Ville Jukarainen

FROM DATA MINING TO SENTIMENT ANALYSIS
Classifying documents through existing opinion mining methods

Bachelor's Thesis
Degree Programme in Business Information Technology

May 2012

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Opinnäytetyön päivämäärä

14.5.2012

Tekijä(t)

Ville Jukarainen

Koulutusohjelma ja suuntautuminen

Tietojenkäsittelyn koulutusohjelma

Nimeke

Tiedonlouhinnasta mielipiteen analysointiin - Tekstidokumenttien luokittelu olemassa olevien mielipiteiden louhintatekniikoiden avulla.

Tiivistelmä

Tässä opinnäytetyössä ehdotetaan ratkaisua dokumenttitason yleisen mielipiteidenlouhinnan ongelmaan, jolla etsitään mielipiteitä esimerkiksi tuotearvioista, uutisartikkeleista ja blogikirjoituksista. Tämä ratkaisu perustuu olemassa oleviin metodeihin sekä itseorganisoituvan kartan avulla tehtävään luokitteluun. Tarkoituksena on luoda järjestelmä, joka voi luokitella englanninkielisiä dokumentteja mielipideluokkiin, kuten positiivisiin, neutraaleihin ja negatiivisiin. Lisäksi ratkaisulle ehdotetaan toteutusta, jolla se voidaan sulauttaa Cluetail Oy:n olemassa oleviin järjestelmiinsä käyttäen Python-ohjelmointikieltä.

Työ alkoi tutustumalla koneoppimiseen, tiedonlouhintaan ja luonnollisenkielen prosessointiin, joihin mielipiteen analysointi pohjautuu. Tässä opinnäytetyössä etsitään mielipiteitä osittain ohjattujen sekä ohjaamattomien tekniikoita avulla. Sanastoihin perustuvaa positiivisten ja negatiivisten termien lukumäärää sekä lauseenjäsensääntöjen, ja ohjaamattoman statistisen mielipideorientaatiomenetelmän avulla poimittuja sanapareja käytetään luomaan mielipidevektori, joka kuvastaa annetun tekstidokumentin yleisesti vallitsevaa mielipidesuuntausta. Järjestelmä testattiin kahdella tuotearvosteluaineistolla. Mielipideorientaatiota (positiivinen - negatiivinen) sekä useampaa mielipideluokkaa (erittäin positiivinen, positiivinen, neutraali, negatiivinen, erittäin negatiivinen) etsittiin näistä aineistoista, ja suoritettujen testien tulokset esitellään. Lopuksi opinnäytetyössä käydään läpi opinnäytetyön aikana nousseita ongelmia, parannusehdotuksia sekä mahdollisuuksia laajentaa järjestelmä toisiin kieliin.

Kaiken kaikkiaan tämä opinnäytetyö on askel kohti toimivaa mielipiteiden louhintajärjestelmää sekä kapea johdanto mielipiteiden analysointiin aiheesta kiinnostuneille.

Asiasanat (avainsanat)

Itseorganisoituva kartta, Koneoppiminen, Luonnollisen kielen prosessointi, Mielipiteiden louhinta, Mielipiteen analysointi, Neuroverkot, Python, Tiedonlouhinta

Sivumäärä	Kieli	URN
50	Englanti	

Huomautus (huomautukset liitteistä)

Liite 1. Lähdekoodi Itseorganisoituva kartan implementaatio Python ohjelmointikielellä.

Ohjaavan opettajan nimi	Opinnäytetyön toimeksiantaja
Jari Kortelainen	Cluetail Oy

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 14.5.2012
Author(s) Ville Jukarainen	**Degree programme and option** Degree Programme in Business Information Technology

Name of the bachelor's thesis

From Data Mining to Sentiment Analysis – Classifying documents through existing opinion mining methods

Abstract

This thesis proposes a solution for *document-level opinion mining*, a method of finding overall opinion from given sources, for example, product reviews, news articles and blogs. This suggestion was done by using existing methods and an unsupervised *self-organizing map* for classification. The task is to create a system that can classify documents written in the English language, according to opinion categories, for example, positive, neutral and negative. Also, a design suggestion is made for how the presented solution could be implemented in Cluetail Ltd. systems, using Python programming-language.

Thesis process started by learning about the underlying techniques (Machine learning, data mining and natural language processing). These techniques create the foundation for learning sentiment analysis. Partially supervised and unsupervised learning methods were chosen for this approach. Lexicon based positive - negative term appearance features, and bigram features extracted according to part-of-speech tags and with calculated *opinion orientation* using an unsupervised statistical method. These features formed a feature vector for each document which describes the found overall opinion. Two review datasets with known opinion categories were used, and the capabilities were tested both with opinion polarities (positive - negative) and multiple opinion categories (very positive, positive, neutral, negative, very negative). The findings from these results, issues, improvements and the ways to extend this work in other languages are discussed on the last pages of this thesis.

Overall this work is the first step towards functional document-level opinion mining system, and a simple introduction to sentiment analysis meant for anyone who may have interest to learn about it.

Subject headings, (keywords)

Data mining, Natural language processing, Neural networks, Machine learning, Opinion mining, Python, Sentiment analysis, Self-organizing map

Pages	**Language**	**URN**
50	English	

Remarks, notes on appendices

Appendix 1. Source code for Self-organizing map implementation in Python.

Tutor	**Bachelor's thesis assigned by**
Jari Kortelainen	Cluetail Oy

CONTENTS

APPENDIX

1 Kohonen Self-Organizing Map Python implementation

1 INTRODUCTION

This thesis suggests a preliminary proposal for discovering the overall opinion within text documents from various publication sources. These sources can be, for example, news articles, blog posts, social media and product reviews. *Sentiment analysis*, also called *opinion mining*, is a relatively new field of study which has gained more and more interest during the recent decade. It is used, for example, to classify written documents in *semantic groups* like Positive and Negative. Neutral could also be mentioned but it can be seen as having no opinion or that there are the same amount of positive and negative opinions that cancel each other.

As the computers are getting faster, an increasing amount of data on the internet is becoming more available for everyone. A demand has risen for data mining that enables individuals and companies to extract and find useful information from the available data. This is also the case with sentiment analysis. There is a need for a way to identify the opinions of people regarding companies, products and people. Through this, for example, companies might gain valuable knowledge about their brand image, how their new product was received in media after launching, or how PR-campaign affected the publicity of a politician. By seeing the opinions, the information can be used to pinpoint possible problems that require more attention or to confirm successful moves. It could also be used helping people for searching relevant products, giving them an overall view of available options ranked by people's opinions. Sentiment analysis has many different uses and the task of finding opinion polarity within a document is just one of many. Other use cases are, for example, detecting subjectivity in documents, which means finding the subjective perspectives expressed by the writer, and it can also be applied for finding the different and possibly controversial perspectives that are expressed in a document.

What makes sentiment analysis difficult is that opinions are quite often subjective. A sentence like "God is great" can be positive for some people, but some would not agree. From an objective point of view, that sentence can be positive because there is a commonly known adjective "great" which may refer to something positive. On the other hand it could also be about one's size. Often opinions are domain dependent. This means a sentence like "The weather has been really hot and sunny." can have a positive orientation if it was about summer vacation. However if it was mentioned when talking about gardening, the sentence might have completely the opposite meaning because plants do not necessarily enjoy similar weather conditions as humans.

