. W3C, Aug. 2009.

[21] M. Jarmasz. "Roget's thesaurus as a lexical resource for natural language processing." In: *arXiv preprint arXiv:1204.0140* (2012).

[22] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: `http://www.scipy.org/`.

[23] S. Lee et al. *Intelligent Autonomous Systems 12*. Springer, 2012.

[24] D.D. Lewis and W.A. Gale. "A sequential algorithm for training text classifiers." In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc. 1994, pp. 3–12.

[25] W.G. Lycan. *Philosophy of language: A contemporary introduction*. Routledge, 2012.

[26] Travis E. Oliphant. *Guide to NumPy*. Provo, UT, Mar. 2006. URL: `http://www.tramy.us/`.

[27] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] Radim Rehurek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora." In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 22, 2010, pp. 45–50.

[29] G. Salton and M.J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, Inc., 1986.

[30] D. Thomas, C. Fowler, and A. Hunt. *Programming Ruby*. Vol. 13. Pragmatic Bookshelf, 2004.

[31] A.M. Turing. "Computing machinery and intelligence." In: *Mind* 59.236 (1950), pp. 433–460.

[32] T. Zesch, C. Müller, and I. Gurevych. "Extracting lexical semantic knowledge from Wikipedia and Wiktionary." In: *Proceedings of the Conference on Language Resources and Evaluation (LREC)*. Vol. 15. 2008, p. 60.

[33] H. Zhang. "The optimality of naive Bayes." In: *A A* 1.2 (2004), p. 3.

Eidesstattliche Erklärung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Aussagen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keinerlei anderer als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Vorname Nachname, Münster, January 23, 2013