Non-domain specific and translated sentiment analysis
- Local lexical analyzer

Master thesis

Author:

Erik Kringstad Olsen

Date:

June 1, 2015

Abstract

There is an ever-growing amount of opinionated data available on the Web, in form of reviews, discussions and blogs. This data can potentially provide a lot of information through sentiment analysis and data mining in general. However, most research in the field of sentiment analysis has been locked to a single language and a single domain. Thus, the main objective of this thesis is to answer the question:

"How can a sentiment analysis tool that is independent of domain and language be developed?"

The aim of this thesis is to show the possibilities of a sentiment analysis program independent of languages and domain, capable of being used in different languages and domains without added effort.

A sentiment analysis program was developed to test the feasibility of sentiment analysis across different domains or in languages other than English. Three main methods of sentiment analysis were implemented into the program, with tests being run on three different datasets. These three methods are a single weight analysis method where a documents sentiment equals the sum of its words sentiment, a sentence analyzer where a document is analyzed sentence by sentence, and a co-occurrence analyzer operating on the assumption that words occurring together often share the same sentiment value.

The results shown are not able to achieve the same quality as those shown in other published articles, and in the case of the non-English analysis, are inconclusive. However the possible viability of a completely resourceless sentiment analysis program is shown. Further research and improvement is needed to achieve better results.

Acknowledgments

I wish to thank my supervisor, Weiqin Chen, for her constant help and suggestions for the thesis, and her invaluable insight. I would also like to thank my fellow student, Ken-Thomas Nilsen, for bearing with me for the duration of the thesis, always willing to provide useful comments. And finally I wish to thank Irena Vašíčková for always being there for me, and supporting me through the toughest and easiest parts of the work.

Table of Contents

1. Introduction

For each year that passes, the amount of information available freely online increases. Not all information is inherently useful, but there is still so much that could possibly be learnt from the enormous amount available. The field of data mining is the logical answer to the opportunity presented in the sheer mass of data. Data mining presents the tools necessary to specific pieces of information from massive or general data. It can be used to find patterns that are not visible to humans, such as mining patient records to find connections between them to help future diagnosis, or finding the specific shopping patterns of different generations or genders to more effectively be able to target marketing.

The field of data mining is not a new one, but rather a field that continuously expands reflecting the way new data is generated. One of the more recent generators of large amounts of data is social media. Opinions, reviews, debates and more are shared by an enormous amount of people across several different social platforms. Opinion mining, or sentiment analysis, is one of the more modern ways of handling this new trove of data available. The terms 'opinion mining' and 'sentiment analysis' first appeared in the early 2000's, and have grown in relevance almost in conjunction with the rise of social media (Liu, 2012, page 7).

A lot of research has been done in the field of sentiment analysis, with goals of mining opinions from reviews or automatically creating sentiment lexicons from thesauri or different sources. Some of the biggest challenges still remaining in sentiment analysis are the ability to use one sentiment analysis application across several different domains, and the ability to functionally use sentiment analysis across language barriers.

Cross-domain sentiment analysis differs from the fields general focus, in the fact that it, as the name suggests, attempts to analyze data across different domains.

This thesis will dominantly focus on sentiment analysis as a tool that does not necessarily have to be specially adjusted to fit a certain domain, but rather a tool that can be applied in any domain or language where opinionated data can be found.

1.1 Research questions

The aim of this thesis is to research the topic of sentiment analysis and how it can be applied across different domains and languages. To further examine this topic I formulated the following research questions:

RQ: "How can a sentiment analysis tool that is independent of domain and language be developed?"

This question covers the overarching goal of the thesis. As explained in the introduction, the ever increasing mass of data is not purely locked in a single domain of knowledge, nor is it locked to a single language. To answer this question I have decided to divide it into smaller parts. They are as follows:

R1: "What method of sentiment analysis provides the highest percentage of accuracy?"

In order to answer this question I have developed a program that utilizes several different methods of sentiment analysis and then analyzed the difference in results. One method is not necessarily universally better than others, so the accuracy of each method was tested on different datasets.

R2: "How feasible is sentiment analysis across specific domains?"

One of the goals of the thesis is to look into whether or not using sentiment analysis across several different domains is feasible, compared to training a program for each specific domain.

R3: “How viable are cheaper[1] methods of translated sentiment analysis viable?”

There are several ways translated sentiment analysis can be done, and one of the goals of this thesis is to see whether the cheaper methods are viable options compared to the more expensive[2] ones.

To answer these questions I have developed a sentiment analysis prototype program, called Local Lexical Analyzer (LLA). Most existing sentiment analysis projects use external sources to analyze their given datasets or domains, whereas LLA will only depend on the datasets themselves. This means that LLA functions completely without any sentiment lexicons or other similar tools. This was done specifically because these sentiment lexicons are for the most part tailored to a specific domain, and exist dominantly in English.

1.2 Thesis overview

The thesis is divided into five individual chapters. The first chapter contains the introduction for the thesis, as well as the research questions guiding the design and development of the project.

The second chapter covers the topic of sentiment analysis, succinctly explaining what the field is about, as well as covering problems within the field of sentiment analysis. In addition it contains relevant theory from the field, highlighting some of the research already performed relevant to the thesis.

The third chapter is about the design and development of the LLA. First a quick overview of supervised learning and how this will be used in the development is presented. Secondly it covers the development process and testing under development for each of the different algorithms used in the project, showing results and a rundown and comparison of each of the algorithms. And finally a brief summary of the tools used during development.

[1] Cheaper in this context has two meanings. Firstly if it is less demanding in development, and secondly if it is potentially a faster running operation.
[2] Expensive inverts the meanings above.

The fourth chapter covers the evaluation of the LLA. Comparing development data to the results from the new Norwegian dataset, as well as cross-domain data. It also includes an in depth discussion of the results.

The fifth chapter is the conclusion. This chapter covers my reflections on the entire process, and reviews the goals from the introduction. This chapter also presents suggestions for future development in the Local Lexical Analyzer, as well as improvements and suggestions for further research.

2. Sentiment analysis

Sentiment in the case of sentiment analysis is, as defined by the Merriam-Webster online dictionary “an attitude, thought, or judgment prompted by feeling”. The sentence ‘I love you.’ is a positively loaded sentence with the subject ‘you’. It is easily seen that the actor in the sentence is ‘I’ and that the actor is positively inclined towards the subject. For humans capable of understanding the sentence above this seems obvious, however a machine will not know this. The text editor used to write this thesis has no understanding of the words written or, for that matter, the sentiment behind them. Without a set of rules a machine would interpret the sentence ‘This is a potato.’ as just as romantic as ‘I love you.’. So the basic problem of sentiment analysis, and consequently text analysis as a whole is that a program needs to be taught how to ‘understand’ human language. This is of course not a trivial course of action, and the understanding will not be perfect.

For a program to be able to ‘interpret’ a text it needs a set of rules of interpretation. Interpretation will in this case be considered that a program can be given a text and return significant information about the content of the text, more specifically the sentiment of the text. A simple example of very basic sentiment analysis would be to give a program a set of positive words and a set of negative words, then let it compare a given document to these words. For each positive hit it increases the weight of the document, and for each negative hit it decreases it. If the total weight is positive after this process the document is classified as positive, and vice versa. This is a very straightforward way of implementing sentiment analysis, but also quite naive. A text can be hugely positive or negative without including the most obvious sentiment-giving words. A text can also be sarcastic, or include negating words like “isn’t” or “doesn’t”. There are ways of avoiding these problems, or at least working around them with sentiment inversion in the case of negation and through algorithms meant for detecting sarcasm, though 100% accuracy is in most cases unachievable.

There are several known algorithms for sentiment analysis, such as weighting the sentiment of an entire document to weighting each individual sentence of a document (Feldman, 2013). For this thesis several of the known algorithms will be used in the development of a sentiment analysis program prototype, though some diversions from the traditional methods will be employed, explained in more detail later.

An obvious part of developing a sentiment analysis program is to measure whether or not it actually manages to classify documents correctly based on their sentiment, or to be more precise, if the program manages to classify a document in the same fashion as a human. In many fields subjectivity can pose a problem, as human interpretation of a document can vary to a large extent. There is however a promising trend in the field of Natural Language Processing and existing analytics programs where the classification is already done within reasonable limits of human classification (Bermingham et al., 2009)

2.1 Complications in the field of sentiment analysis

Automatic text analysis is in no way perfect. There are a lot of complications that prevents a 100% in accuracy. One of the perhaps more modern, and difficult, problems to be found in documents is sarcasm and other similar ways of using words to portray something other than their literal meaning (Liu, 2012, page 52). As an example, consider descriptions and reviews found in online stores. These reviews are often laden with sarcasm and as such a sentiment analysis program would struggle to correctly handle these instances. There are ways of avoiding these problems, as described by Tsur, Davidov and Rappoport (2010), who developed an algorithm for detecting sarcasm in text. But as the goal of this thesis is to create a tool to be used on a wide variety of texts, it is unknown as to what degree sarcasm and wordplay will prove to be present, and as such there has been made no implementation to consider sarcasm. Wordplay is a very interesting topic in the field of text analysis, worthy of further research, but outside the scope of this thesis.

2.2 Current sentiment analysis methods

Prabowo og Thelwall (2009) suggests a wide range of ways to perform sentiment analysis. Amongst these is a method to automatically generate a list of words with a predefined sentiment, called rule-based classification. Given an already classified training set with a close to equal spread of positive and negative documents we can give each word in the documents a positive or negative weight, depending on which type of document they occur in. This way it is possible to match a new document against the weights aggregated by the training and thereby classifying it as either positive or negative (Prabowo and Thelwall, 2009). A problem with this method, of course, is that words do not exclusively occur in positive or negative documents. It is likely to find positive words in negative texts and the other way around. This can be balanced by assigning each word a weight based on their total number of positive and negative occurrences. So a word that occurs 5 times in positive documents and 3 times in negative documents would in this case have a weight of 2. Given a large enough training set this should ensure that words that humans effortlessly see as positive or negative will be weighted in this fashion. Another possible example is statistic based classification (Prabowo and Thelwall, 2009). This method requires that there already exists a list of positive and negative words, to be used to expand a sentiment wordlist, known as a sentiment lexicon. The idea behind this is to see which words most often occur together with the words in the existing list, and adding the most frequent ones, thus expanding the sentiment-weighted lexicon. (Prabowo and Thelwall, 2009). The development of the Local Lexical Analyzer will implement the former of these two methods, in an attempt to create a sentiment analysis program fully independent of outside lexical knowledge that can be difficult to gather in languages in which Natural Language Processing has not seen a lot of use. This is not to say that a list of stopwords, or an algorithm for stemming won’t be useful, as explained in chapter three.

2.3 Translated sentiment analysis

As stated previously, one of the main goals of this thesis is to produce a sentiment analysis program capable of being used in every language compatible with the UTF8[3] typeset. There are a few different approaches to translated sentiment analysis, and the three suggested possibilities of translated sentiment analysis will be explained below, with their advantages and disadvantages listed. For this thesis the intention is to test the feasibility of one of the cheapest methods of translated sentiment analysis, which requires no extra prior work on the datasets to be used, unlike the two other possible methods.

2.3.1 Machine translated sentiment analysis

The base concept of machine translated analysis is very simple. Shortly explained the process only entails translating all the texts to be classified by means of a machine translation program (eg: google translate), and then running the sentiment analysis program in it's original settings on the translated text.

Drawbacks and limitations

This method's success is very dependent on the type of text being translated. If we assume the translation of professional texts then we can also assume a certain level of professionalism in proofreading and correcting of said texts. But, if we use less professional sources, twitter messages for example, we have to assume spelling errors (both intended and unintended), slang and inconsistent capitalization and punctuation (smileys). These inconsistencies in the text will for the most part leave the parts that aren't comprehensible to the machine translator untranslated. What this means is that the chance of an accurate analysis decreases, as we can't know if the untranslated words are important to the overall sentiment or not without going through the text by hand. So the more anomalies occurring in the target text, the bigger the chance of inaccurate analysis becomes.

[3] "8-bit Unicode Transformation Format. A method of encoding Unicode codepoints using one-byte unsigned integers; from one to four such integers are used depending on the codepoint. UTF-8 is the most widely used Unicode encoding scheme and is the default encoding for XML documents." From A Dictionary Of Computing (2008)

In addition the inherent weakness of machine translation in regards to spelling errors, we also have to consider the cost of using machine-translated text. Rather than the translation process being done once, as the next method will show, this requires us to translate every single piece of text that is to be classified. For smaller sets of texts this is negligible, but the bigger the set of texts to be classified becomes, the more significant the operation time of translation becomes.

Advantages

One of the main benefits of this method is that the program can remain unadjusted as soon as the target language is discovered. After we have discovered the target language it is a simple case of translating the text(s) into the program default language (English) and running the analysis. This prevents a lot of potential implementation hurdles in different languages, as things such as stop-word lists, stemming and lemmatization in different languages becomes unnecessary.

2.3.2 Dual wordlists

The concept behind the dual wordlist is quite similar to the machine translated idea, but rather than translating each individual piece of text to match the English wordlist, each word in the wordlist is translated to match the individual languages. This requires an existing wordlist of a known language.

Drawbacks and limitations

Unlike in the machine translated method, this method requires significant changes to the settings of the program. This should for the most case be trivial, but it completely depends on the language in question. One of the main problems being that stemming and stopword removal has to be done both on the non-English text and on the translated wordlist, or neither if this is not possible in the target language. Depending on the language this could either be trivial, if the algorithms for stemming and a list of stopwords are already developed and readily available for the given language, but this is in many languages not the case. As explained earlier, the LLA is supposed to function without the addition of resources such as stemming and stopword lists, but this does not mean that it would not function significantly better with these resources.

Advantages

The main advantage of this method is that the unclassified texts can remain unprocessed before the program is run, as the only translation takes place on the existing, English, wordlist. This can save a considerable amount of time compared to translating every single piece of text.

2.3.3 LLA-generated wordlists

Rather than translating the target texts, or translating the wordlist by using machine translation, there is a third possibility, and that is to not translate at all. Given the algorithms for stemming and a stopword list in a given language, it will in theory be unproblematic to use the LLA in any given language (provided they use the UTF-8 typeset). The process for generating the wordlists will be presented in chapter 3, but in short, the LLA simply goes through a training set of documents to generate a list of positive and negative words. Unlike the previous two methods, this method will generate a new wordlist for each given language by itself, rather than using any machine translation.

Drawbacks and limitations

The main drawback of this method is quite clear, development time. For the most part, all languages used in common speech, are unique in some way, and different enough from each other that they are not (completely) interchangeable. This means that a Norwegian document can not be properly analyzed using an English word set. That is to say that you need to find the algorithms for stemming and a stopword list in those languages for maximum classification accuracy. However, the method is still applicable without these resources.

Advantages

The main advantage of this method is that context and language specific quirks do not create any trouble. Unlike the two other methods, since the wordlist is generated from the same type of documents from the same language, it is possible to assume that the documents in question are in some way similar, that is to say that they are in the same domain. In addition, since the wordlist is created and used in the same language, the patterns of the language which might be discarded during machine translation are kept intact, as the texts and their content are not changed (save for any potential stemming and stopword removal). Thus the context of the documents can be assumed to remain intact.

2.4 Related works

The field of sentiment analysis is enormous, and just like the ever growing amount of data available, the amount of studies and projects done in sentiment analysis keeps on accruing. This section will provide a short overview of work done in the field, especially focusing on translated and domain-specific sentiment analysis.