The goal of this thesis is to research and suggest a method for creating a sentiment analysis tool using Python programming language, unsupervised machine learning methods, and public data. This assignment was given by Cluetail Ltd., a Finnish company specializing in web and social media monitoring services. The focus is in identifying opinion alignment of documents written in the English language. The research problem is to study how unsupervised learning is performed in neural networks, what sentiment analysis is, and how to perform sentiment analysis by using unsupervised neural networks. Even though this thesis' techniques are focused on being used for documents written in the English language, it may also be possible to extend the same techniques for other languages. Suggestions for this will be discussed in further chapters.

As mentioned, there has been a lot of work in sentiment analysis during the past decades. All of the features used in this work to detect opinions can be found from a survey called "Opinion mining and sentiment analysis" by Pang & Lee (2008). This survey gathers existing techniques and approaches used in sentiment analysis. It was used as a pointer for finding more information to complete this thesis, also research made by Turney (2001, 2002) and Pang et al. (2002) contributed in looking for the solution. The books; *Web Data Mining* by Liu (2011) and *Artificial Intelligence: A Modern Approach* by Russell & Norvig (2011) helped in understanding the problem and looking for the solutions.

2 PYTHON PROGRAMMING LANGUAGE TOOLS

Python programming language was chosen because the existing systems are programmed in Python, therefore using the same language is natural. The availability of modules that helped in the development process contributed to choosing the language. Python offers good portability across different environments. The tools used include SQLite-database for data storage, NeuroLab library which provides pre-existing neural network algorithms implemented in Python, and Natural Language Toolkit that was used for Part-of-Speech tagging (Santorini 1990) and *n*-gram word extraction.

More details about these tools and how they were used are explained in further chapters.

2.1 Python programming language

The first version of Python programming language (http://www.python.org/) was released in the late 1980's by a Dutch developer Guido van Rossum. It is named after "Monty Pythons Flying Circus" TV-series. Python is claimed to be easy to learn and portable across multiple platforms. Python aims for usability and simplicity; this is achieved by providing easy syntax and advanced features like *garbage collection*. Garbage collection is an internal memory management system that automatically attempts to free memory reserved by application after data objects leave the scope of the program and are not needed anymore. Applications developed in Python are compiled to bytecode before they are executed in the Python virtual machine; however, it also has an interactive shell which allows users to type and execute code without the trouble of writing code into a file, and compiling it separately. Python also tosses out some of the common characteristics, like the curly brackets and semi-colons that are used in other languages. Instead of curly brackets, statements are organized in blocks by indentation. In the beginning, these small differences can make Python frustrating for developers who have used more "traditional" programming languages like C/C++.

Python can be extended with external modules and there are multiple ways to do it. One the most straightforward methods would be searching the module from Python Package Index (Pypi) or finding the project website and then downloading and installing using the provided instructions. There are also tools like "*easy_install*" that allow users to search and install packages with single command. Packaging system like "*apt*" on Debian-based systems, contain many common Python modules that can be installed.

The original version of Python is made in C-programming language, also called as cPython, but there exists implementations in Java (*Jython*) and .Net (*Iron Python*). Python can run on Windows, Linux/Unix and Mac OS platforms. Many of the Unix/Linux operating systems come with Python preinstalled. Besides these platforms, there are also implementations on Symbian, iOS and Android. Python can achieve equal and faster execution speed compared to programs developed in pure C.

This is partly because many of its built-in modules that are actually made in C-language, and it also allows developers to make custom extension modules that give a possibility of creating new object-types and to use C-libraries.

Python has gained popularity in scientific and mathematical computation and there are packages like Numpy and Scipy, which provide scientific computation tools similar to Matlab. Python is known to be used by organizations like NASA, Google, and European Space Agency. It is also used for scripting in applications like Blender and Autodesk Maya, and web application frameworks like Django which have been developed with Python.

Python has become popular in artificial intelligence and natural language processing. Pre-existing maintained packages and simplicity help in the development process and keep the work focused on the problem without having to worry about each tool and how long it will take to developed them.

2.2 NeuroLab library

NeuroLab library (http://packages.Python.org/neurolab/index.html) is a collection of neural network algorithms written in Python programming language. These algorithms can be easily accessed and implemented after installing the library. At this moment the used version 0.2.1 supports following the neural networks:

- Feed Forward Multilayer Perceptron
- Competing layer
- Single Layer Perceptron
- Learning Vector Quantization
- Elman Recurrent network
- Hopfield Recurrent network

Neural networks, especially the *competing layer* neural network known as *Kohonen Self-Organizing Map* is explained in more detail in chapter 3.

The library can be downloaded and installed from the above-mentioned web-site and it can be found from Python packaging index by its name. Using the library is relatively easy. An example code to create and test neural networks can be found in *NeuroLab Online Manual* (2012).

2.3 Natural Language Toolkit

Natural Language Toolkit (NTLK) (http://www.nltk.org) is an open source library, distributed under Apache 2.0 license, multiplatform collection of tools and modules for natural language processing and text analysis, available in Python programming language. It provides a large set of tools and algorithms for text tokenization, stemming, tagging, chunking, parsing, classification, clustering, measurements and semantic interpretation. It also provides an access to many text corpora and lexicon.

In this work, NLTK is used for *n*-gram extraction and part-of-speech tagging from documents. The use of these features is explained more in the chapter 2. More about using NLTK library can be found in the book *Natural Language Processing with Python* (Bird et al. 2009).

2.4 SQLite

SQLite (http://www.sqlite.org/) is a multiplatform library that is a self-contained SQL database engine. Its strength comes from the lack of need for additional configuration and special server software. SQLite is published in public domain which means it is available for everyone.

SQLite can be used in environments where traditional databases are not convenient or even impossible, for example, handheld devices and smart phones. Web-browsers like Firefox, Chrome and Opera utilize SQLite to store data easily and without extra burden for the user. Self-containment makes it also convenient for testing purposes where new databases need to be created and removed without going through the trouble of setting up dedicated server software. Databases can also be easily distributed and copied.

In Python SQLite is used via sqlite3 database-API that provides a simple access to the engine. In this work SQLite was used for storing words and rules that have been extracted and classified from data. SQLite was chosen because of the simplicity and portability across the environments. An example code can be found in Python 2.7.3 Documentation (2012) concerning SQLite3.

3 DATA MINING AND MACHINE LEARNING

This section introduces data mining and machine learning techniques. In the first chapter, data mining, web data mining and the concept involved in these fields are introduced. In the second chapter, neural networks are explained and unsupervised Kohonen self-organizing map is discussed more in-depth. In the last chapter of this section, other learning models are introduced and a model called support vector machines is presented.

3.1 Data Mining

As mentioned in chapter 1, computing power and the amount of public and private data is expanding. Modern computers, growing number of storage space, and the internet have made data collection an easy task. But just collecting data is not always enough. Individuals, businesses and organizations keep looking for ways to use data to learn and to gain valuable information. This is not anything new. It has been done since the beginning of time and has even kept humans alive. By understanding patterns in human and animal behavior, humans have avoided dangers and succeeded in the search for food. So it is in human nature to look for these things and it is taught in schools to help students improve their problem solving skills.

In modern time, data mining has grown along with computers providing means to do these kinds of tasks intelligently, from electronically stored and/or produced data. Theories and algorithms developed for digital computer systems help to search data more efficiently and detect information that could be hidden otherwise. This information can provide knowledge about the current and/or help to predict the coming events; for example, companies use data mining to detect shopping habits and patterns to strategize their sales and help them to provide more personal service for their customers. Data has also become a valuable asset that is being collected and sold. Big companies like Google, Facebook and Amazon collect and analyze data to gain useful information and harness it for their internal and external use. This also raises many questions about privacy because the digital footprint left behind the user can tell a lot about what a person likes, does and spends money on.

Witten et al. (2011) explains: “Data mining is about solving problems by analyzing data already present in databases. Suppose, to take well-worn example, the problem is fickle customer loyalty in a highly competitive marketplace. A database of customer choices, along with customer profiles, holds the key to this problem. Patterns of behavior of former customers can be analyzed to identify distinguishing characteristics of those likely to switch products and those likely to remain loyal. Once those characteristics are found, they can be put to work to identify present customers who are likely to jump ship. – More positively, the same techniques can be used to identify customers who might be attracted to another service the enterprise provides, one they are not presently enjoying.”

Description by Witten et al. (2011) for data mining, points that the search process should be fully- or semi-automated, and the discovered result patterns need to be meaningful so they can be used for beneficial purposes. These patterns should also make it possible to get more information from the new data.

Liu (2011) describes that the data mining task begins by getting understanding about the domain. This is usually done by the data analyst who finds the appropriate sources and data. Mining can be accomplished in three steps; it begins by pre-processing raw data from all the noise and abnormalities. In the case where there is too much data, sampling is applied, and required data attributes or features are chosen. The second step is mining the data to find patterns or knowledge. The last step is to post-process the mined results by choosing ones that are useful.

Web data mining is a field which has come to exist with the internet. Liu (2011) describes it in the following manner: “Web mining aims to discover useful information and knowledge from Web hyperlinks, page contents, and usage data. Although Web mining uses many data mining techniques, as mentioned above, it is not purely an application of traditional data mining techniques due to the heterogeneity and semi-structured or unstructured nature of the Web data.” Online search engines are examples how web data mining is used. Modern search engines crawl through, index and analyze millions of web-pages and their context enabling people to search information quickly and easily.

3.2 Neural Networks

The idea of creating an artificial neural network came from observing human brain functions. Human brains consist of huge network of cells also known as neurons that link together creating connections and communicating via electrochemical signals. These neurons link together in a topological network. This is also what the earliest approaches to artificial neural networks tried to model, the functions of biological brains. These neural networks have advantageous characteristics that can also be applied for digital systems. Computers are developed to provide help in doing complex calculations; however these tasks are commonly specific and the algorithms are engineered to fit the problem in hand. This means they are not flexible for changes without specifically re-programming the algorithms to meet the requirements.

Artificial neural networks can be used to solve complex problems by using simple mathematical operations. They can be applied to different problems, for example, pattern recognition, voice recognition, medical applications, business applications, signal processing and robotics. Following features are common in both biological and most of the advanced artificial neural networks. According to Kohonen (1997):

1. Analog representation and processing of information (which allows any degree of asynchronism of the processes in massively parallel computation without interlocking).
2. Ability to average conditionally over sets of data (statistical operations).
3. Fault tolerance, as well as graceful degradation and recovery from errors; this ability has a great survival value.
4. Adaption to changing environment, and emergence of “intelligent” information processing functions by self-organization, in response to data

For example, human brains lose brain cells everyday but it does not normally render people functionless nor prevent them from learning new things or performing daily tasks. Even in the cases of serious brain damage, it is known that other parts of the brain could replace the lost areas, recovering lost skills.

3.2.1 Biological Neural Networks

Biological neurons consist of three important components that can be seen in figure 1. *Dendrites*; one neuron can have up to hundreds of individual inputs that are connected to surrounding neurons where they receive signals. *Cell body* or *soma*; sums the incoming signals from dendrites and when the wanted input has been reached, causes the cell to fire the response output. *Axon*; transmits an output signal to connecting neurons when the cell has received the satisfactory input causing it to fire. These input signals are not weighted equally and some of them are more important than the others. In human brain this happens in the matter connecting axons and dendrites, the similar function is also modeled in artificial neurons by adding additional weight to some of the inputs. Neurons can be seen to function in a binary manner, but the rate in which they fire also adjusts the weight giving a different value to a cell that activates more or less often than the others (Fausett 1994).

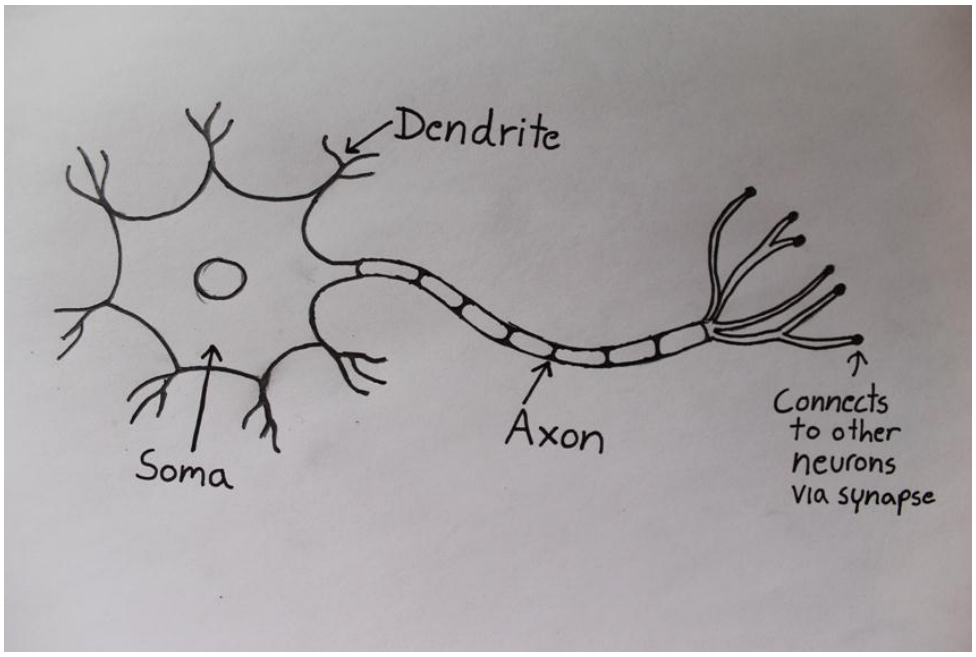


FIGURE 1. Biological neuron

3.2.2 Artificial Neural Networks

Artificial neural network consist of individual neurons, also called *units* that are in layers. They function in similar manner as their biological counterparts. Artificial neurons receive inputs that are summed together. An activation function is then used to decide when the neuron should fire. Upon activation an output is transmitted forwards. Artificial neural networks have three characteristics. *Architecture of the network* determines how the neurons are connected together. *Training algorithm* defines how the neurons learn from received input. And an *activation function* decides when the neuron has received necessary input causing it to fire. (Fausett 1994)

There are many different activation functions that have been developed and used in neural networks. One commonly used is a non-linear activation function also known as step activation function, see figure 2. It uses a hard threshold which means the activation function works in binary manner, having only two output states; it is either 1 or 0. So, the activation level is given by