2.4.1 Sentiment in news

Most of today’s research on sentiment analysis has been done on reviews or similar texts (Balahur et al., 2003), where the sentiment of the content is quite explicit. This makes sense, as it makes the analysis that much easier when the sentiment of the reviews are easily identified. However, this is not a justifiable reason to simply constrain all sentiment analysis to content where the sentiment is self-evident. An example of a text where the sentiment in the text is not (always) self-evident would be news and newspapers in general. There are of course exceptions, but as a rule the ones writing the news should at the very least make an attempt at a neutral tone. But the fact that a piece of news is written in a neutral tone does not exclude the news from being sad or happy, catastrophic or joyous. This does however pose a challenge to most sentiment analysis methods currently in use, as they are for the most part based on some sort of lexical knowledge of the words themselves. That is not to say that newspaper articles does not contain common sentiment bearing words, but the context is a lot less straightforward than in for example movie reviews. It has however been shown that by extracting the target of the article, and focusing on the sentiment directly connected to the target, rather than the article as a whole can prove to improve accuracy of classification considerably (Balahur et.al., 2013). This requires a way to identify who or what is the target of a given article and then to just analyze the text directly connected to the target, and exclude the rest. A logical way to solve this problem would be to implement a part-of-speech tagger. This is a tool that is meant to identify which part of speech each individual word belongs to (e.g. Noun, Verb, Adjective). Using the POS-tagger it would then be possible to identify the main subject of a given newspaper article, by finding the main subject in for example the headline or beginning of the article. With the subject localized, sentiment-bearing words close to the subject in the text could then be given greater value.

2.4.2 Cross-domain sentiment analysis

As mentioned earlier, a lot of the work done so far in the field of sentiment analysis has been done on movie reviews, products reviews etc.. This is very handy to test the feasibility of different sentiment analysis methods, but a persistent problem in the field in general is that each project is directed at a single source of data, for example only a set of movie reviews from IMDB.com, or product reviews from Amazon. The projects usually train exclusively on one set, and show good results when tested on the corresponding test set in the same domain. However, a problem arises if one attempt to use the same trained rules on a test set from a different domain, as the accuracy of the system trained in one domain significantly decreases when applied to a different domain (Blitzer et.al., 2007). One of the biggest effects of this is that if one wished to analyze sentiment in a specific domain, one first needs to build a sentiment analysis program specifically for that domain. Blitzer et.al. (2007) however suggests the possibility of training on a certain domain, and testing within another (where the domains are not the same, but still share some similarity). So for testing cross-domain sentiment analysis one needs several different datasets, of different domains, but with similarities.

2.4.3 Sentiment summary

Connecting to the newspaper analysis is the idea of sentiment summary proposed by Beineke et. al. (2004), where the idea is to be able to extract a single sentence from a larger text (a summary sentence) which conveys the sentiment of the text in its entirety. The idea from the article is based upon the movie review quotations from the webpage Rotten Tomatoes, a movie review aggregation site. For each of the official reviews there is a short quote exemplifying what the review itself contains. Beineke et. al. (2004) proposes a way of automatically extracting this sentence. Chapter 3 of this thesis, the development chapter, will cover a method of sentiment analysis directly connected to analyzing sentences rather than the entirety of text. This method is a possible way to pick a sentiment summary sentence, simply picking the strongest[4] sentence in the text.

2.4.4 Building sentiment lexicon

A large amount of the work done in sentiment analysis uses a pre-constructed lexicon of sentiment-bearing words to analyze texts (Kaji and Kitsuregawa, 2007). These lexicons can be built using thesauri, taking a small set of sentiment-bearing words such as ‘excellent’ and ‘horrible’ and then expanding on each of the two polar sides of sentiment. First by adding the directly connected words in the thesaurus, followed by adding words based on how often they co-occur with words already in one of the two polar sides (Kaji and Kitsuregawa, 2007).

Rather than using a pre-built sentiment lexicon this project will employ a different method, using supervised learning. Instead of using thesauri, the project will use a training data set from a specific field to build the sentiment lexicon, with different weights given to each individual word, based on different weighting schemes explained in chapter 3.

2.4.5 Weighting schemes and normalization

According to a study done by Paltoglou and Thelwall in 2010, increases in sentiment analysis accuracy can be achieved by implementing weighting schemes that are not purely binary in nature. They show that by implementing a modification on weights, for example through the use of TF/IDF[5], overall accuracy can be achieved. It should however be noted that it has also been shown that an implementation of only TF does

[4] Strongest meaning the sentence with the highest sentiment weight corresponding to the overall sentiment of the text in question.

[5] TF stands for Term Frequency, meaning the total amount of times a Term occurs in a document (Baeza-Yates and Ribeiro-Neto, 2011, page 69). IDF stands for Inverted Document Frequency, with higher weights representing the rarity of a term across documents in a dataset (Baeza-Yates and Ribeiro-Neto, 2011, page 72).

not increase accuracy, and can in many cases be detrimental (Paltoglou and Thelwall, 2010). Thus, for the development of the different methods for this thesis, a variant of TF/IDF will be implemented.

3. Design and Development

This chapter will go into detail on the different algorithms implemented in the sentiment analysis program. There are however a few necessary processes to be handled before any of the algorithms can be run on the dataset.

3.1 Pre-processing

During the development of the LLA primarily one dataset was used, the Large Movie Review Dataset provided by Stanford for use in testing and developing in the field of sentiment analysis (Maas et.al, 2011). The LMRD consists of an impressive 50.000 classified movie reviews, 25.000 positive and 25.000 negative. In addition to the separation of the set into negative and positive, it is also split into two sets for the purposes of training and testing.

All the data in the LMRD is user generated, in form of movie reviews. The reviews are informal, and in a lot of cases grammatically questionable. This presents a few problems that needs to be handled before any analysis can be run on the data. Following are the main items of pre-processing used.

3.1.1 Stemming

Stemming, or suffix-stripping, is the process of cutting down words (to their stems), in an attempt to minimize the number of unique words available for analysis (Porter, 1980). During stemming one attempts to remove postfixes of words, such as removing “ing” or “ly” from the end of words, so that similar words can more effectively be grouped together. Stemming uses simple rules as to what to remove, looking at the end of a word you see if the last three letters of the word are “legal”, if not you remove the offending pattern and check again.

Implementation of stemming in the LLA was done through the use of Lucene's PorterStemmer (Porter, 1980).

3.1.2 Lower casing

This is a process which again tries to create uniformity between the different words. The same word can appear in several movie reviews, or in the same review multiple times, with different capitalization. This proves to be a problem if one attempts to match words exactly to each other, as a program who has not been told to ignore capitalization will, correctly, classify the words "Good" and "good" as different. For the purposes of sentiment analysis however, the value of capitalization is negligible and it would be beneficial that the words "Good" and "good" be regarded as the same word. With this in mind, I have implemented a process in my program which simply turns every piece of text into lower case, thus ensuring uniformity.

3.1.3 Stopwords

Stopwords can be defined as words which holds (little to) no value in terms of relevance (Dolimic and Savoy, 2010). This includes words such as "the", "and", "a" and so on. These words are so common and frequent, and occurring in pretty much every review that they hold no value in regards to sentiment. As the word "the" is not an inherently positive or negative word, it makes sense to exclude it completely from the list of words used to classify the data. Thus I have included a popular stopword list, used in the indexation process of MySQL[6] databases, implementing it in such a fashion that if any word checked by the weighting or the analyzing process matches one of the words on the stopword list, they are simply ignored and excluded from the process of classification entirely.

3.1.4 Tokenization and removal of punctuation

Tokenization is the process of separating a string of characters into several shorter strings or characters, or words (The Stanford NLP Group n.d.). Of course this process requires certain rules to be done properly, if the shorter strings are to retain any

[6] Use of the MySQL stopword list is approved by Oracle. Proof of approval in the appendix.

semblance of meaning. My implementation of tokenization includes two simple parts to cut down paragraphs into sentences, and sentences into separate words.

The first part is to separate a string of characters into two strings on each occurrence of punctuation. Thus the string “This is a sentence. This is an exclamatory sentence! Is this a sentence?” becomes three separate strings: “This is a sentence”, “This is an exclamatory sentence” and “Is this a sentence”. An occurrence of ellipsis (“...”) or other forms of repeated punctuation proves no problem, as each occurrence only splits into a separate string of zero characters, and is then discarded as the string is too short to contain valuable information.

The second part is to then separate the sentences on each occurrence of white space (i.e. each word are separated).

3.1.5 The non-existence of pre-processing

Even though the previous sections explains the parts of pre-processing used for the analysis, the LLA can function without some or all of these steps. The point is not to exclude useful tools when they are available, but instead to still be able to function when they are not.

3.2 Supervised Learning

For the implementation of sentiment analysis, supervised learning was used to ‘teach’ the program how to classify new data. Supervised learning is a subset of machine learning, an area of computer science where a program ‘learns’ a set of patterns to figure out how to solve a given problem. The supervision in supervised learning is represented by the program being given a baseline of rules already from the start to solve the problem. This is normally done by separating a dataset into two set, a training set and a test set. Below is an explanation of the significance of the two different sets.

3.2.1 Training

The training set is the set of data which teaches the program the rules of analysis. For this thesis the LMRD was used, which contains 25.000 pure text files for the sole purpose of training. These files have been classified as either positive or negative in equal amount, giving a plethora of examples for the program. The main idea behind using the supervised learning is that the program will pick up patterns, or rules, from the positive and negative files. To explain it shortly, the program is supposed to go through all the negative texts, pick up the frequency of words occurring in them, and the more frequent the words are, the more negative they are considered. Then the same is done for positive words. After this is done we are sitting on two sets of weights, negative and positive. These are then combined into one set of both positive and negative weights, where words occurring on both lists are adjusted to match their total negativity and positivity, ending on one side of the scale, but to a lesser degree. A completed list of weights might then look like this example from Table 3.1:

Word	Weight
Happy	112
Disaster	-15
Potato	1
Movie	5
Boring	-78

Table 3.1, non normalized weights

It is important to note that even though the word “disaster” might be considered far more negative than the word “happy” is considered positive in terms of sentiment strength, what is actually counted here is the number of occurrences. The word “disaster” is less likely to show up in standard text than “happy” and as a result is given a less significant weight. To balance the fact that strongly negative or positive words might be given less significant weights, we will normalize the scores. The justification for normalization and balancing is that a text containing one severely negative word and a hundred weakly positive words could still be classified as negative because the weight of the one negative word outweighs all the positive ones, even though the text to a normal reader would come off as mostly positive.

Word	Weight	Normalized weight
Happy	112	3.049
Disaster	-15	-2.176
Potato	1	1
Movie	5	1.698
Boring	-78	-2.892

Table 3.2, TF-normalized weights

After normalization, the new weights might look something like what is shown in Table 3.2. The normalized weights are considerably closer to each other numerically, but still distinct enough that the bigger weights will matter a lot more than the smaller. The normalization is done very simply, with a slight adjustment to term frequency normalization shown below.

$$NW = 1 + log(weight)$$

Formula 3.1: Term Frequency Normalization

There is still one problem remaining in the example above. Before the normalization the word “Movie” had a comparably low weight, reflected by the fact that it might have occurred in a lot of positive and negative texts. But after the normalization it’s score is more significantly positive. So to fix this, inverse document frequency was implemented to balance the normalized weight scores.

$$IDF = log(\frac{N}{DF})$$

Formula 3.2: IDF normalization. N is the total number of documents, DF is the total number of documents the term occurs in

Word	Weight	Normalized weight	Document frequency	W*IDF
Happy	112	3.049	150	1.594
Disaster	-15	-2.176	15	-3.313
Potato	1	1	1	2.698
Movie	5	1.698	315	0.340
Boring	-78	-2.892	123	-1.761

Table 3.3 TF/IDF normalized weights. Total number of documents for this example is 500

The weights, as shown in Table 3.3, now more accurately describe their sentiment, not necessarily based on the word itself, but by the rules discovered by training the program. So even though the word “Movie” occurs more often in positive documents, it is by a small fraction, and as such it’s weight is very low compared to a rare word like “Disaster” which in this example has exclusively occurred in negative documents.

3.2.2 Testing

After the process with training, we now have a set of rules for how the program should judge new data. And that’s where the testing comes in. The test set is a set of data, containing different files than the training set, and to be able to ascertain if the classification by the program is done correctly or not, this set also needs to be pre-classified as negative or positive. The process of testing consists of letting the program go through all the files of one type in the test set (i.e. positive or negative) and then let it use the rules learned from training to determine if the files it is looking at are positive or negative. The accuracy of the program is then easily seen, as it corresponds directly to the percentage of correctly classified files. For comparison, the subjective accuracy of human subjects agreeing on sentiment, as shown by Wilson, Wiebe and Hoffman (2009) is at 82%, showing that even though we as humans might implicitly understand the difference between positive and negative words, we do not always agree on context.

3.3 Naive Sentiment Analyzer

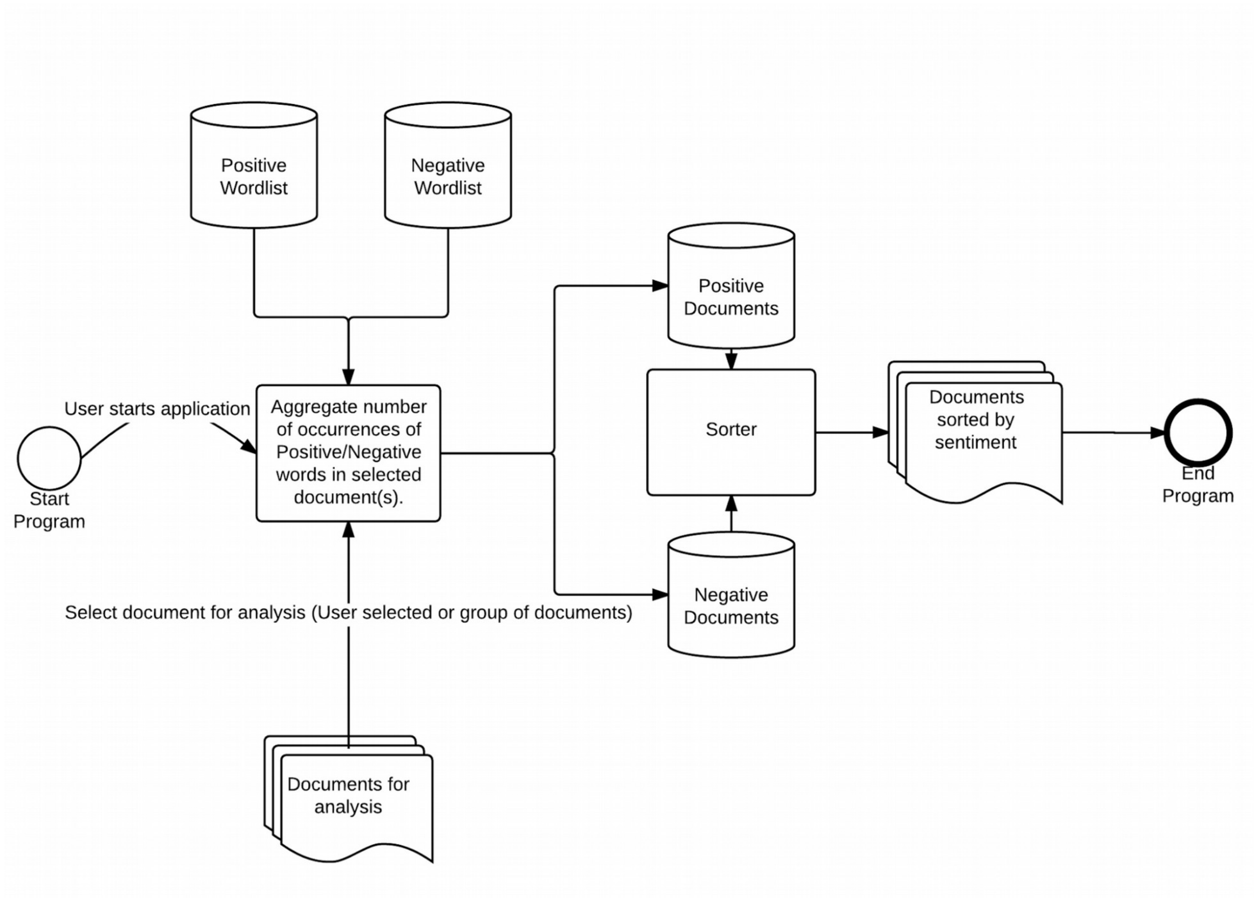

Positive Wordlist
Negative Wordlist
Positive Documents
User starts application
Aggregate number of occurrences of Positive/Negative words in selected document(s).
Sorter
Documents sorted by sentiment
End Program
Start Program
Select document for analysis (User selected or group of documents)
Negative Documents
Documents for analysis

Figure 3.3.1 - Naive analyzer

The naive sentiment analyzer is a simple sentiment analysis tool that looks at each document in a vacuum. This means that it takes no consideration as to what sort of document it is, nor does it use any sort of training-set to learn the rules of analysis. The naive analyzer simply looks at a predefined list of positive and negative words (eg. happy, wonderful, sad, terrible) and aggregates the number of occurrences of each word, as illustrated in Figure 3.3.1. Positive words are given a weight of 1 and negative words are given a weight of -1, no special consideration is taken in any circumstance.
As the analyzer simply compares the words to the predefined lists there is no need for pre-processing of the document(s) to be analyzed.