$$f(x) = \begin{cases} 0; x \leq 0 \\ 1; x > 0 \end{cases} \tag{1}$$

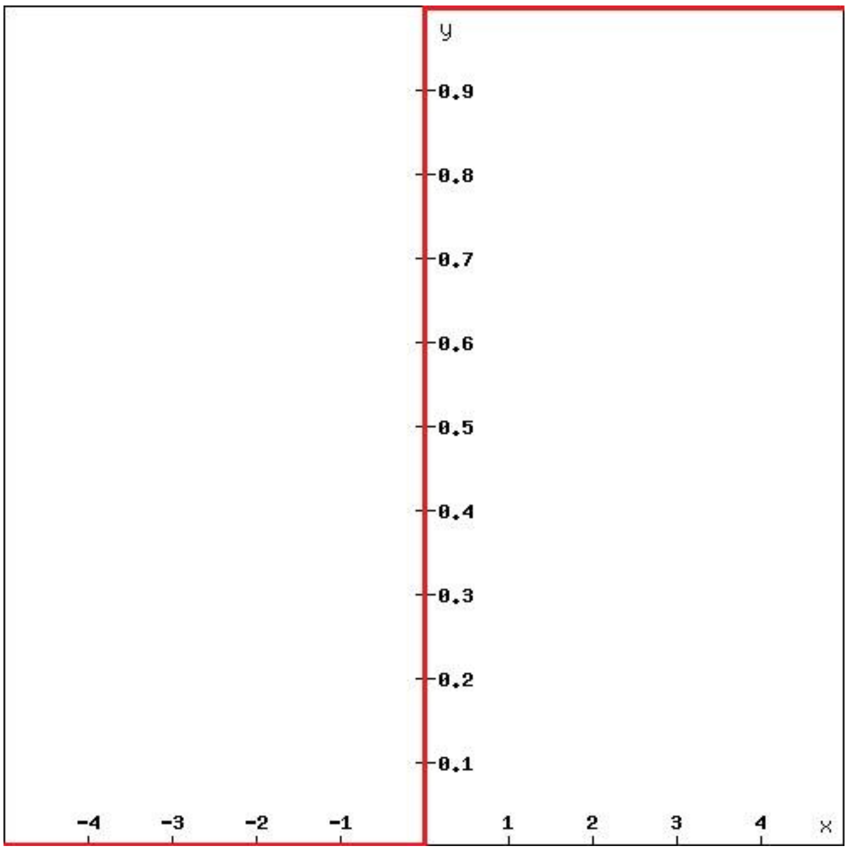


FIGURE 2. Non-linear activation function

Another commonly used activation function is called sigmoid function, which is a logistic function, see figure 3. Sigmoid perceptron uses a soft threshold therefore it can have an output value between 0 and 1. The activation level is given by

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

These activation functions can also be scaled to meet the output requirements, for example, by making the sigmoid function to output value between -1 and 1 (Fausett, 1994). This is called as bipolar sigmoid function.

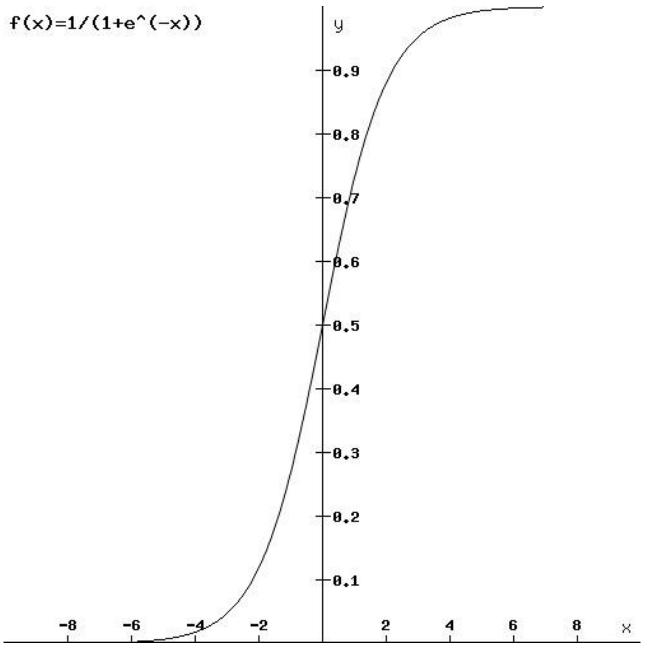


FIGURE 3. Sigmoid function

Training algorithm defines learning behavior of neural network. First of all, there are two different types of learning schemes; supervised and unsupervised learning. Supervised learning is the most commonly used method for teaching artificial neural networks. Learning is performed by training the network with pairs of input - output vectors. This method requires a prior knowledge about the data and is commonly used in pattern recognition.

Unsupervised learning is used in training self-organizing networks like so called Kohonen Self-Organizing Maps. Unsupervised learning algorithms use only input vectors for learning, no target outputs are provided. Network self-adjusts in to clusters that present the typical cases learned from provided input vectors, and the amount of neurons define the maximum amount of different classes that can be found. Unsupervised learning is effective when the expected target weights and class memberships are not known in advance.

Error functions are used to determine how well neural network performs, by finding the difference between calculated output and the given target output. Knowing the rate of error helps adjusting the neurons in network layers to match closer to the targeted output. Commonly used error functions include Euclidean distance and mean-square error.

In the following, three common network architectures are shortly explained. *Single-layer feed-forward networks,* see figure 4, have only one layer of connecting weights w_{ij} that are received from *l* input neurons x_i. Each input neuron x_i is connected to all outputs y_j that fire when activation function is satisfied (Fausett 1994), by

$$y_j = f_j\left(\sum_{i=1}^{I} w_{ij} x_i\right) \tag{3}$$

The name feed-forward describes the flow in the network which goes only one direction from input layer to output layer. Perceptron and Adaline (Adaptive Linear Neuron) (see Fausett 1994) are examples of single-layer feed-forward networks. Applications for these networks can be found, for example, in the field of pattern recognition.

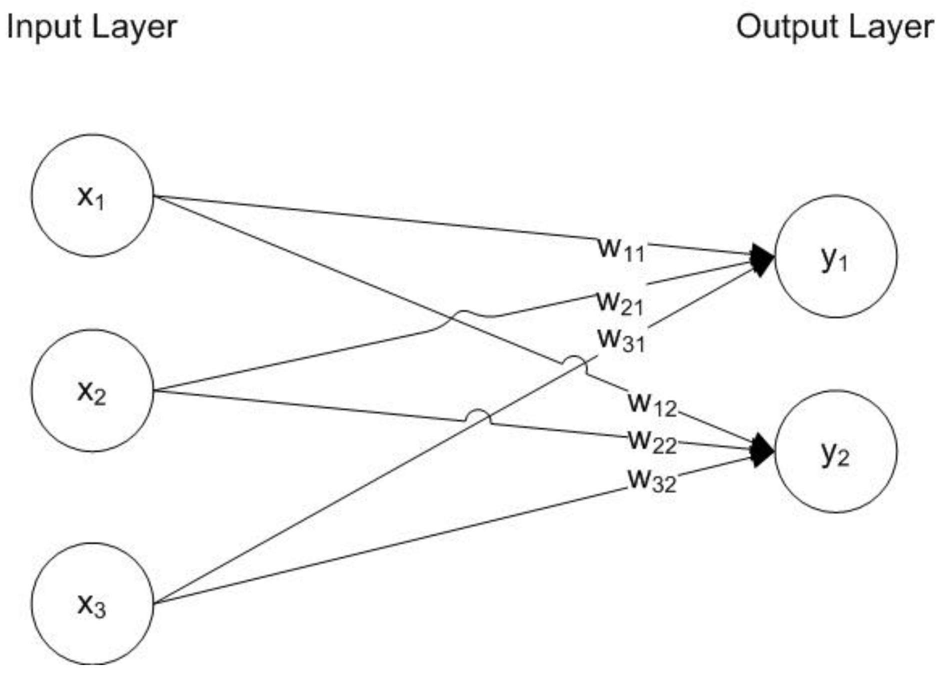


FIGURE 4. Single-layer feed-forward network

Multi-layer feed-forward networks were created because the limitations of single-layer networks were known. They can have from one up to *n* layers of neurons. Commonly the layers between inputs x_i and outputs z_j are called hidden layers that consist of hidden units y_k. In feed-forward network all the layers receive inputs from upstream and send output to next layer.