3.3.1 Development

The naive classification algorithm is rightfully not considered the best, it is however relatively easy to implement, and it provides a nice comparison to the other possible methods. For this method there were two parts of development needed.

Part one consisted of creating a function that allows the LLA to quickly compare parts of text to a predefined list of words. This is to see whether or not that part of the text contains sentiment-bearing words.
Part two, simply put, was to implement a way to loop through all documents in a set and move all documents to either a folder marked ‘positive’ or a folder marked ‘negative’.

3.3.2 Pseudocode

```
for each document{
        tokenize()
        for each word{
                if word is positive{
                        sentiment value ++
                }
                else if word is negative{
                        sentiment value --
                }
        }
}
```

3.3.3 Testing

250 positive documents, 250 negative documents (Large Movie Review Dataset)

86% accuracy on classification of positive documents.

34.4% accuracy on classification of negative documents.

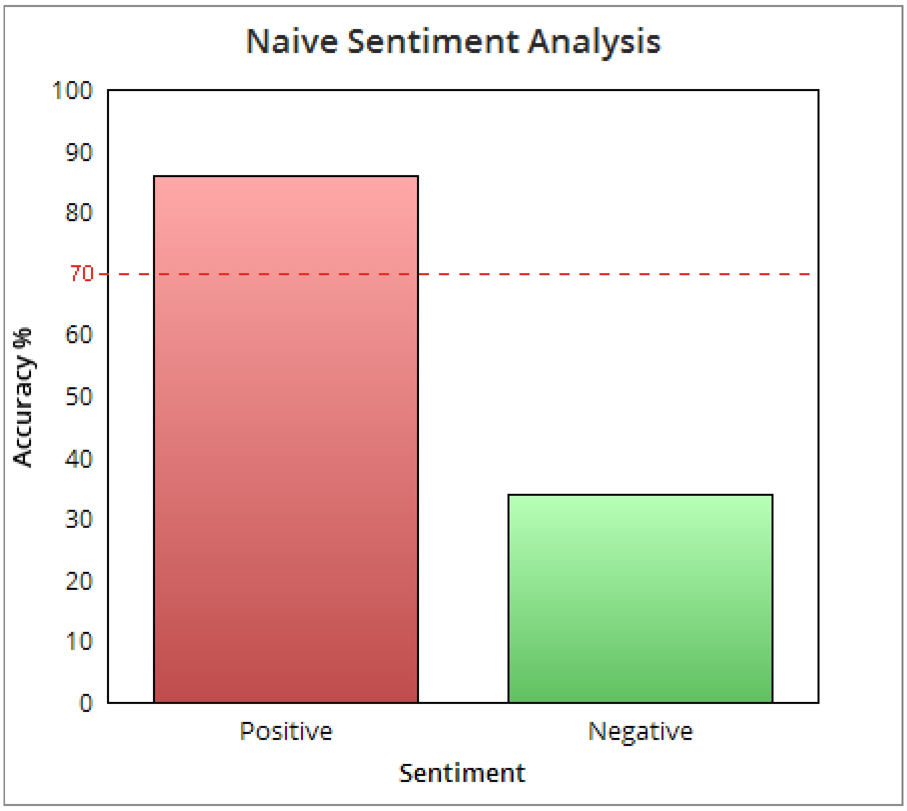


Figure 3.3.2 Naive Sentiment Analysis results

3.3.4 Results

This test was done mainly to verify that the analyzer worked, not as an attempt to achieve great results. Unsurprisingly, the results, as shown in Figure 3.3.2, are rather lacking and heavily slanted towards the documents being classified as positive. This can be symptomatic of the fact that people in general write more “flowery” when they are describing something positive (including more inherently positive words) and that a lot of negative sentences are heavily based on negating positive words with negating words (eg. not, don’t, can’t). This problem is likely to persist, as the naive analyzer never takes context into the equation, it purely measures against predefined positive and negative words.

3.4 Single Weight Weighter and Analyser

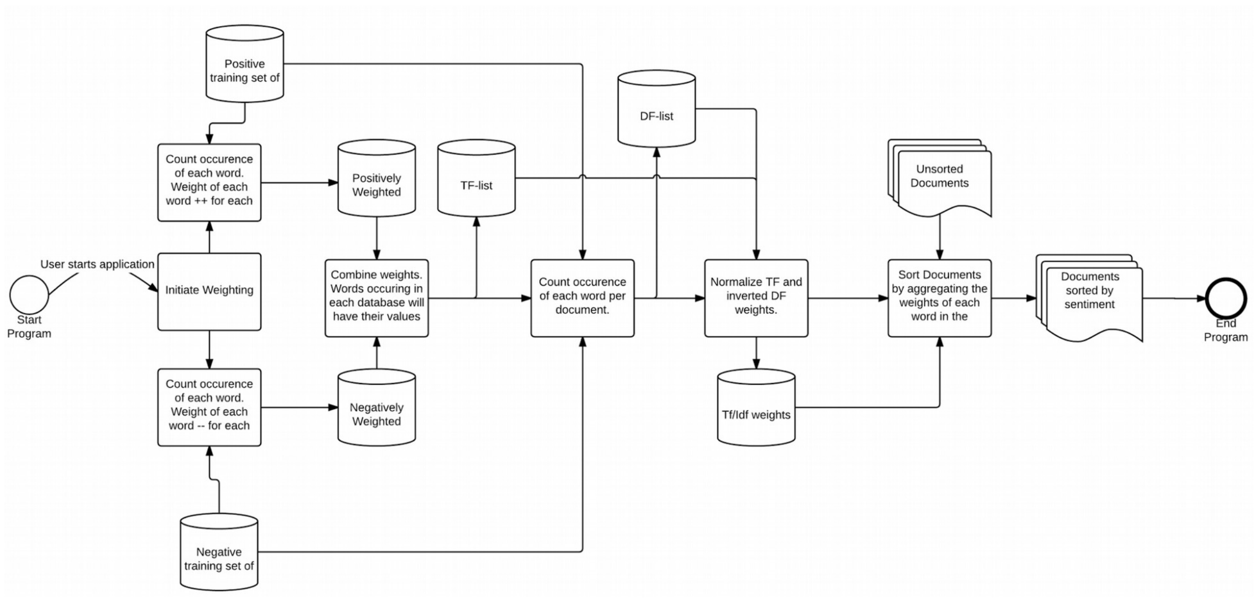

Positive training set of
Count occurence of each word. Weight of each word ++ for each
Positively Weighted
TF-list
DF-list
Unsorted Documents
User starts application
Start Program
Initiate Weighting
Combine weights. Words occuring in each database will have their values
Count occurence of each word per document.
Normalize TF and inverted DF weights.
Sort Documents by aggregating the weights of each word in the
Documents sorted by sentiment
End Program
Count occurence of each word. Weight of each word -- for each
Negatively Weighted
Tf/Idf weights
Negative training set of

Figure 3.4.1 - Combination diagram of the Single Weight Weighter and Analyzer

3.4.1 Single Weight Weighter

The single weight weighter is a system that takes a given training set, connected to a particular field, to create weights for each individual word. Each occurrence of the word in a positive document increases its weight, and reversely for negative documents. A wordlist is created by going through each document, as illustrated in Figure 3.4.1. If the word is new, it is given a weight of 1 if it is in a positive document, and a weight of -1 if it is in a negative document. If the word already exists in the wordlist its weight is adjusted by 1 in the relevant direction. Once all documents have been parsed in this way we have our weights.

3.4.2 Single Weight Analyzer

The single weight analyzer works in much the same way as the naive sentiment analyzer. The main difference being that instead of having a predefined list of positive and negative words, the list is created by looking at a training set of similar documents.

In addition, each word weighs differently, and as such some words are more positive/negative than others.

A document that is to be analyzed has all its words compared to the weighted list of words, and for each “hit” in the wordlist the corresponding weight is added to the result. If the final sum is greater than 0 it is classified as a positive document, and if it is less than 0 it is classified as a negative document.

3.4.3 Development

The LMRD features no special syntax, and as such it was no problem to base the program around analyzing pure text, without consideration for any special symbols or syntax. This also ensured that the program could easily use any other source of text, provided it was written with the UTF-8 typeset and that it featured no special syntax.

Development started with a focus on a way to create weights for individual words, preferably such that the weights could quickly be accessed upon analysis. As the LMRD contains 25.000 files for classification, it was vital that the runtime was kept to a minimum. To ensure that operational time of looking up the weights of each individual word was kept low, the weights were stored in a hashmap. This hashmap would use the word in question as a key, and the total sentiment value of the word as the value. Thus, during the weighting, all individual words found in the 25.000 files of the training set would be used as a key in the hashmap. If the word already existed, the value would be increased or decreased by 1. If the key had not yet been used the value would be set to either 1 or -1 depending on the what the file in the training set had been classified as.

Afterwards the TF/IDF balancing scheme (as explained earlier) was applied to the prior method. This was achieved by counting the total number of occurrences of each word in unique documents in a separate hashmap.

3.4.4 Pseudocode for Weighter

```
for each positive document in training set{
    tokenizer()
    for each word{
        if word not in weight hashmap{
            word as key in weight hashmap with value 1
        }else{
            increase hashmap value by 1
        }
        if first occurrence of word in document{
            if word not in document frequency hashmap{
                word as key in document frequency hashmap with value 1
            }else{
                increase hashmap value by 1
            }
        }
    }
}
```

The same is done for negative documents, decreasing the value of the weight, using the same hashmap.
After all weights have been aggregated is the normalization

```
for each entry in hashmap{
    weight = value of word in hashmap
    TF normalized weight = log10(weight)
    DF value = value of word in document frequency hashmap
    IDF value = log10(total number of documents/DF value)
}
```

3.4.5 Pseudocode for Analyzer

```
    for each document{
        tokenize()
        for each word{
            if word equals word in hashmap{
                total sentiment value += hashmap value
            }
        }
    }
```

3.4.6 Testing

First test:

No quantifiable results. When this test was attempted it was discovered that there was a serious runtime problem. As such the test was cancelled, and attempts at finding the bottlenecks in the program were underway. The problem lay in the process of gathering all the weights during the weighting, which were alleviated for the second test by using hash-mapping.

Second test (TF):

Test: 1000 positive and 1000 negative documents. (Large Movie Review Dataset)

positive accuracy: 62.9%

negative accuracy: 51.2%

Results

As this was tested with a straight tf-frequency it's not surprising that the results are less than favourable. Some words (stopwords in particular) are given incredibly high weights, resulting in the documents mainly being weighted by their amount of positive/negative stopwords and other words in general.

Third test (TF-IDF)

Test: 1000 positive and 1000 negative documents. (Large Movie Review Dataset)

positive accuracy: 71.4%

negative accuracy: 46.1%

Results

This test still does not take into account the problems with including stopwords. It does however resolve the issue of normalization. Each weight has been normalized in concordance with the TF-IDF model, and as such gives a more balanced look at the classification. Overall the test results are better than from the prior test, but it is clear that this way of analyzing is still subpar without further tweaking of the weights.

Fourth test (stemming and stopword removal)

Test: 12.500 positive and 12.500 negative documents. (Large Movie Review Dataset)

positive accuracy: 54.9%

negative accuracy: 68.7%

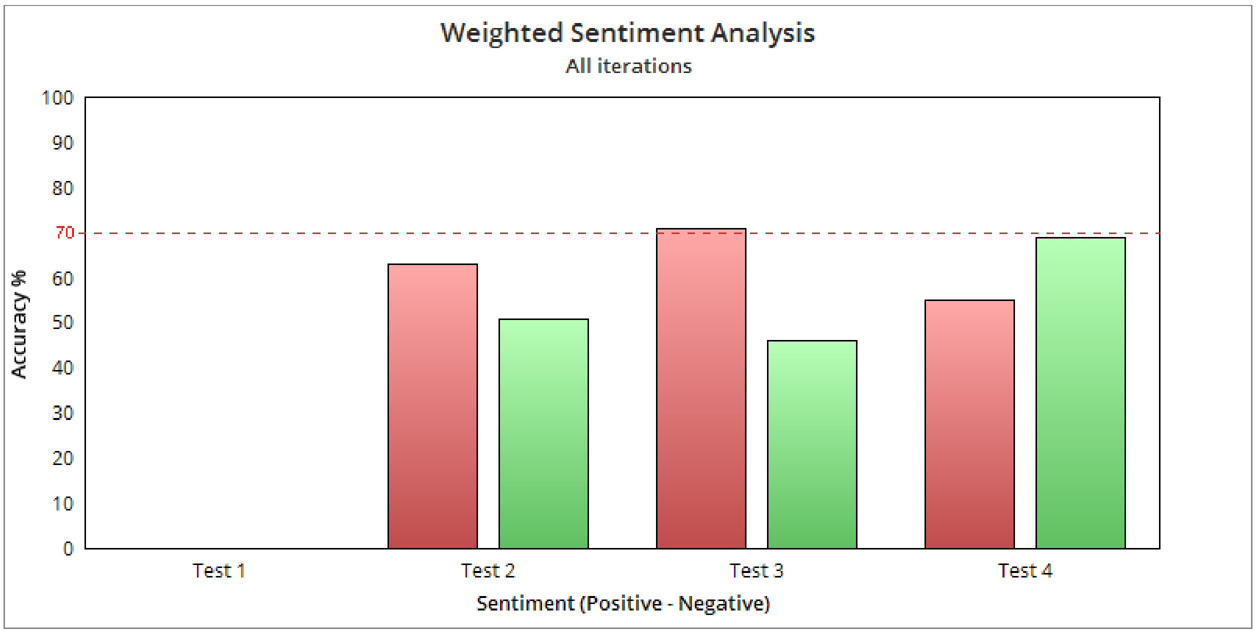


Figure 3.4.2, Single Weight Analyzer results

3.4.7 Final results

For this test both stemming and stopword removal were used. This led to a slight improvement in overall score, as shown in Figure 3.4.2, and a very interesting shift in correct classification. It turns out that stopwords accounted for a significant amount of the weights, as the negative classification went from nearly random (~50%) to acceptable levels, while the positive classifications turned the other way. The results as is, are not perfect, but they do provide a basis for comparison on future methods as well as testing on different languages.