As seen in Figure 5 input weights w_{ij} for hidden layer are received and the output y_j calculated in hidden layer is forwarded to output z_m using weights $h_{jm.}$ Multi-layer networks can solve more complex problems compared to single-layer but they can be harder to train. Training algorithms like back-propagation have been developed for teaching multi-layer feed-forward networks. The so called *Universal Approximation Theorem* says that any multi-layer feed-forward network with one hidden layer represents a continuous function under mild assumptions on the activation functions.

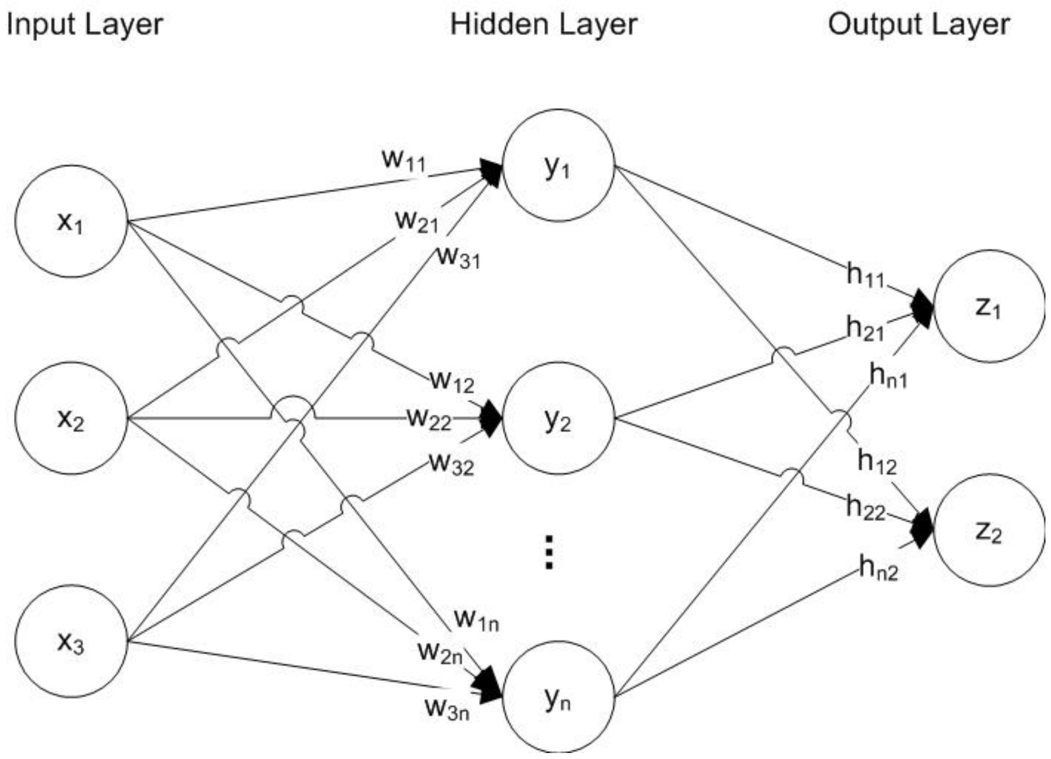


FIGURE 5. Multi-layer feed-forward network

Recurrent networks, see figure 6, add an important feature to previously mentioned structures. They give the network a short-term memory which enables it to remember the previous outputs. This can be useful, for example, in pattern recognition and it helps speeding up the training process. Recurrence is performed by making the hidden layer neuron or neurons to feed own calculated output back to its inputs. Old outputs are then taken in to account when calculating the new output.

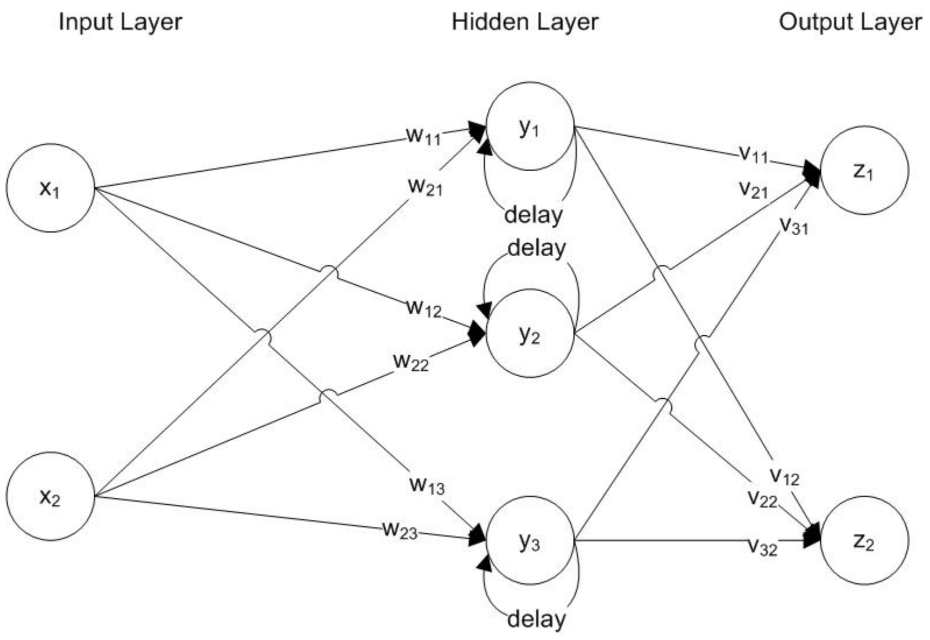


FIGURE 6. Recurrent network

3.2.3 Kohonen Self-Organizing Maps

Suggested by Teuvo Kohonen (Kohonen 1984) *Kohonen Self-Organizing Map* (SOM) is a topology-preserving unsupervised neural network that forms topological structure among the cluster units (Fausett 1994). This kind of network can be used to visualize and analyze high dimensional data, acquired by calculating and mapping the input data into clusters that present the typical cases. Because the training is performed *un-supervised*, no target outputs are required for teaching. Kohonen Self-Organizing Maps, see Figure 7, consists of *m cluster units* that are in one- or two-dimensional array, inputs are received as *n*-tuples (Kohonen 1989). This neural network approach differs from the previous ones introduced in the previous chapter by its structure because the SOM-layer also serves as an output.

Self-organizing maps can also be a hybrid-model. Together with other types of neural network layers, for example, *Counter Propagation neural network* (see Graupe 2007, 161 - 166). The network uses *Kohonen-layer* together with *Grossberg-layer* in which case the Kohonen-layer is used to cluster the similar clusters through the *"Winner-*

Takes-All" method and then the Grossberg-layer is applied to calculate the wanted output. This setup enables the network to generalize received input.