3.5 Sentence Analyzer

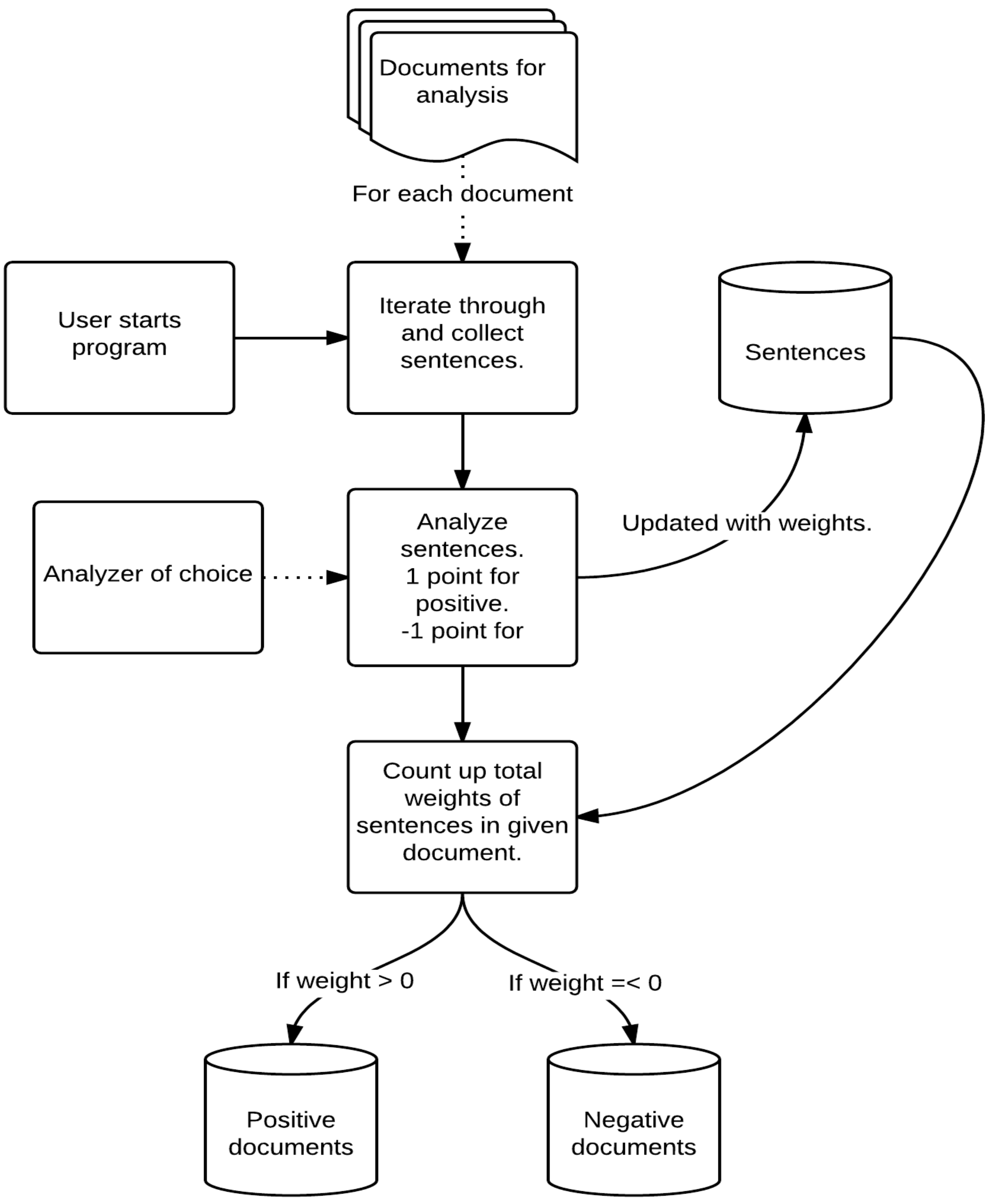


Figure 3.5.1, Sentence Analyzer

The sentence analyzer is in principle just an extension of the single word analyzer. Rather than aggregating the weights of each word in the document everything is done at the sentence level. This means that each sentence is given an individual aggregate weight, and the total weight of the document is the number of positive sentences minus the number of negative sentences, as illustrated in Figure 3.5.1.

The main positive side of the sentence analyzer is that it can normalize heavily weighted sentences. If the words of one sentence would have a positive weight of over 100 (quite high) it would be normalized by the sentence itself only having a weight of 1 in comparison to the other sentences. So a single sentence with a lot of heavily weighted words will not dominate the document as a whole.

3.5.1 Development

Using the same hash-mapping weighting scheme from the single weight weighter, development went smoothly. The main difference is that the weights are totalled on sentences rather than the entire document, thusly there was a need for a way to recognize sentences as their own entities. This was done by implementing a function using regular expressions looking for punctuation, line changes and so on. After the sentences are separated, they are each weighted using the single weight analyzer, and given a score of 1 or -1 depending on their overall sentiment. Then the total score of all sentences in a document is aggregated, leading to classification of documents as either negative or positive.

3.5.2 Pseudocode

```
for each document{
    split sentences()
    for each sentence{
        tokenize()
        for each word{
            if word equals word in hashmap{
                sentence sentiment value += hashmap value
            }
        }
        if sentence is positive{
```

```
                        total sentiment value ++
                }else{
                        total sentiment value --
                }
        }
}
```

3.5.3 Testing

First test: 12.500 positive documents and 12.500 negative documents.

Runtime: ~5 seconds.

Results: 85.76% correct positive classification

        56.24% correct negative classification

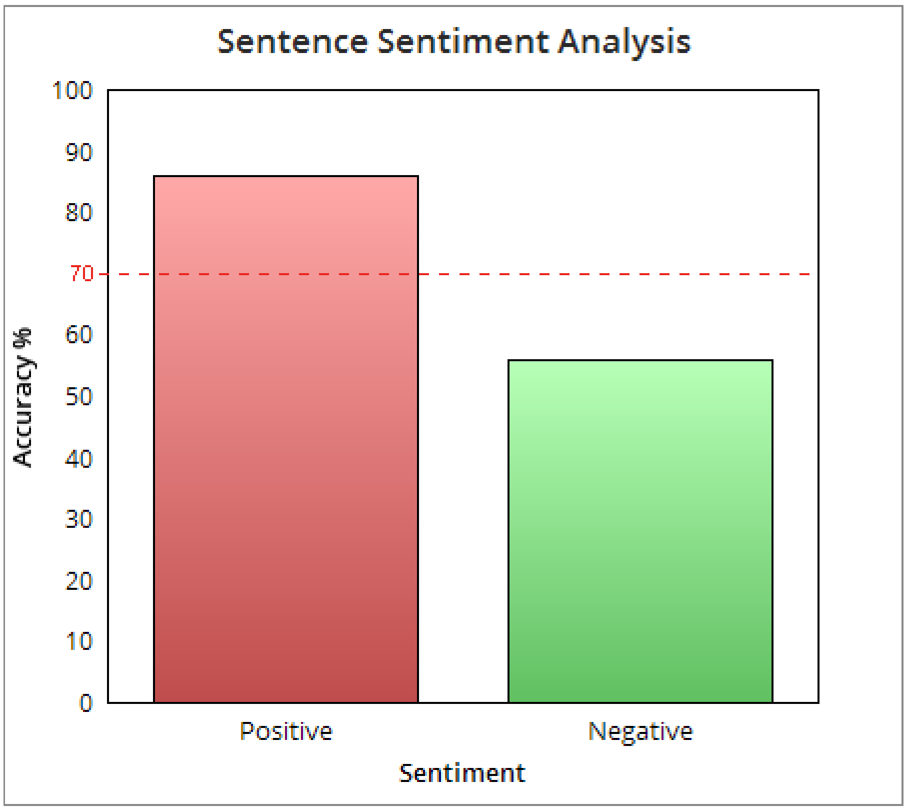
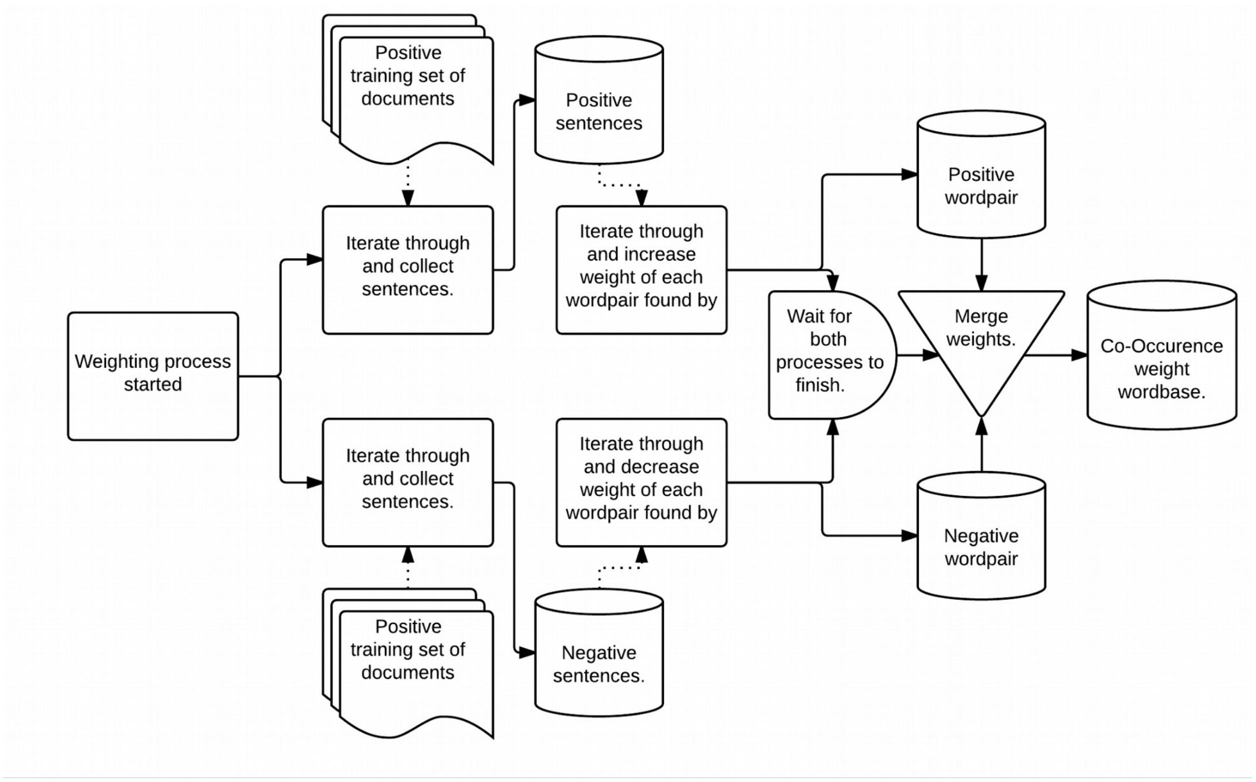


Figure 3.5.2, Sentence Analyzer results

3.5.4 Results

The sentence analyzer outshines the single weight analyzer in test. It is however still heavily lacking when it comes to negative classification. Though it is lacking, it gives some credence to the assumption that weighing parts of the texts as a unit, rather than each word separately can be effective. The results, as shown in Figure 3.5.2, produce a mean accuracy above 70%, significantly better than previous tests.

3.6 Co-Occurrence Weighter and Analyzer



Positive training set of documents
Positive sentences
Positive wordpair
Iterate through and collect sentences.
Iterate through and increase weight of each wordpair found by
Weighting process started
Wait for both processes to finish.
Merge weights.
Co-Occurence weight wordbase.
Iterate through and collect sentences.
Iterate through and decrease weight of each wordpair found by
Negative wordpair
Positive training set of documents
Negative sentences.

Figure 3.6.1, Co-Occurrence Weighter

The Co-Occurrence weighter works on pretty much the exact same basis as the single weight weighter, only multiplied. For the sake of easing on the complexity and computational requirements I have elected to do the weighting only on word-pairs, rather than the triples or larger combinations. Word-pairs are two words which occur next to each other in a text, i.e. co-occurrence. Though in theory you could include as many words as desired, when the combinations reach the same lengths as the average sentence it is advisable to just use the sentence analyser instead.

The weighter goes through each line of the training-sets and creates a list of wordpairs. Then these wordpairs are treated the same way as single words in the single weight weighter. For each positive occurrence their score is increased, and opposite for the negative occurrences, as illustrated in Figure 3.6.1.

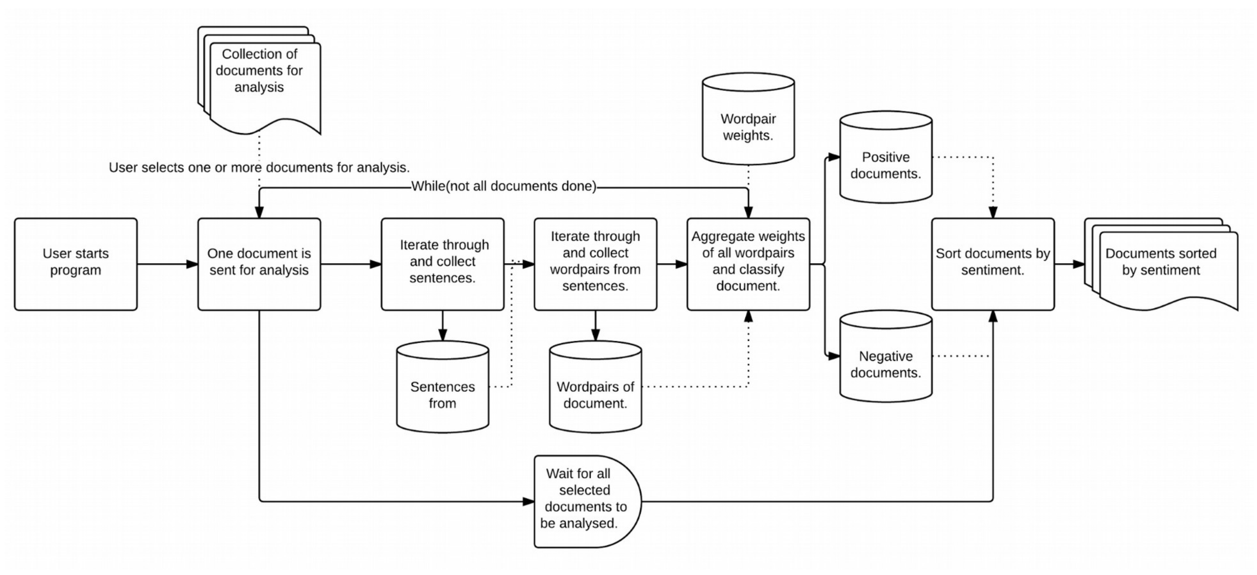

Collection of documents for analysis
User selects one or more documents for analysis.
While(not all documents done)
Wordpair weights.
Positive documents.
User starts program
One document is sent for analysis
Iterate through and collect sentences.
Iterate through and collect wordpairs from sentences.
Aggregate weights of all wordpairs and classify document.
Sort documents by sentiment.
Documents sorted by sentiment
Sentences from
Wordpairs of document.
Negative documents.
Wait for all selected documents to be analysed.

Figure 3.6.2 Co-occurrence Analyzer

3.6.1 Development of weighter and analyzer

The co-occurrence weighter is in reality a variance of the single weight weighter. Instead of using single words as keys for a hashmap, two words are used in conjunction. This provides a significantly larger hashmap, as the number of unique combination of words is always larger than the number of words, unless each word occurs only once in the entire dataset. The developmental process is for the most part based on the single weight weighter. One important addition is the function to pick out two words occurring next to each other in a sentence rather than single words.

3.6.2 Pseudocode for weighter

```
for each positive document in training set{
    split sentences()
    for each sentence{
        tokenize()
        for each word{
            if word is not last word in word[]{
                wordpair = word + next word
            }
            if wordpair is not in weight hashmap{
                use wordpair as key with value 1
            }else{
                weight hashmap value ++
            }
        }
    }
}
```

The same is done for negative documents, decreasing the value of the weight, using the same hashmap.

Normalization is not done on the Co-Occurrence Weighter. The number of unique weights created in the Co-Occurrence Weighter exceeds the number of weights in the Single Weight Weighter by a great amount. Each weight keeps its original weight, as weights with greater values are rare and should be considered significantly sentiment bearing.

3.6.3 Pseudocode for analyzer

```
for each document{
    split sentences()
    for each sentence{
        tokenize()
        for each word{
            if word is not last word in word[]{
                wordpair = word + next word
                if wordpair equals wordpair in hashmap{
                    total sentiment value += hashmap value
                }
            }
        }
    }
}
```

3.6.4 Testing

First test: 12.500 positive documents and 12.500 negative documents.

Runtime: ~6 seconds.

Results: 61.32% correct positive classification

60.54% correct negative classification

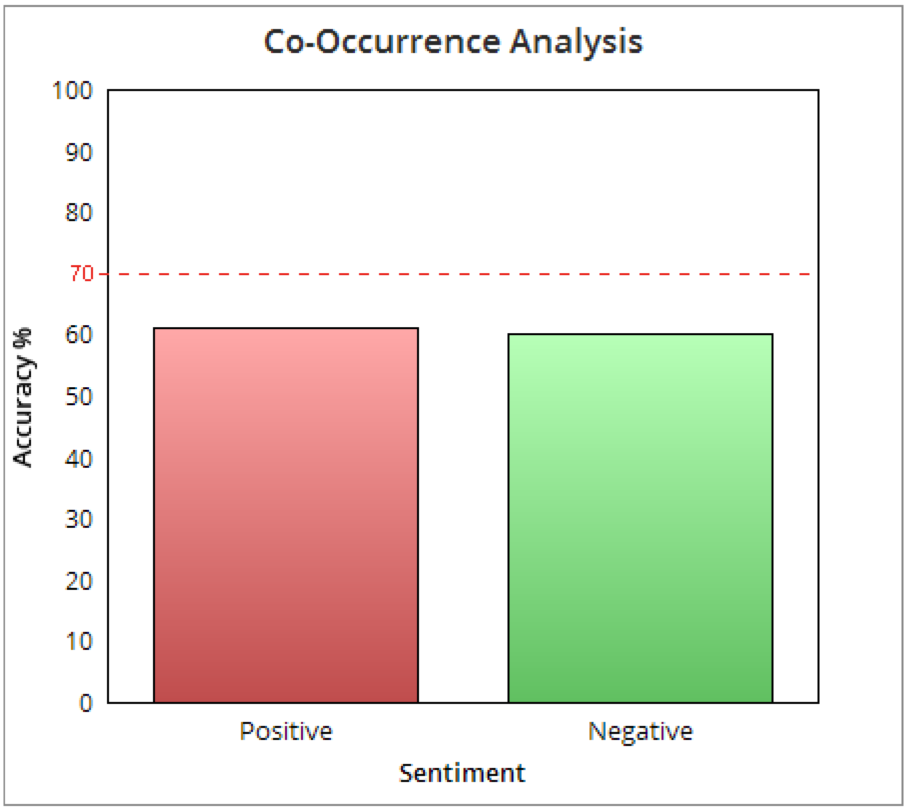


Figure 3.6.3, Co-occurrence results

3.6.5 Results

The Co-Occurrence analyzer proves to be the most stable of the different analysis schemes, as shown in Figure 3.6.3, it is however in both positive and negative classification more than 20% worse off than the classification agreement of humans (Wilson et.al, 2009).

3.7 Summary of algorithms

As shown previously, there are a lot of different algorithms to chose from when it comes to Sentiment Analysis. From the fairly weak, extremely easy, and quite frankly outmoded naive analysis to other more complex methods, like the sentence and co-occurrence. It is quite easy to show that some of the more complex methods bring better results, with the datasets used, but arguments can be made for the simpler methods. First of all, they are simple. That is to say that they are easily implemented and easily understood. They are also cost-efficient in terms of computing power and time. But considering the gap in accuracy and the relatively small trade-off in power and time it is for the most part worthwhile to "invest" in the more complex algorithms.

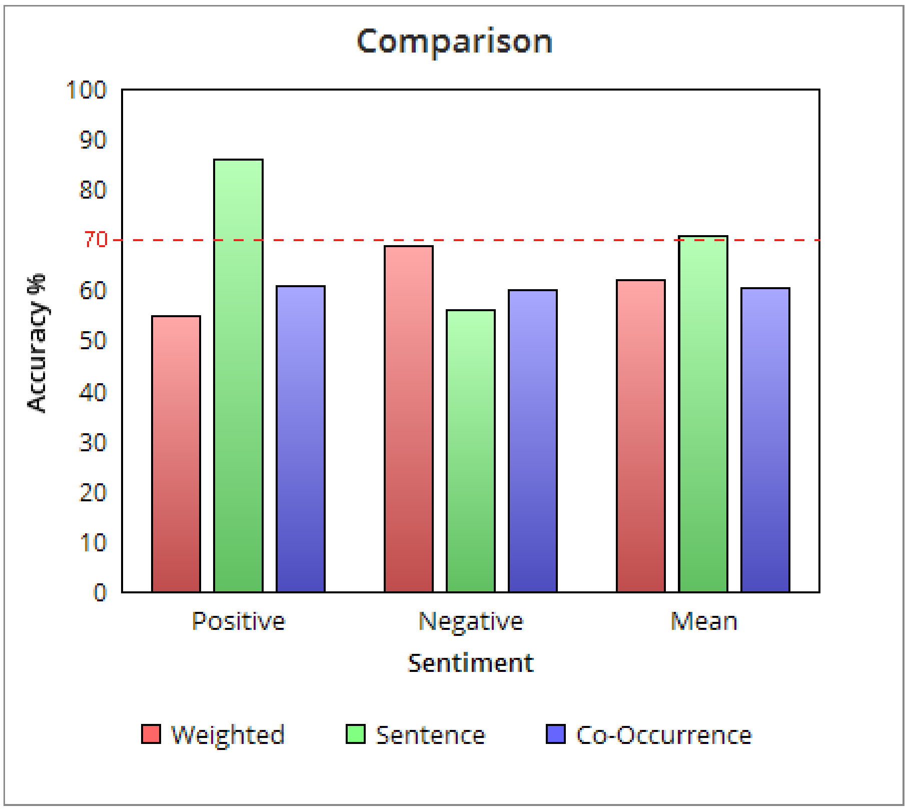


Figure 3.7.1 Comparative results

There exists arguments for two 'winners' amongst the algorithms. First and most obvious would be the sentence analyzer, achieving a mean accuracy above 70%, making it considerably closer to human levels of agreement (Wilson et.al.,2009). It is however a huge difference of its positive and negative accuracy, which the co-occurrence analyzer manages to alleviate. This analyzer receives sub-70% results, but is almost completely balanced in its likelihood to classify positive and negative texts correctly.

3.8 Development tools

This section will discuss the development tools that were used in the creation of the LLA.

3.8.1 Java

The choice to use Java for this project was not an arbitrary one. Java, while syntax-heavy and in some cases complex, is fully capable of representing every single programming need for this project. While it might not be the most efficient choice for all of the parts of the LLA, the value gained from not having to use different platforms, and potentially having to struggle to get these to co-operate, outweighs the potential negative sides. This coupled with the fact that this is the programming language that I am most capable and comfortable in working with made i the obvious choice for development.

3.8.2 IntelliJ

The use of Java as programming language more or less necessitates the use of and Integrated Development Environment (IDE) as writing Java code beyond a few lines can quickly become too much to handle by hand. Therefore I chose to write my code in the IDE IntelliJ, by JetBrains. IntelliJ offers a wide range of useful features for programming in Java, of which the most important, subjectively, are auto-completion and suggestion of code, easily structured packages, and integrated Github support.

3.8.3 Git and Github

Git is a useful tool for any development project. It is in essence a version control tool. Primarily this is used when a group of people are working on the same code, to easily verify that each of them is working on the latest code available, to share any individual updates, and to merge their work. It also makes sure that you do not overwrite somebodies code if a newer code has been submitted since you started changing it. In my case however, as I’ve been working on this project alone, it is simply used as a tool to store my code in an online repository, so that I can access it anywhere, regardless of which machine I am working from.

The repository is hosted by Github, which offers free repositories for use in projects like this. As mentioned earlier, Intellij has Github functionality included, so that I can automatically update my repository every time I change, remove or add to my code.

3.8.4 Lucidchart

As the amount of code, and the reach of the different programming ideas can become quite huge, it is important to have some way to keep track of your implementation plans. I chose to use Lucidchart[7], and online charting tool, to chart out the structure of my algorithms, going by this plan as I programmed. Each of these charts are included in the respective parts explaining each of the different algorithms.

[7] https://www.lucidchart.com/

4. Evaluation

This chapter will go through the several steps of evaluation. First is a short explanation of the two new datasets, followed by a short explanation of how the evaluation will be run. Finally is the test results, followed by a discussion of the result and development.

4.1 Data

Gathering the data from the tests was a fairly straightforward part of the process. The LLA was programmed in such a way as to return to the percentage of accurately classified files immediately after running the classification. For the development of the LLA the LMRD was used to both test and train. For evaluation of the possibility of translated sentiment analysis and cross-domain analysis using the LLA, some other datasets than the LMRD were necessary. The other datasets used were the Norwegian newspaper dataset and a sentence polarity dataset described below.

4.1.1 Norwegian dataset

The LMRD has been widely used in sentiment analysis projects and for the purposes of developing it works perfectly, but to achieve sentiment analysis in different languages than English we require datasets in different languages. As a proof of concept I decided to gather a Norwegian dataset and classify it. The process of generating a new dataset in a different language is, perhaps necessarily, a tedious process. The whole generation of the set, which will be used for part of the evaluation of the LLA, can be split into several parts:

- Locating the necessary articles
- Pruning
- Classifying
- Splitting

Locating the necessary articles

A part of the supervised learning for the program is that it should be given a number of text-files which are already classified by sentiment, and then let it create rules from it. The rules may be general, and applicable to any text of the same language, but there is no guarantee for this, as the context of a given text may vary the meaning of certain words considerably. In addition there might be differences in the vocational languages and nomenclature used in different fields. This necessitates the use of texts taken from the same field for rule-building and classification. With this in mind, a tool for locating texts in the same field, and preferably on the same subject, was needed. The tool used was Retriever, an online newspaper article archive with extensive search functions and the possibility to download a set of articles to the same file. With this tool I then downloaded every Norwegian newspaper article about “Stortingsvalget” from the year 2014 to ensure a common theme, and to make sure that the subject picked was contested, so as to avoid the completely neutral texts.

Pruning

An unfortunate side effect of downloading all the newspaper articles from Retriever was the format they were delivered. Once you have selected all the articles you wish to use they are downloaded as one very large text-file, rather than separate files. So the first part of the pruning was to separate the articles. This required me to locate the distinct start and end of each article to know where to cut off each file, and then code a little program to split the files. This was however not enough to use the files, as the files, even though they are pure text files, came with a typeset incompatible with my classification program. Thus I coded another short program to retype each of the files in the UTF-8 typeset.

Classifying the aforementioned articles as positive or negative

There are several possible ways of classifying the datasets, amongst them automatic and manual. The manual classification however needlessly uses an enormous amount of time, depending on the size of the dataset. Considering that the Norwegian newspaper dataset consists of several thousand full size newspaper articles, classifying these by hand would take an inordinate amount of time, hence my decision to do this process automatically. This was done using the naive analyzer. I realize that the naive analyzer does not produce the best results, though it provides the most straight-forward way of generating a test and training set. Even though there will be some files that are classified incorrectly, there will be a pattern in the selection of the files.

Splitting the dataset into a training set and a test set

As the Norwegian newspaper dataset is considerably smaller than the LMRD in terms of files, K-fold cross validation was used, in a hope to avoid overfitting during training. Overfitting is a problem that can occur in the field of machine learning, especially when the training set used to teach a program is too small. What overfitting means in this context is that the ‘rules’ learned by the training process are too specific to match any future unknown test set. This happens when there is a small amount of training data, so that a program might end up learning ‘rules’ completely specific to the training data, or in extreme cases more or less ‘learn’ the entirety of the training data. In this case, using the training data as test data could in many cases result in close to 100% accuracy. The problem however occurs when new unknown data is introduced. The test data might not fit most of the created ‘rules’ and as such the accuracy percentage would plummet. K-fold cross validation is one way of avoiding overfitting, working around the limited size of the data in total. As an example, given that the complete amount of data available both for testing and training equals 1000 files, a rather small amount, K-fold cross validation could be used to good effect. In this case the data could be separated into K subsets of equal size. Training and testing are then done K times, each time with a different partition used as training set and test set. K-1 sets are used for training each time, and the subset left out of training is then used for testing. After all the K tests are performed the mean accuracy of the tests can then be collected. Thus the Norwegian newspaper dataset was split into 2*5 subsets, 5 for positive training and 5 for negative training.

4.1.2 Movie Review Polarity Dataset

In addition to attempting to look at the viability of multi-lingual sentiment analysis, there is also the task of performing cross-domain sentiment analysis. As explained by (Blitzer et.al., 2007) complete cross-domain is not advisable, as there needs to be some connection between the domains. Thus another pre-classified set of movie reviews was selected. A movie review polarity dataset (MRPD) first used by Pang and Lee (2004).

The difference between the MRPD and the LMRD is in the length and content of the movie reviews. Where the LMRD contains short snippets, more akin to comments than actual reviews, this dataset contains page-long reviews going more in depth on the subject matter. This means that the content of the two datasets pertain to the same field, movie reviews, but are not written in the same manner.

4.2 Procedure

For the preliminary results achieved by running the LLA during development, the procedure was straightforward. The LMRD is already separated into a dedicated test and training set, thus running classification on the test set provided no challenge. For each of the separate tests the relevant weighting scheme was run on the training set and then the classification was run on the test set.

4.2.1 Evaluation standard

As stated earlier, the standard level of human agreement is at 82% (Wilson et.al, 2009). However, a wide variety of accuracy levels have been posted in different articles, ranging from 60% + to nearly 100% accuracy. Different methods literally produces different results, and even though any accuracy percentage above 50% (completely random assignment of classification) could potentially be considered as a positive result, I aim to achieve a percentage of accuracy closer to 70%. Thus 70% accuracy will be used as the baseline when it comes to judging all the results from the different tests.

4.3 Results from Norwegian newspaper data

Three different tests were performed on the Norwegian newspaper dataset, to test the viability of translated sentiment analysis using the LLA.