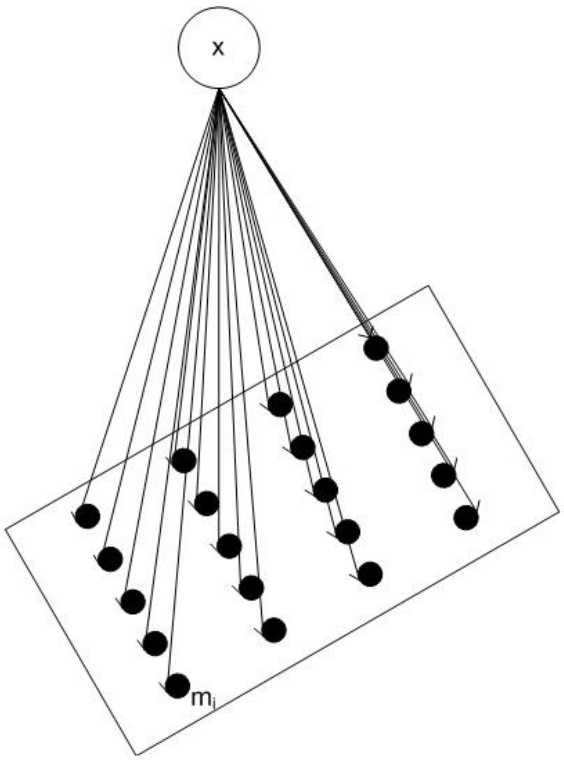


FIGURE 7. Kohonen SOM

Weight-vectors presenting each cluster in the network determine the sample input pattern matching the target cluster. Network is trained through "Winner-Takes-All"-method which means only the cluster-unit matching closest to the given input is chosen. In the beginning, no target outputs are given and the received input is treated as the desired output for the chosen cluster unit. Cluster units are called *reference vectors* in this work. Reference vector and its neighboring reference vectors are updated to match closer to the received input pattern. This closeness between input vector and each cluster is commonly determined by using the squared minimum Euclidean distance to decide which one is chosen as the winner. In the case where there is more than one equally distanced unit, the winner is decided by randomly choosing between matching units. So, for an input data vector **x** and reference vectors $\mathbf{m}_i$ the best matching reference vector $\mathbf{m}_c$ satisfies (Kohonen 1997)

$$c = \operatorname{argmin}_i\{\|\mathbf{x} - \mathbf{m}_i\|\} \tag{4}$$

After determining the winner, neighboring cluster units that receive updates are decided by given *radius R* and the units found by using so called neighborhood function are updated. Suggested by Kohonen, the beginning of the radius should be around half of the largest dimension of the array.

Training of the network is performed over time and the input data vectors are cycled on each time moment t. These moments in time are also called as *epochs*. At the time moment t, all the reference vectors $\mathbf{m}_i$ that belong to the set of neighboring vectors $N_c(t)$ have learning rate $\alpha(t)$ otherwise the learning rate is 0 and no learning happens. Thus we have learning rates

$$h_{ci} = \begin{cases} \alpha(t); \ \mathbf{m}_i \ \in N_c(t) \\ \ 0; \ \text{otherwise} \end{cases} \tag{5}$$

Calculating the updated reference vectors given by (see Kohonen, 1997)

$$\mathbf{m}_i(new) = \mathbf{m}_i(old) + h_{ci}[\mathbf{x} - \mathbf{m}_i(old)] \tag{6}$$

Both learning rate and neighborhood of the winning cluster unit $\mathbf{m}_c$ degrade over time. Kohonen (1997) explains the ordering of the clusters happens approximately during the first 1000 steps and then the rest of the steps are needed for fine adjustments. Therefore, in the beginning, the learning rate should be close to the initial learning rate and then decrease later on. However there is no importance in choosing between linear, exponential or inversely proportional time function.

```
Step 0    Initialize reference vectors $\mathbf{m}_i$
          Set topological neighborhood parameters
          Set learning rate parameters
Step 1    While stopping condition is false, do Steps 2-8.
          Step 2    For each input vector $\mathbf{x}$, do Steps 3-5.
                    Step 3    For each $i$, compute:
                              $D(i) = \sum_j (m_{ij} - x_j)^2$
                    Step 4    Find index I such that D($i$) is a minimum
                    Step 5    For all reference vectors $\mathbf{m}_i$
                                $\mathbf{m}_i(new) = \mathbf{m}_i(old) + h_{ci}[\mathbf{x} - \mathbf{m}_i(old)]$
                    Step 6    Update learning rate
                    Step 7    Reduce radius of topological neighborhood at
                              specified times.
                    Step 8    Test stopping condition
```

FIGURE 8. Algorithm, (see Fausett, 1994)

Algorithm in figure 8 described by Fausett (1994) presents a pseudo-code structure for creating topology-preserving Kohonen Self-organizing maps.

3.3 Other learning models

Along the neural networks, there are other machine learning methods. *Support Vector Machines* (SVM) is possibly the most popular supervised learning method used in sentiment analysis and text classification currently. It has a strong theoretical basis and has been found as most accurate when supervised learning is possible. SVM’s are able to generalize well which leads to its high accuracy. Another popular learning model is *Bayes Network* which is based on probabilistic reasoning and it can be regarded as a graph that indicates associations between random variables in the network. Probabilities can then be calculated for given input indicating the likelihood that it belongs to a certain category. More about Bayes networks can be read from books written by previously mentioned Russell & Norvig (2011) and Liu (2011).

3.3.1 Support Vector Machines

This section is based on Liu (2011, 109 - 124). As stated before, support vector machines maybe the most popular approach for sentiment analysis and classification. Multiple studies have been conducted evaluating differences between machine learning techniques and support vector machines have been found to be very accurate in most applications, especially the ones required to handle high-dimensional data.

Support Vector Machines (SVM) is a linear learning system that creates a two-class classifier. The training examples used for teaching SVM are a set of *input vector* and *class label* ($\mathbf{x}_i$, y_i) pairs, where input vectors are real-valued *r*-dimensional and class labels are class where they belong, for example, 1 or -1 which can also be denoted as $y_i \in \{1, -1\}$. And b is a real-valued *bias*. In the task to create a classifier, the first task is assigning input vectors into positive and negative classes; according

$$y_i = \begin{cases} 1; \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1; \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases} \tag{7}$$

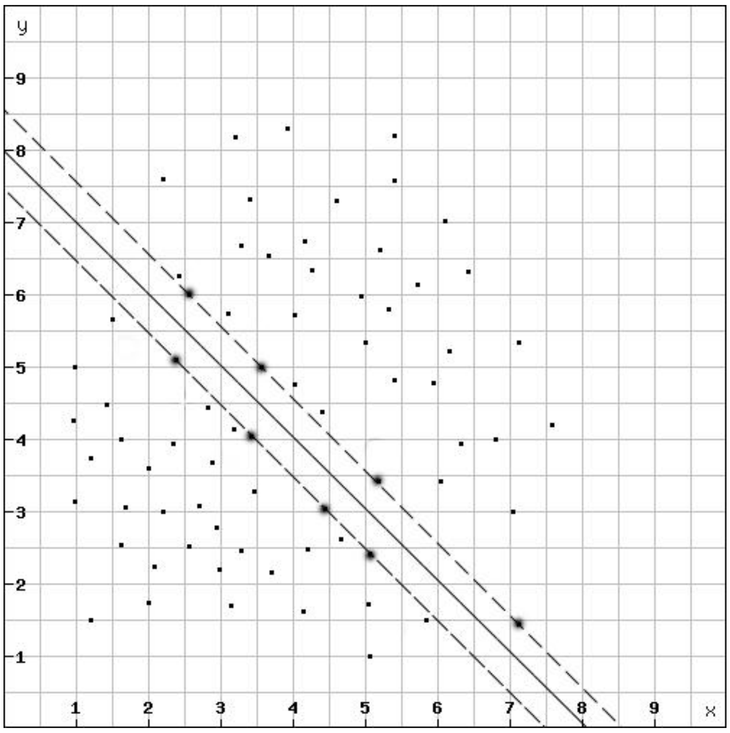


FIGURE 9. Linearly separable dataset

This section is based on the description about *Support Vector Machines* by Liu (2011, 109-124). The goal for SVM is to find a *hyperplane* that is separating positive and negative samples in two distinct groups as seen in figure 9. Other names used for the hyperplane are decision boundary and decision surface. Hyperplane is a line in 2-dimensional space and plane in 3-dimensional place.