Single Weight Analyzer

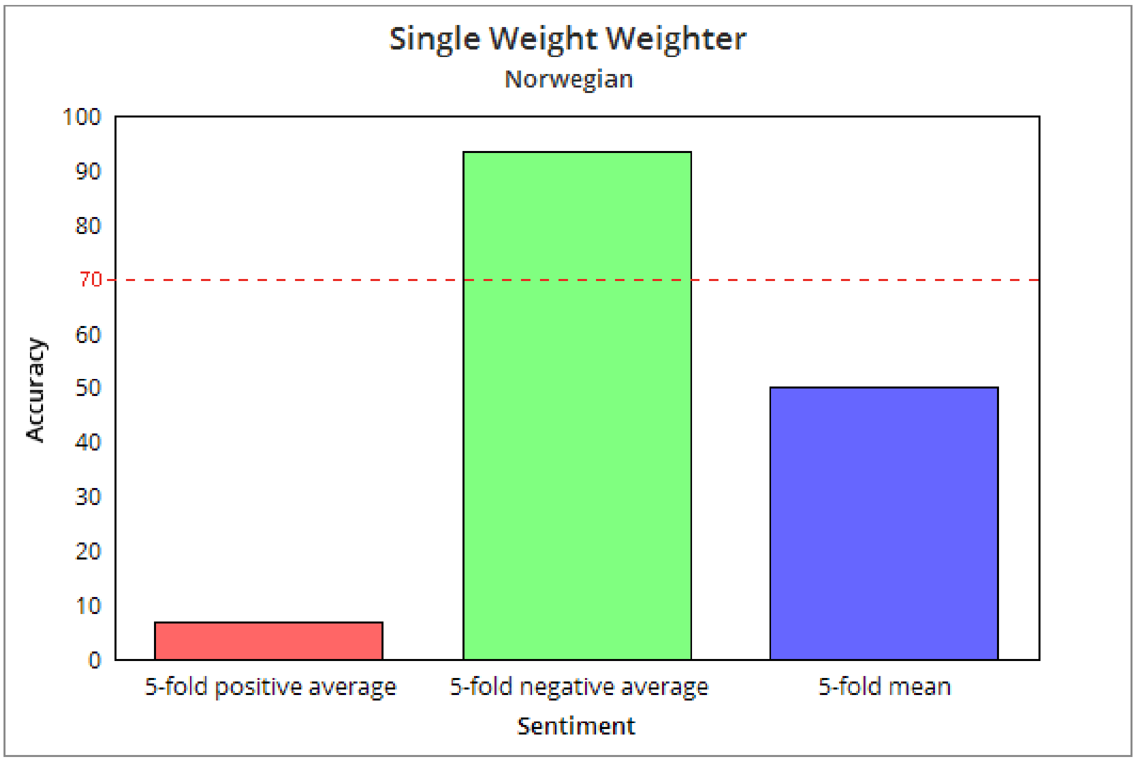


Figure 4.3.1 Norwegian Single Weight results

First off is the Single Weight weighting scheme. And immediately it is apparent that something is amiss. The results, as can be seen in Figure 4.3.1, are significantly skewed in favour of negative classification. This comes as a result of nearly all files being classified by the program as negative. There are a few reasons for why this might have happened.

1. The dataset was too small, such that even K-fold cross validation did not help enough to offset the overfitting problem
2. The newspaper articles are all of varying degrees of length. If the pre-classified negative newspaper articles were on average longer than the positive, they would contribute more to the weighting scheme, leading to higher negative, and lower positive weights.
3. The pre-classification was flawed. That is to say that the previous classification was wrong, and that no discernable patterns would emerge, as there was no actual data to base them on.
4. Neutrality of news. A final possibility is that since the analyzer works in a binary fashion (positive, negative) rather than a trinary (positive, neutral, negative) and news, even news covering politics are supposed to be presented in a neutral fashion, the pre-classification ended up more arbitrary than could be wished for. As shown in part 3.4, using data from news can be tricky unless a target can be specified in the text. And even in the case of a distinct target, there is a need for the sentiment to actually be present.

Sentence Analyzer

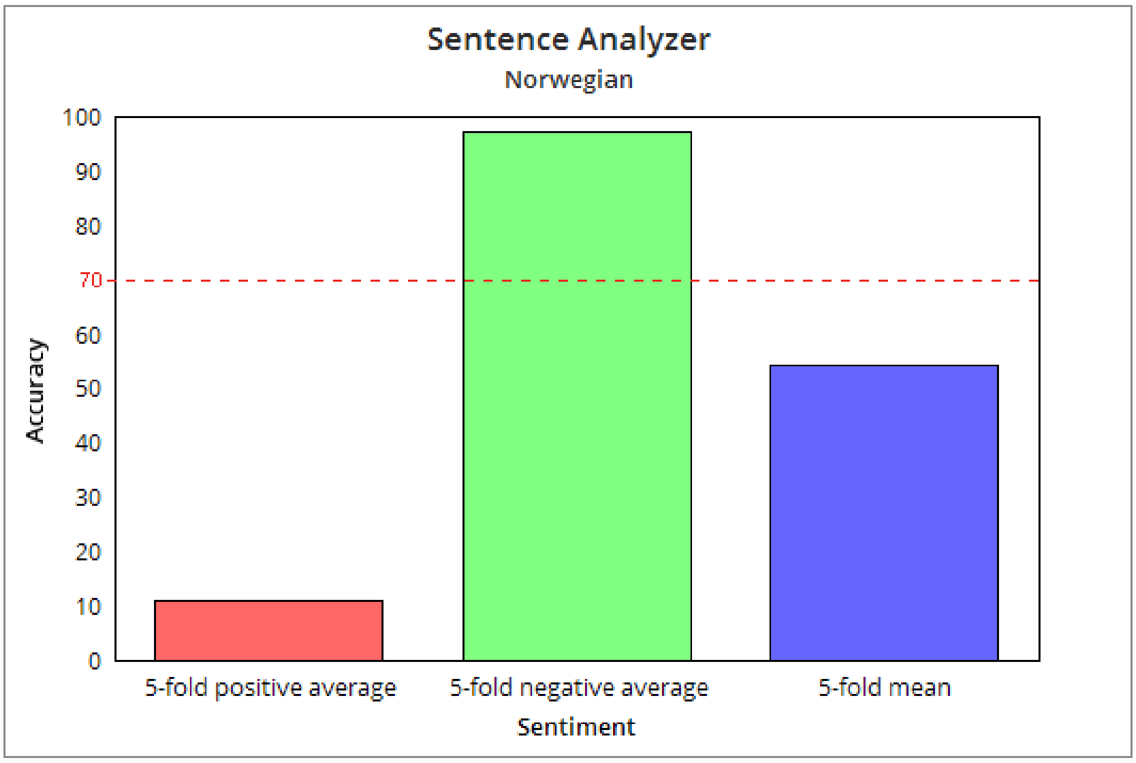


Figure 4.3.2 Norwegian Sentence results

The sentence analysis suffer from similar problems, with only slightly better results overall, as illustrated in Figure 4.3.2. To reiterate, the potential problems that were encountered here were either a too small dataset, varying lengths of the articles (lack of uniformity), flawed pre-classification, or the fact that the news are supposed to be inherently neutral.

Co-Occurrence analyzer

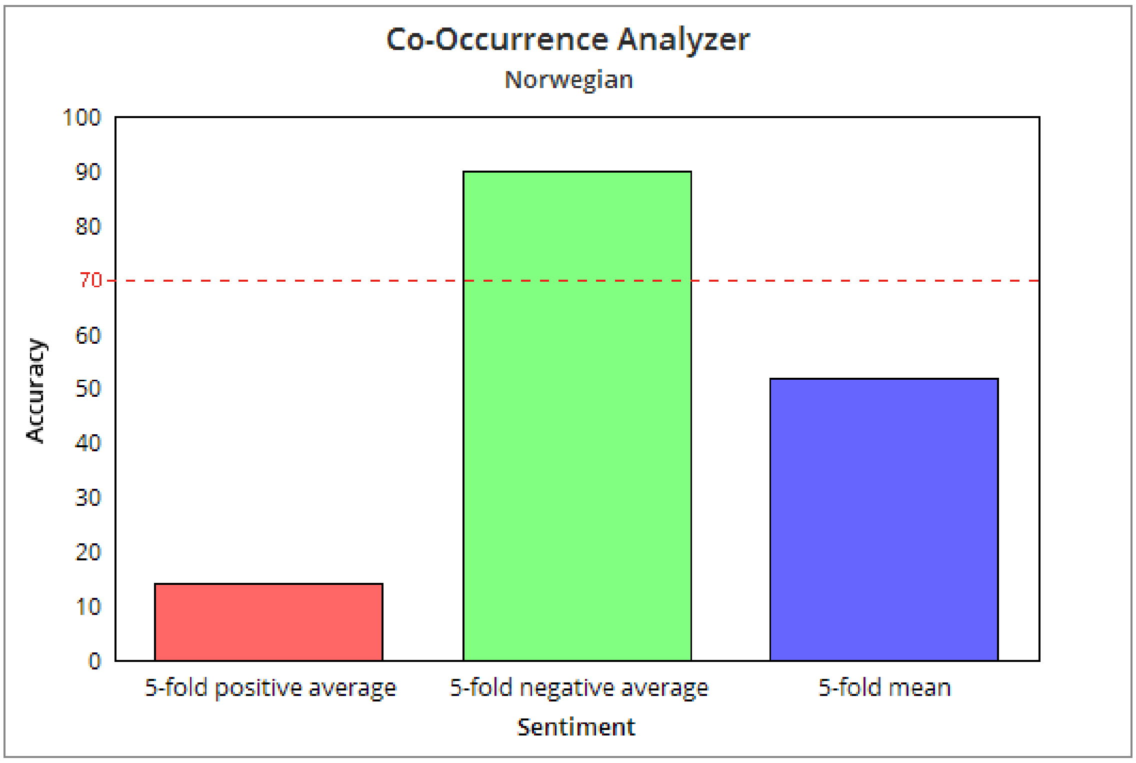


Figure 4.3.3 Norwegian Co-occurrence results

Finally there is the co-occurrence analyzer. The results, as made apparent by Figure 4.3.3, are still not good, but it backs up what was shown in chapter 3, namely that the co-occurrence method produces the most balanced results. These are significantly better than the other two, in that the results are more balanced and the positive score is slightly better, but the result is still less than encouraging. And the same overarching problems of the dataset and other problems are still existing. The mean score is still closer to 50% accuracy than 70%, but considering the hugely different results in negative and positive accuracy, the results are not of much value.

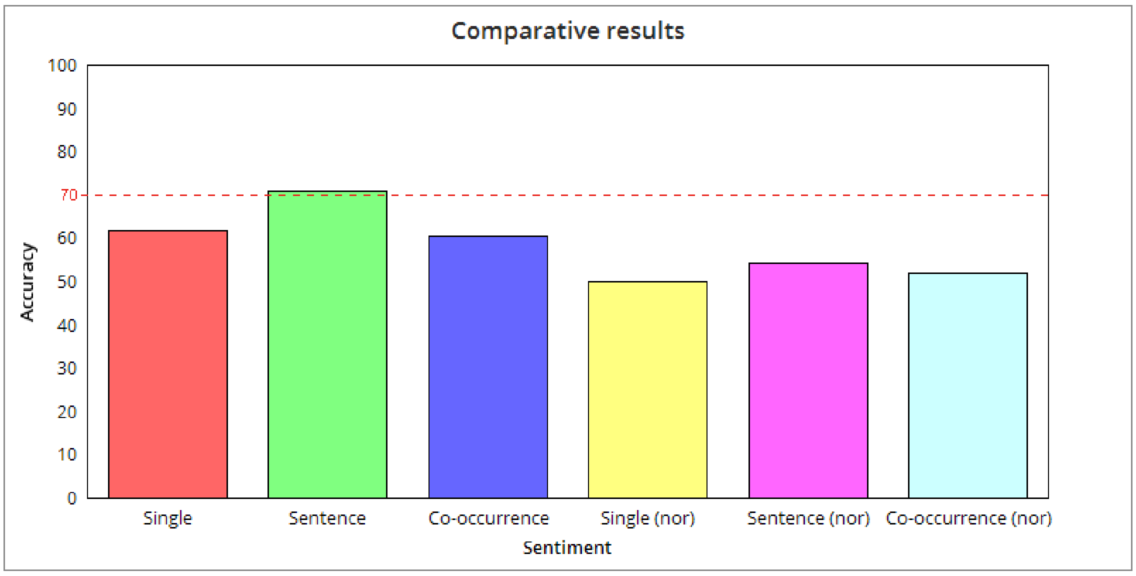


Figure 4.3.4 Norwegian and LMRD comparison

4.3.1 Summary

The attempt of LLA-created weights for analysis unfortunately created lackluster results, as can be seen in Figure 4.3.4, compared to the regular testing during development. As mentioned earlier, there can be several reasons for this, and it might be the case that it is a combination of more than one reason. For future attempts at testing this a dataset that has been professionally annotated by humans, and of big enough size to produce significant results is needed. This is unfortunately difficult to come across in languages other than English, as the interest for creating non-English annotated datasets is still low.

4.4 Results from Cross-domain analysis

To evaluate the results properly, three tests have been done on each of the different sentiment analysis methods. First is the baseline test, namely using the files from the MRPD dataset for both training and testing.

Second is using the LMRD for training and the MRPDt for testing.

Third and final is using the MRPD for training and the LRMD for testing.

4.4.1 Baseline test of the MRPD

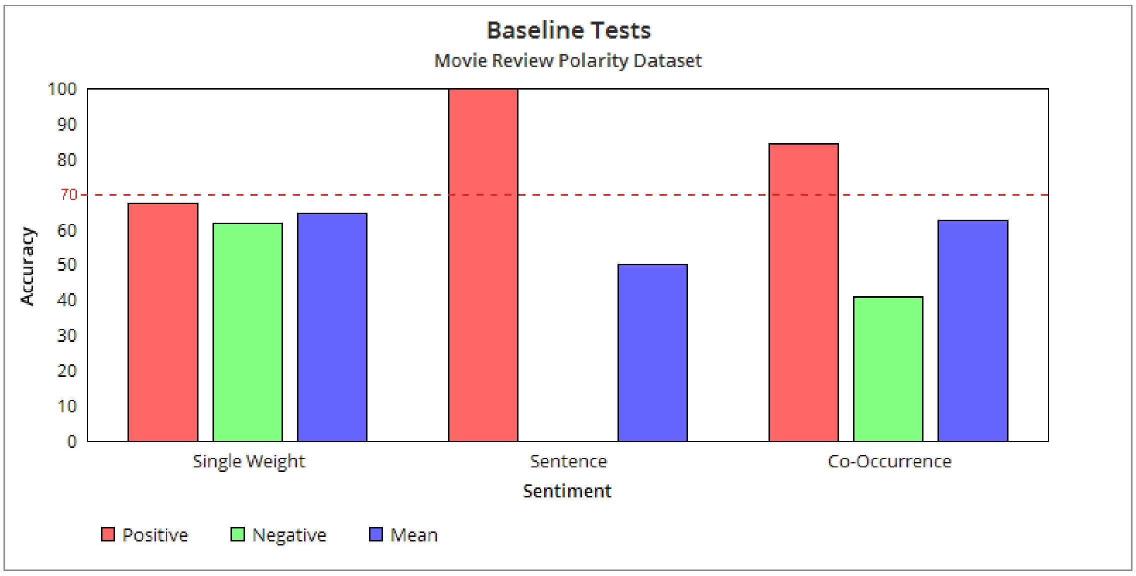


Figure 4.4.1 Baseline test of the MRPD

To provide a baseline for comparison, a test was done on the MRPD, using the same dataset for training. This ensures that the similarity between the content of the training set and test set is at an maximum, and should in theory provide the best results. The MRPD consistently achieves near 100% positive classification on all documents, a problem that will be looked at in the discussion. The mean results of the single weight and the co-occurrence analysis are slightly higher than the results achieved during development using the LMRD, as can be seen in Figure 4.4.1.