In general there are 3 distinct cases that require different approaches when SVM is used. These cases are caused by the input data and how it is located in the vector space. The goal of SVM is to maximize the margin between hyperplane locating it as far from both positive and negative data points as possible, and at the same time still separating these two classes. This is achieved by finding the data points that are closest to the hyperplane and they are used to support it, thus the name Support Vector Machines. The easiest and most clear case happens when all the data points are clearly linearly separable in two classes. Then a maximal margin hyperplane can be found and it presents the decision boundary.

The second case is when all the data points cannot be separated in two distinct classes. This case is often the most common due to noisiness in the data. In the first case, it would have been impossible to solve the problem of finding a satisfactory hyperplane. However this can be fixed by introducing a so called soft-margin SVM, which allows errors to exist among the labeled data points. The final decision boundary is then found by taking into account all the data points that are on the margin, in the margin and the ones that are error cases.

The third case is called kernel functions. They extend the SVM to work with nonlinearly separable data and use the same formulas and techniques as given for linear data. Nonlinear data located in input space is transformed in to higher-dimensional space which is called feature space. This enables SVM to find a linear decision boundary which separates the data in to positive and negative classes. Kernel functions are used for finding the separating plane which requires calculating the dot products for the data points in transformed feature space.

Though Support Vector Machines can be used for solving many problems, there are still some limitations that need to be known. Liu (2011, 109-124) describes the limitations as following:

1. It works only in real-valued space. For a categorical attribute, we need to convert its categorical values to numeric values. One way to do this is to create an extra binary attribute for each categorical value, and set the attribute value to 1 if the categorical value appears, and 0 otherwise.

2. It allows only two classes, i.e., binary classification. For multiple class classification problems, several strategies can be applied, for example, one-against-rest, and error-correcting output coding.
3. The hyperplane produced by SVM is hard to understand by users. It is difficult to picture where the hyperplane is in a high-dimensional space. The matter is made worse by kernels. This, SVM is commonly used in applications that do not require human understanding.

4 SENTIMENT ANALYSIS AND TECHNIQUES

Sentiment analysis is an application of natural language processing used for finding and extracting opinionated information from variable sources. Finding this information can provide deeper knowledge about a topic, feature, writer or the document in hand. Sentiment analysis or opinion mining has gained a lot interest in the past decade because the internet has provided such a vast sea of opinionated documents that are now available for anyone.

Before the internet was so common, opinions were shared by asking friends, co-workers or relatives but now it is all available on the internet. People can search and read opinions written by strangers about everything a man can imagine. Web-sites like Epinions, Amazon and IMDB collect reviews written by customers and have them available for everyone. Along with big sites mentioned, Web 2.0 technologies have encouraged individuals and professionals to express their opinions and ideas in blogs, forums and social media. These expressions are then soaked in by readers and sometimes used to help in the decision process.

This ability to share opinions has given them a lot of weight. Companies, individuals and institutions now acknowledge that bad reviews, blog posts or comments cannot just be ignored. Bad reviews can ruin the reputation of a person or whole company if nothing is done to fix it. Therefore, there is a need for technology that helps to follow what is being said and for an easy way to detect and show specific opinions. So, conceptually data-mining generally is not enough for opinion mining. Opinion mining demands its own methods, perhaps based on natural language processing, which may apply data-mining techniques.

Sentiment analysis can be used to detect and classify texts or documents in *polarity groups* like; *Positive*: “The band played well”, *Neutral*: “The band was playing” or *Negative*: “The band played badly”. These opinions are then collected from documents and overall polarity is determined. *Subjectivity information* can also be detected from documents, like from the sentence “I was looking at the pre-match statistics and I believe they made the worst possible choice.”

Aspect-based opinion mining or *feature-based opinion mining* can be used to find specific opinions from documents instead of overall polarity groups, because often documents contain both positive and negative opinions. Finding and extracting individual opinions can be useful when dealing with documents that contain opinionated comments about features: “The price of these shoes is too high” where “price” is the feature and “too high” is the opinion related to it. However finding the opinion about the price does not tell the overall opinion of the shoes. Another use case for sentiment analysis is finding *opinion holders* and *targets*, for example, in a sentence: “Ville said this chair is uncomfortable”, “Ville” is the opinion holder and “chair” is the target receiving the opinion.

In this thesis, only *document-level sentiment classification* will be discussed and covered more in-depth. More about the other tasks related to sentiment analysis can be read from the book: *Web Data Mining* by Liu (2011) and from a survey *Opinion mining and sentiment analysis* by Pang & Lee (2008).

Finding opinion polarity with a sophisticated computer system is a complicated task and it identifies many problems that need to be solved before an effective system could be created. Solving the problem can be approached by using data mining and/or knowledge-based approaches. According to Liu (2011) there are following problems that need to be solved.

The first problem is finding the opinion target. This means finding the *entity* (term suggested by Liu) that is being evaluated and then the aspects that together form the entity. These entity-aspect relations could also be denoted with hierarchical tree as seen figure 10. This example tree is not complete and does not cover all the possible aspects that continue to lower sub-levels, but will hopefully demonstrate the concept.

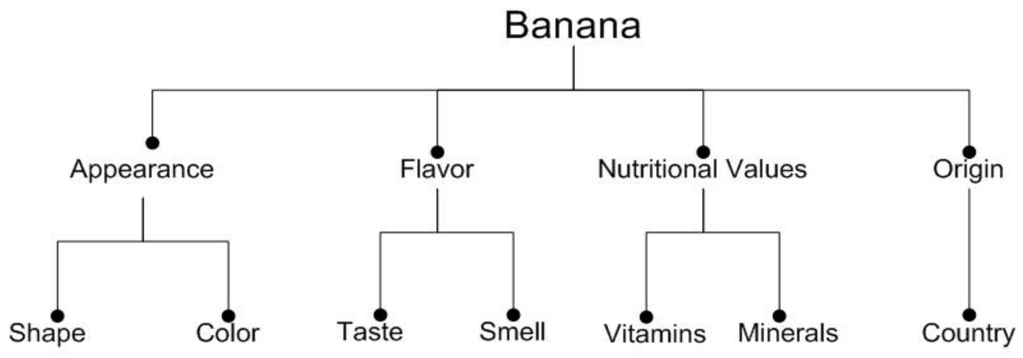


FIGURE 10. Hierarchical tree for entity *banana*

Aspect consists of attributes and components related to the entity, for example, in the sentence “Company A is successful, their campaign went really well however their attitude is not good.” We can see the entity “Company A” and the related opinion word “successful”. Then we have the aspect “campaign” with the opinion “went really well” and the aspect “attitude” with the opinion “not good”. These aspects could also have sub-aspects, however due the complexity of the described details and because it can be difficult to understand the hierarchical representation, these sub-levels can be simplified and flattened under the term *aspects*, mapping all the aspects on second level under the root level entity. (Liu 2011)

The second task is to find the opinions. There are commonly two types of opinions; *comparative* and *regular*. Comparative opinions usually evaluate two or more entities and express the differences between them and/or they can resemble the preference of opinion holder based on some of the shared aspects of the entities**.** Regular opinions are commonly positive or negative views, attitudes, entities, emotions or appraisals about an entity or aspect that comes from the opinion holder. The term *opinion holder* denotes a person or organization that holds an opinion. (Liu 2011)

The main part of opinion mining task is finding a set of features that are used to describe sentiment information in the document, for example, the following list of features, composed by Liu (2011) have been used by researchers:

- *Terms and their frequency*. Term can be a single word or *n*-gram (*n* contiguous words) and frequency is how often term appears in a document. A study done by Pang & Lee (2002) has shown that simple term presence (*true*/*false*) in a document may be a better indicator for opinion than the term frequency.