4.4.2 Cross-domain LMRD → MRPD

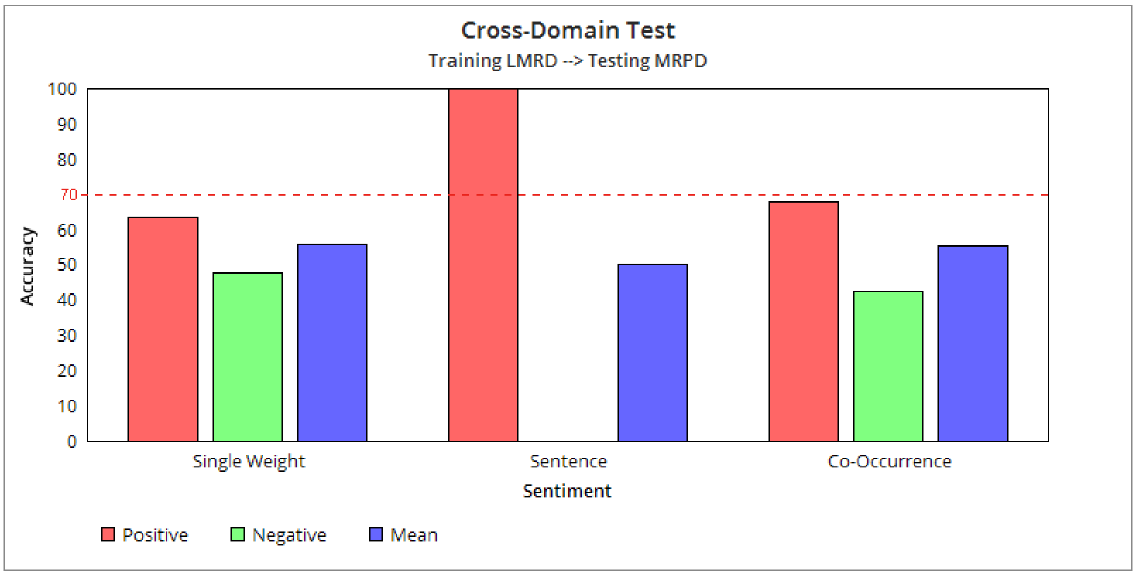


Figure 4.4.2 Cross-domain results LMRD → MRPD

For the first of the two cross-domain tests, the LMRD was used for training, just as it was during development. Immediately the differences in results are apparent, as shown in Figure 4.4.2. The mean accuracy is lowered for all tests (excepting the flawed sentence analysis), so where the baseline provided accuracy scores closer to 70% than 50%, this test does the opposite, with the mean score of both single weight and co-occurrence being approximately 55%. Both of the analysis methods ended up with a positive bias, also visible in the baseline test, which might suggest that the weights created from the LMRD in general leans more towards the positive side. The positive bias of the analysis will be discussed later in this chapter.

4.4.3 Cross-domain MRPD → LMRD

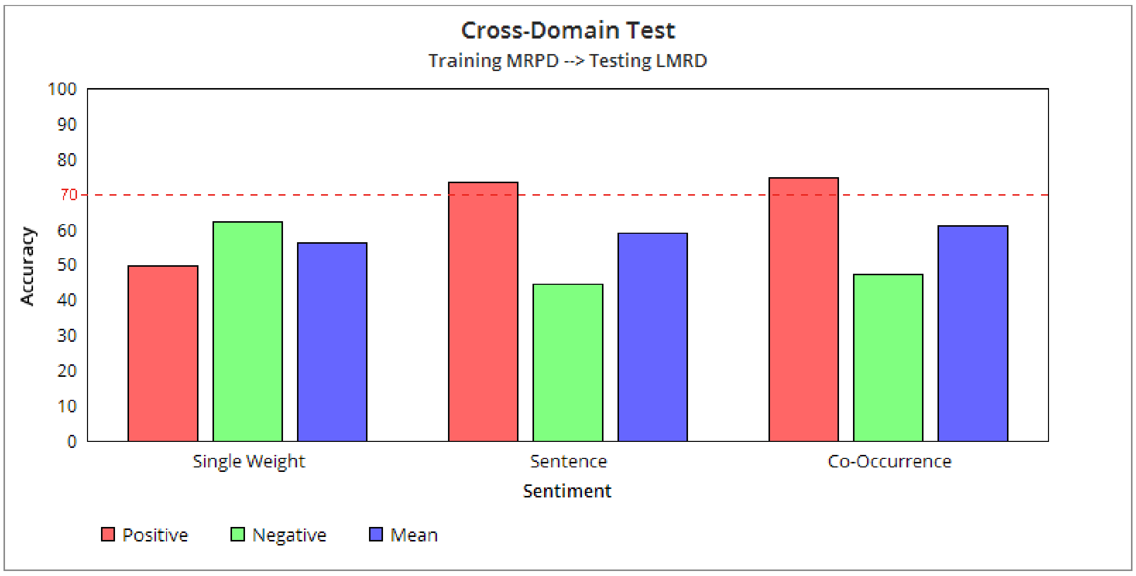


Figure 4.4.3 Cross-domain results MRPD → LMRD

For the second test, the MRPD was used for training and the LMRD for testing. The results are quite similar to the previous cross domain test, as seen in Figure 4.4.3, thus the mean results are worse than the baseline across the board. The following subsection will discuss the reason behind these poorer results, as well as addressing the positive bias present in all of the analysis.

4.5 Discussion

The introduction of this thesis contained three research questions. The development and results from the LLA will now be discussed in conjunction with the questions.

R1: "What method of sentiment analysis provides the highest percentage of accuracy?"

Rather than utilizing the lexicon based sentiment analysis mostly used in research today, an LLA-created lexicon was used. As shown in both chapter 4 and the previous section of this chapter, this did not necessarily produce the best results, although it does provide the possibility for comparison. And the answer as to which method produces the best results is, unsurprisingly, based on the training set and how the content of the test set is presented. Each of the three methods, single weight, sentence and co-occurrence came out on top in one test, each with a mean score of higher than 60% accuracy. Seeing as each of the three datasets has quite different composition, this may account for the difference in effectiveness of methods. Discounting the results from the testing of the Norwegian dataset for a little bit, as the results from it will be discussed later, considering that at best the tests achieved results only slightly better than random (50%), we can draw a few assumptions from the data of the tests on the two other datasets.

First, the sentence analyzer performs better on shorter texts, or in other words, on texts containing fewer sentences. It was shown from the tests during development (see Figure 3.5.2) that the sentence analyzer has a clear positive bias, whereupon it is more likely to classify negative texts as positive than it is to classify positive texts as negative. Thus it can be concluded that the analyzer in most cases will classify a sentence as positive, unless it is very strongly negative, and the more sentences there are in a text, the higher the likelihood of more positive sentences. The LMRD contains short, highly polar reviews, clearly stating the sentiment. The MRPD contains longer, more explanatory reviews. The longer the text is, there is a higher likelihood of it containing more neutral sentences, and given the positive bias, this will suffocate the negative results.

Second. The single weight analyzer is the simplest method (apart from the naive analyzer) and most akin to standard sentiment analysis algorithms, although it does not use a pre-built lexicon, as explained earlier. It consistently achieves less than 70% accuracy results, and in general worse results than the other methods, but seeing as the two other methods are offshoots and improvements upon the single weight analyzer, nothing achieved by those methods could be achieved without the single weight weighter. This does not mean that the two other methods are not more effective (because they are), just that they are all connected at a base level.

Lastly, with the exception some of the cross-domain testing, the co-occurrence analyzer continuously provides the most balanced results. While the mean accuracy is not necessarily the highest, the small spread between positive and negative accuracy is laudable. While it does not achieve as high a mean accuracy as the sentence analyzer, it does seemingly not suffer as much from sentiment bias in the text. This lends credence to the assumption behind the co-occurrence analysis itself, namely that words occurring together often, often share the same sentiment.

R2: "How feasible is sentiment analysis across specific domains?"

As explained in chapter 3, the reason for fashioning a different sentiment analysis system for different datasets is the inherent difference of different sets. This does not mean, however, that it is impossible to use the same system across different datasets, only that it won't necessarily produce the best results (Blitzer et.al., 2007). As shown by the test results, the in-domain tests produce results closer to 70% accuracy than 50%. The cross-domain tests posit exactly the opposite in results. From the existing literature on cross-domain sentiment analysis, this is definitely not unexpected, as further methods to connect the domains to each other is required to balance the weights more properly.

R3: "How viable are cheaper methods of translated sentiment analysis viable?"

This question, unfortunately, remains largely unanswered by this thesis. The main problem with trying to research this is not the methods implemented to analyze, but rather the dataset in question. The Norwegian newspaper dataset created for this thesis does not hold up to the same standard of the two other datasets, especially in regard to polarity of the articles. From the beginning there was no guarantee that the articles would be more than just slightly un-neutral, which was a problem pointed out in chapter 3. This is a problem in general when it comes to news articles, because if they are written correctly, they are supposed to have a neutral tone overall.

A different dataset would most definitely be able to produce different results, but a Norwegian dataset of the same quality of as the English ones is difficult to find and time-consuming to create. The addition of a Norwegian stopword list and algorithm for stemming could also significantly improve the results, as shown by the earlier tests run on the single weight analyzer.

5. Conclusion and future work

This chapter will look at the results gathered from development and evaluation. After that it will cover my reflection on the process, as well as suggesting further improvements and research for the LLA.

5.1 Conclusion

I have developed a sentiment analysis program, Local Lexical Analyzer, aiming to analyze documents in a dataset, completely independent of the language or domain of the dataset. The program includes three main methods of sentiment analysis, analyzing documents in three different ways. The goal was to test the feasibility of a tool capable of filling a void currently present in the field of sentiment analysis, namely a sentiment analysis tool made for a general purpose, rather than a specific one, in terms of domain and language. To that end, the following research question was put forward:

RQ: "How can a sentiment analysis tool that is independent of domain and language be developed?"

Each of the three methods presented, single weight, sentence and co-occurrence, presented results worse than those presented in other research papers on the subject of sentiment analysis. This was however expected, as a general tool in most cases will underperform compared to a specific one. When tested on a dataset often used when researching sentiment analysis, the LLA performed significantly better than when applied to a different domain and language.

Although the goal was to create a sentiment analysis tool capable of operating without outside resources such as specific sentiment lexicons, this does not mean that these resources should not be applied when they are available. The LLA will function without these resources, but implementing them to improve the accuracy results will always be possible in domains and languages wherein these resources exist.

With more time further improvements could have been implemented into the LLA, thus increasing the overall accuracy of classification. Due to time constraints several possible improvements to the LLA that could help analysis, especially in cross-domain analysis were not implemented.

5.2 Reflection on the process

It was apparent from the beginning of the thesis that the field of sentiment analysis is very open, which has its advantages and disadvantages. One advantage is that it puts very few constraints on what can be achieved and developed, and it is disadvantageous for exactly the same reason. The start of the development was one both empowered and held back by the extreme amount of possibilities. One of the goals that were fully achieved was to create a program, the Local Lexical Analyzer, which would function independently of other sources than the datasets of given domains. This can be highly useful, especially in specific domains where there exists no prior sentiment lexicons, or in languages in which there has been done little or no research on this field.

As shown by the results, the Local Lexical Analyzer does not quite achieve the goals set regarding accuracy percentage, and it is quite logically outshined by different projects using sentiment lexicons or other sources. This is however not a fault of the LLA, as these lexicons could easily be implemented as a part of the weighting schemes. What has been shown in the development of the LLA is that the LLA is not just a pipedream, but a possible way of using sentiment analysis in fields which previously has not been covered.

The Local Lexical Analyzer has several possible areas of improvement, as covered in chapter four, in addition to this there are several possible future applications which I wish to cover in the next subsection.

5.3 Suggestions for further improvements and research

From the results it is clear that there is room for improvement. Considering the fact that only one of the methods implemented achieves a mean score above 70% accuracy, it is clear that some of the implementation is less than ideal.

5.3.1 Accuracy improvement

To improve the overall accuracy of the LLA the obvious place to start is the single weight weighter. This weighter provides the basis for all of the analysis (with the co-occurrence weighter following the same method, but multiplied) and is the cornerstone for the entire project. The only straightforward way to improve the accuracy of the analysis would be to change the weights, or rather how the weights are created and changed. In the current iteration of the LLA the weights are created and then remain static, and this is a weakness. One possible improvement would be to run testing on the weights using the training set, further adjusting the positive and negative weights based on the results from this.

5.3.2 Cross-domain improvement

The main problem with cross-domain sentiment analysis lies in the name, namely that it crosses domains. As explained earlier it is inadvisable to use two datasets that have nothing in common, as there would be nothing combining the two, thus not creating a viable set of weights for analysis. So to handle this problem, even with more similar datasets, it is important to bridge the gap between them. There needs to be a way to discern where the similarities (and dissimilarities) lie, and enhance the weights that contribute to the most common ground between the datasets. This unfortunately makes it imperative that the content of both datasets are known in advance, and requires a way to discern which words, phrases or sentences are shared (the most) between the datasets.

5.3.3 Handling sentiment bias

After running some tests on the LLA-created weight-lists it could be seen that the total value of the positive weights are in all cases greater than the total value of the negative weights, and in some cases much greater. There can be plenty of speculation of why this is the case, it might for example be the case that in all of the datasets the positive texts in the training set are in general longer and/or contain more unique words than their negative counterparts. On a case by case basis it is possible to handle this by simply increasing the value of negative weights to reach a balance. There is however a few problems with this. The first one, as an imperative effect of increasing negative weights, is that positive classification will become even less accurate. The other problem with this is that the reason for the positive bias lies in the datasets themselves, rather than the LLA.

Assuming that increasing the value of the negative weights increases the mean accuracy percentage (that is to say that negative accuracy increases more than positive accuracy decreases), it would be possible to make the LLA adjust the weights itself. This would require the program to be run a number of times in succession. The first runthrough of the weights functions as it does at the moment, creating the weights from the training dataset. Then a test is run on a test set. If the positive accuracy far outshines the negative, or vice versa, a second training round is used to adjust the weights accordingly, giving a positive modifier (e.g. *1.1, *1.2) to polar side lacking behind. Then a test would be run again, comparing the results to the first run, and if the new results are within some pre-decided scope the weighting is finished, otherwise the process is repeated again.

5.4 Future development

One of the best parts of working with sentiment analysis is that this subsection of the thesis can figuratively go on forever. There is always room for improvement, and with the amount of work and research being done on sentiment analysis and natural language processing in general there will always be more ideas to test and implement. More specifically for this thesis however, there are quite a few possible routes to go.

5.4.1 Evidence analysis

One of the visions from early development of the LLA was to achieve a program that could detect whether or not hypotheses held true. This was a very ambitious goal, and even though this was not achieved, it was always kept in mind as part of the development process. The most fundamental parts are already covered, though still in need of improvement, but the ability to interpret what the different hypotheses are, and finally to interpret the conclusion is still in need of development. There are two main components needed for this program to come to fruition. Firstly, a way to discern the subjects and goals of the hypotheses.

H1 : *If* **I put a potato in boiling water**, *then* **the potato will be boiled**.

In the example above, **the potato** is clearly the subject and proposed goal of the hypothesis is to **boil the potato**. But methods for the program to actually separate the subject from the rest of the text would be required, so that sentiment analysis directly connected to the specified target can be done.

5.4.2 Translated sentiment analysis

Another goal of this thesis was to see whether translated sentiment analysis was viable. Translated might in this case be a bit of a misnomer. What is meant is a program capable of analyzing the sentiment of several different languages, not to actually translate the texts themselves, as machine translation is a different field, not one covered in this thesis. This was of course attempted in the thesis project, but because the test results ended up being inconclusive, it was difficult to draw any conclusion. A large fault in this would have to be the dataset itself, as it was created from newspaper data, that even though the topic covered was one that easily leads to polar views, it in the end suffered from neutrality. So for future testing on the capability of the LLA to analyze on different languages than English, a truly polar dataset would be required. In addition to this, to increase classification accuracy, as shown in chapter 3, a list of stopwords and a language specific stemming algorithm would be useful.

5.4.3 Part-of-Speech sentiment analysis

This is an envisioned sentiment analysis method thought out for the development part of the thesis, but unfortunately did not make the cut due to the development of other methods required more time than envisioned. The use of Part-of-Speech tagging is not in any way a new idea for sentiment analysis, just a possible enhancement to the LLA. The main idea behind this algorithm is that different parts of speech are potentially differentially important in regards to sentiment. That is to say, verbs might be less sentiment carrying than adjectives, and so on.

There would be two necessary parts of development to make this method work with the current iteration of the LLA. Firstly, a Part-of-Speech (PoS) tagger would be needed. The PoS tagger is a tool that detects which part of speech each individual word belongs to. With this tool it is possible to create separate wordlists for each PoS. Thus there would be one wordlist for nouns, adjectives etc.

Secondly comes the training of the algorithm. Each individual PoS list would be given a value between 0 and 100 which would indicate that PoS' importance in the weighting. The training would consist of several rounds, to determine the correct value for each PoS. The first round, with all given a value of 50 would provide the baseline, and after that different permutations of lower and higher values could be run to determine the optimal value of each PoS. It is seemingly a very interesting task in the field of machine learning, and something I would very gladly pursue at a later time.

5.4.4 Hybrid model sentiment analysis

The projected hybrid model is partway realized. That is to say that to some extent the LLA works on the basis of Plug and Play. Upon running the LLA it is easily possible to choose which algorithm should be used for weighting. However, the full vision behind the hybrid model is more the idea of combining all the different methods covered in this thesis in some way. This would entail finding a way to combine the weights from the different weighting schemes. This could feasibly be done by continuously training, using the training set as the test set, adjusting according the reported accuracy.

5.4.5 Target-specific sentiment analysis

As shown by Balahur et. al. (2003), the accuracy of the analysis of newspaper data can be significantly improved by extracting and identifying the target of the news.

Due to time constraints connected to the development of the LLA, implementing a method to extract the target was not achieved. But as it has been shown to provide a significant increase in accuracy it is well worth considering.

References

Baeza-Yates, R., & Ribeiro-Neto, B. (2011). *Modern information retrieval: The concepts and technology behind search* (Second ed.). New York: Addison Wesley.

Balahur, A., Steinberger, R., Kabadjov, M., Zavarella, V., Van Der Goot, E., Halkia, M., Poliquen, B. & Belyaeva, J. (2013). Sentiment analysis in the news. *arXiv preprint arXiv:1309.6202.*

Balahur, A., Mihalcea, R., & Montoyo, A. (2014). Computational approaches to subjectivity and sentiment analysis: Present and envisaged methods and applications. C*omputer Speech & Language*, 28(1), 1-6.

Beineke, P., Hastie, T., Manning, C., & Vaithyanathan, S. (2004). Exploring sentiment summarization. In *Proceedings of the AAAI spring symposium on exploring attitude and affect in text: theories and applications* (Vol. 39).

Bermingham, A., & Smeaton, A. F. (2009, July). A study of inter-annotator agreement for opinion retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (pp. 784-785). ACM.

Blitzer, J., Dredze, M., & Pereira, F. (2007, June). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. InACL (Vol. 7, pp. 440-447).

Daintith, J., & Wright, E. (2008). A dictionary of computing. Oxford University Press, Inc..

Dolamic, L., & Savoy, J. (2010). When stopword lists make the difference. *Journal of the American Society for Information Science and Technology*,61(1), 200-203.

Feldman, R. (2013). Techniques and applications for sentiment analysis.*Communications of the ACM*, 56(4), 82-89.

Kaji, N., & Kitsuregawa, M. (2007, June). Building Lexicon for Sentiment Analysis from Massive Collection of HTML Documents. In *EMNLP-CoNLL* (pp. 1075-1083).

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1), 1-167.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1* (pp. 142-150). Association for Computational Linguistics.

Nasukawa, T., & Yi, J. (2003, October). Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture* (pp. 70-77). ACM.

Paltoglou, G., & Thelwall, M. (2010, July). A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 1386-1395). Association for Computational Linguistics.

Pang, B., & Lee, L. (2004, July). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics* (p. 271). Association for Computational Linguistics.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3), 130-137.

Prabowo, R., & Thelwall, M. (2009). Sentiment analysis: A combined approach. *Journal of Informetrics*, 3(2), 143-157.

Sentiment [Def. 1]. In Merriam-Webster.com. (n.d.). Retrieved May 24, 2015, from http://www.merriam-webster.com/dictionary/sentiment

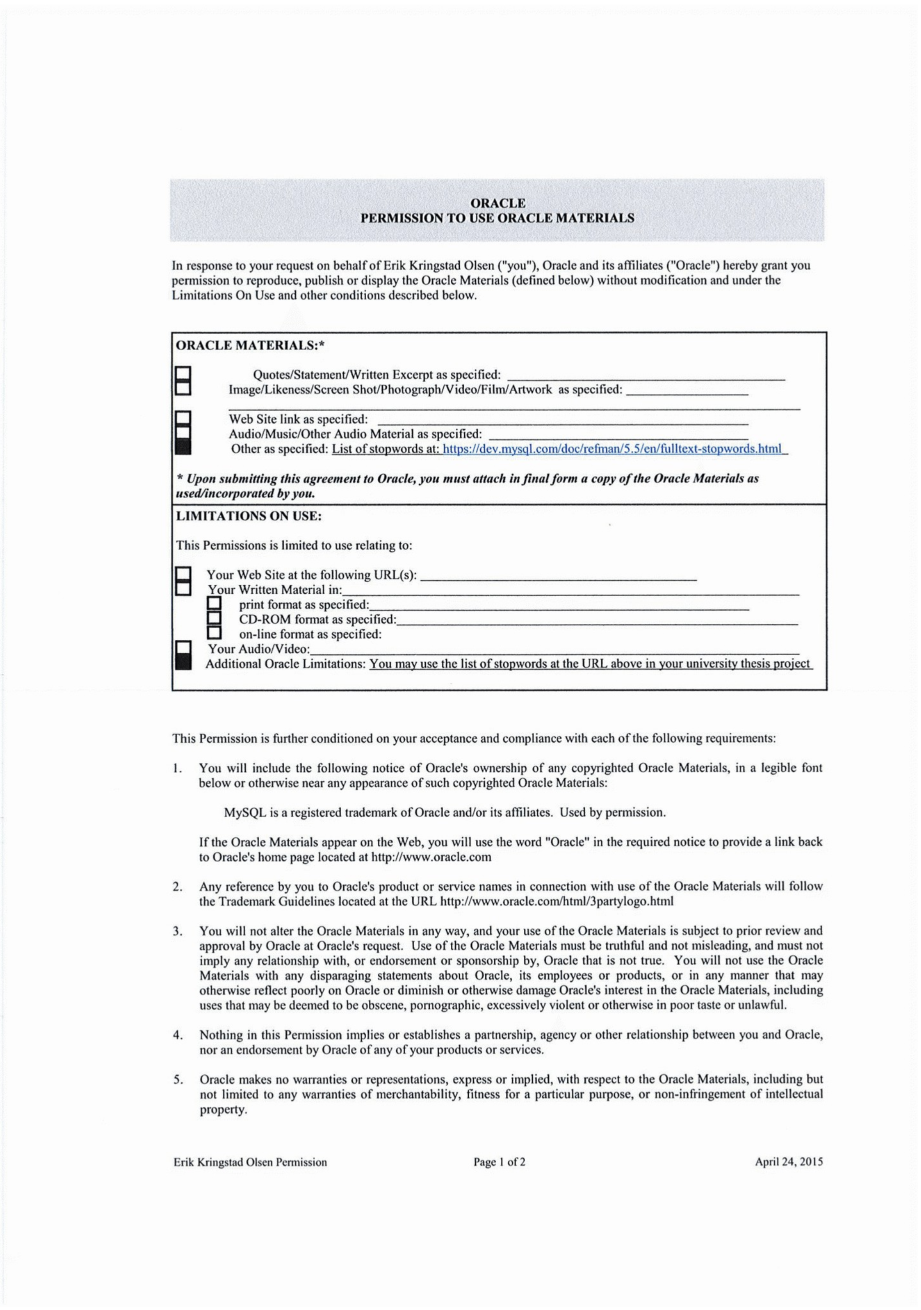
The Stanford NLP Group. (n.d.). Retrieved May 26, 2015, from http://nlp.stanford.edu/software/tokenizer.shtml

Tsur, O., Davidov, D., & Rappoport, A. (2010, May). ICWSM-A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Online Product Reviews. In *ICWSM*.

Wilson, T., Wiebe, J., & Hoffmann, P. (2005, October). Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing* (pp. 347-354). Association for Computational Linguistics.

Appendix

Permission of use, MySQL stopword list, Agreement form



ORACLE
PERMISSION TO USE ORACLE MATERIALS

In response to your request on behalf of Erik Kringstad Olsen ("you"), Oracle and its affiliates ("Oracle") hereby grant you permission to reproduce, publish or display the Oracle Materials (defined below) without modification and under the Limitations On Use and other conditions described below.

ORACLE MATERIALS:*

☐ Quotes/Statement/Written Excerpt as specified: ____________________

☐ Image/Likeness/Screen Shot/Photograph/Video/Film/Artwork as specified: ____________________

☐ Web Site link as specified: ____________________

☐ Audio/Music/Other Audio Material as specified: ____________________

■ Other as specified: List of stopwords at: https://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html

**** Upon submitting this agreement to Oracle, you must attach in final form a copy of the Oracle Materials as used/incorporated by you.***

LIMITATIONS ON USE:

This Permissions is limited to use relating to:

☐ Your Web Site at the following URL(s): ____________________

☐ Your Written Material in: ____________________

- ☐ print format as specified: ____________________
- ☐ CD-ROM format as specified: ____________________
- ☐ on-line format as specified:

☐ Your Audio/Video: ____________________

■ Additional Oracle Limitations: You may use the list of stopwords at the URL above in your university thesis project

This Permission is further conditioned on your acceptance and compliance with each of the following requirements:

1. You will include the following notice of Oracle's ownership of any copyrighted Oracle Materials, in a legible font below or otherwise near any appearance of such copyrighted Oracle Materials:

   MySQL is a registered trademark of Oracle and/or its affiliates. Used by permission.

   If the Oracle Materials appear on the Web, you will use the word "Oracle" in the required notice to provide a link back to Oracle's home page located at http://www.oracle.com

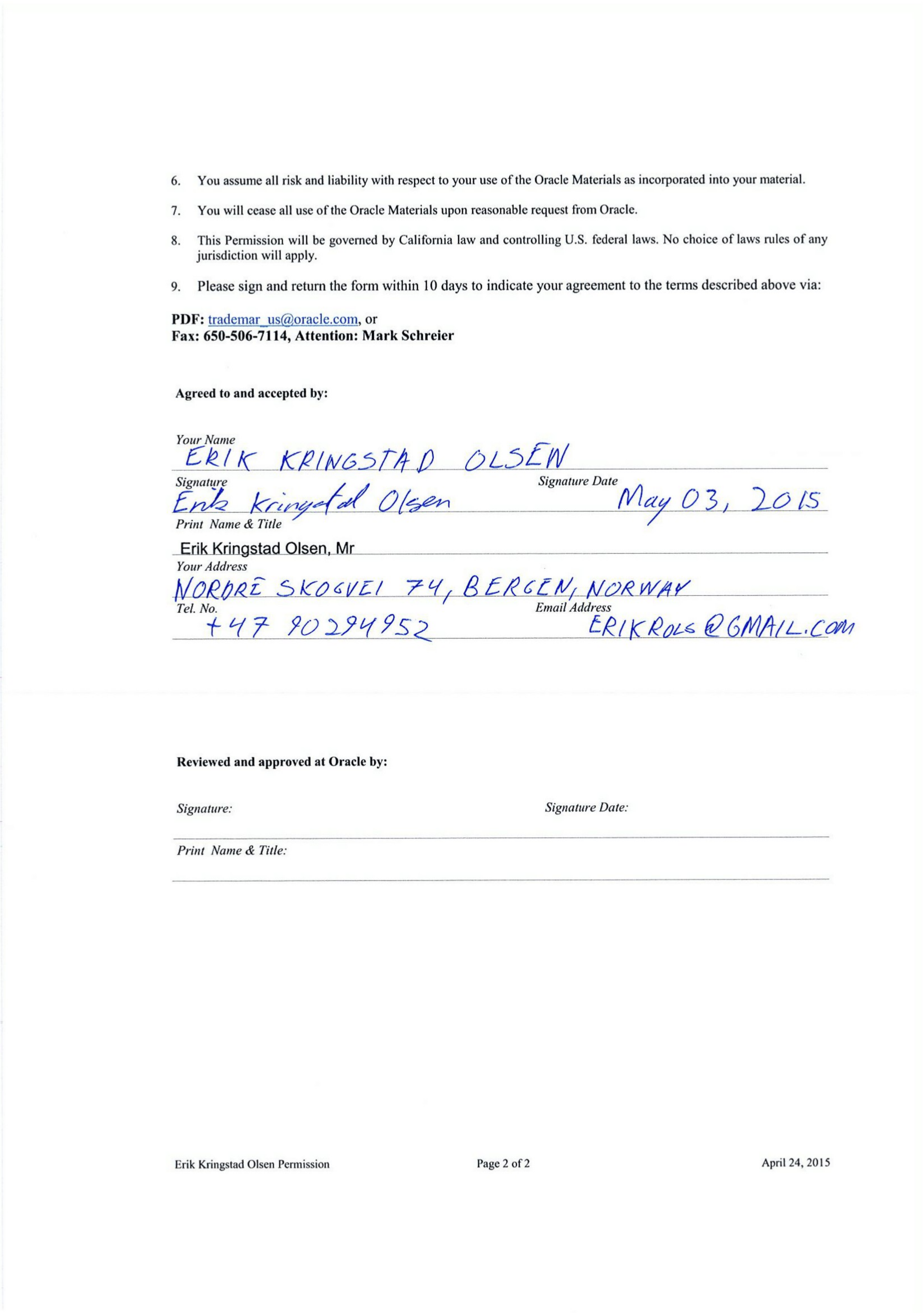
2. Any reference by you to Oracle's product or service names in connection with use of the Oracle Materials will follow the Trademark Guidelines located at the URL http://www.oracle.com/html/3partylogo.html

3. You will not alter the Oracle Materials in any way, and your use of the Oracle Materials is subject to prior review and approval by Oracle at Oracle's request. Use of the Oracle Materials must be truthful and not misleading, and must not imply any relationship with, or endorsement or sponsorship by, Oracle that is not true. You will not use the Oracle Materials with any disparaging statements about Oracle, its employees or products, or in any manner that may otherwise reflect poorly on Oracle or diminish or otherwise damage Oracle's interest in the Oracle Materials, including uses that may be deemed to be obscene, pornographic, excessively violent or otherwise in poor taste or unlawful.

4. Nothing in this Permission implies or establishes a partnership, agency or other relationship between you and Oracle, nor an endorsement by Oracle of any of your products or services.

5. Oracle makes no warranties or representations, express or implied, with respect to the Oracle Materials, including but not limited to any warranties of merchantability, fitness for a particular purpose, or non-infringement of intellectual property.

Erik Kringstad Olsen Permission Page 1 of 2 April 24, 2015



6. You assume all risk and liability with respect to your use of the Oracle Materials as incorporated into your material.

7. You will cease all use of the Oracle Materials upon reasonable request from Oracle.

8. This Permission will be governed by California law and controlling U.S. federal laws. No choice of laws rules of any jurisdiction will apply.

9. Please sign and return the form within 10 days to indicate your agreement to the terms described above via:

PDF: trademar_us@oracle.com, or
Fax: 650-506-7114, Attention: Mark Schreier

Agreed to and accepted by:

Your Name
ERIK KRINGSTAD OLSEN

Signature [illegible]

Signature Date May 03, 2015

Print Name & Title
Erik Kringstad Olsen, Mr

Your Address
NORDRE SKOGVEI 74, BERGEN, NORWAY

Tel. No. +47 90294952

Email Address ERIKROLS@GMAIL.COM

Reviewed and approved at Oracle by:

Signature:

Signature Date:

Print Name & Title:

Erik Kringstad Olsen Permission Page 2 of 2 April 24, 2015