- *Part-of-Speech*. Studies have shown that certain word categories like adjectives, nouns and verbs can be good indicators for opinion.
- *Opinion words and phrases*. Common positive words like amazing, beautiful, good and negative words like bad, horrible, ugly indicate an opinion. Opinion phrases and idioms like “worth its weight in gold” are also important opinion indicators but they can be hard to find without prior knowledge.
- *Negations* are important and they have to be handled carefully. A single negation can turn the polarity of whole sentence, for example, “I like this movie” vs. “I do not like this movie”.
- Other features include *syntactic relations* and *rules of opinions* that try to indicate opinions in a deeper linguistic level than by simple opinion words or phrases.

To find a document-level opinion some assumptions need to be made. First of all it is assumed that the whole document is related to a single *entity* and there is only one opinion holder. This makes reviews more ideal because they are commonly written by one person about a single product or service. For example, in the case of forums and blogs there can be multiple entities that are mentioned, evaluated and/or compared together. This means there are multiple opinions in the same document and finding the overall sentiment over multiple entities does not provide relevant information. There can also be multiple opinion holders distorting the results with different opinions. Document-level opinion does not either provide information about specific aspects. In some domains the location, end, middle or beginning, where the opinion has been expressed can carry more weight. This is the case in movie reviews; often people write their conclusion in the end which carries the overall opinion of whole document. (Liu 2011)

There are also general problems that make opinion mining difficult. Opinions are often subjective. Subjectivity can be causal effect of domain, culture, environment and/or experiences. For example, a single sentence may have a different meaning depending on the domain. Sentence “Scariest experience in my life” can have a positive meaning if someone is telling about their time in an amusement park, however if it is about test driving a new car, it can be assumed that it does not express anything positive for most. When it comes to natural language, there are indirect ways to express opinion. For example, saying "My feet felt much better after I started using these shoes." we can see that there is a subtle opinion that is telling the shoes are good. (Liu 2011)

4.1 *N*-gram word model

The *n-gram word model* is a method of finding a set of *n*-gram words from a given document. The most commonly used ones are *unigram*, *bigram* and *trigram* models. *N*-gram words are used, for example, in *text prediction*, *categorization*; this can be a task to detect email spam, classify documents in specific domains like sports, music and movies, also for other natural language processing purposes like sentiment analysis.

EXAMPLE 1. Unigrams

Text: *"A fat cat sat on a hat and talked to a rat"*
Unigrams: *"A", "fat", "cat", "sat", "on", "a", "hat", "and", "talked", "to", "a", "rat"*

EXAMPLE 2. Bigrams

Text: *"A fat cat sat on a hat and talked to a rat"*
Bigrams: *"A fat", "fat cat", "cat sat", "sat on", "on a", "a hat", "hat and", "and talked", "talked to", "to a", "a rat"*

EXAMPLE 3. Trigrams

Text: *"A fat cat sat on a hat and talked to a rat"*
Trigrams: *"A fat cat", "fat cat sat", "cat sat on", "sat on a", "on a hat", "a hat and", "hat and talked", "and talked to", "talked to a", "to a rat"*

The simplest model of these is the unigram model. A set of unigrams consist of all the individual words in a document as seen in example 1. Bigrams are set of word pairs that appear adjacent to each other in a document. Each of these word pairs forms a single bigram, as can be seen in example 2. Trigrams are a set of words that consist of three adjacent words appearing next to each other (example 3). In *n*-gram model same words can appear multiple times because the task is to find all the possible matches that meet the criteria.

Bigram and trigram words are considered more informative compared to unigrams words that present quite general view and alone they are not enough for tasks like sentiment analysis. Bigrams and trigrams, on the other hand provide more information about word location and probabilities that give better knowledge about the document in hand. *N*-grams are extensively utilized in this thesis.

4.2 Part-of-Speech tagging

Part-of-Speech (POS) tagging is a method of detecting *lexical categories* of words in documents and labeling them accordingly. Common linguistic word categories include; *adjectives*, *adverbs*, *verbs*, *nouns* etc. There are various methods how the detection can be achieved. One commonly used approach is using statistical techniques to calculate the probabilities how often words appear next to each other within the used text corpus. Sometimes, the words may belong to multiple categories depending on how they are positioned in a sentence. In such situation knowing the likelihood for that specific position may help to determine the correct category. More information about the part-of-speech tagging methods can be found, for example, in the previously mentioned book written by Russell and Norvig (2011).

Penn Treebank Part-of-Speech tags, introduced by Santorini (1990), are used in this work and they are found in *Natural Language Toolkit* introduced in section 2.3 that was used for POS tagging document. The complete list of tags can be seen in table 1. POS tags are used because they have been found to be an excellent method of detecting characteristics in the text.

TABLE 1. Part-of-Speech tags

CC - Coordinating conjunction	PRP$ - Possessive pronoun (prolog version PRP-S)
CD - Cardinal number	RB - Adverb
DT - Determiner	RBR - Adverb, comparative
EX - Existential there	RBS - Adverb, superlative
FW - Foreign word	RP - Particle
IN - Preposition or subordinating conjunction	SYM - Symbol
JJ - Adjective	TO - to
JJR - Adjective, comparative	UH - Interjection
JJS - Adjective, superlative	VB - Verb, base form
LS - List item marker	VBD - Verb, past tense
MD - Modal	VBG - Verb, gerund or present participle
NN - Noun, singular or mass	VBN - Verb, past participle
NNS - Noun, plural	VBP - Verb, non-3rd person singular present
NNP - Proper noun, singular	VBZ - Verb, 3rd person singular present
NNPS - Proper noun, plural	WDT - Wh-determiner
PDT - Predeterminer	WP - Wh-pronoun
POS - Possessive ending	WP$ - Possessive wh-pronoun (prolog version WP-S)
PRP - Personal pronoun	WRB - Wh-adverb

For example, adjectives and adverbs are known to contain opinionated information. The certain word categories that appear together as bigrams and trigrams, for example, an adjective being followed by a noun are known to be containing opinion information. This information is used in this thesis to extract bigrams from the documents.

4.3 PMI-IR Algorithm

Pointwise Mutual Information and *Information Retrieval* (*PMI-IR*) is an unsupervised learning algorithm first suggested by Turney (2001) for recognizing polarity of synonyms and later (2002) to classify reviews as *recommended* and *not recommended*. The

PMI algorithm is used to measure *Semantic Orientation* (*SO*) for a given *phrase*. PMI is the degree of statistical dependence that $word_1$ and $word_2$ co-occur, given by

$$\text{PMI}(word_1, word_2) = \log_2\left(\frac{p(word_1 \& word_2)}{p(word_1)\, p(word_2)}\right) \tag{8}$$

The SO measure for each individual phrase is calculated based on the affiliation with a positive reference word "*excellent*" and negative reference word "*poor*" and it is given by

$$\text{SO}(phrase) = \text{PMI}(phrase, \text{"excellent"}) \text{ - } \text{PMI}(phrase, \text{"poor"}) \tag{9}$$

The terms “poor” and “excellent” are used because often 5 stars or point review systems use these words to describe the lowest and highest ends in the rating scale. Therefore the Sematic Orientation gets higher when the given phrase is more often associated with the word “excellent” and lower, if the phase exists more often with the word “poor”.

To get the PMI-IR score, a set of queries is deployed to a search engine and the number of received *hits* is recorded. The original research by Turney (2001) used AltaVista search engine and the search queries were done by using AltaVista’s *NEAR* operator, which allows searching terms existing within the defined number of words from each other. The distance Turney (2002) used was 10 contiguous words. Now, the Semantic Orientation of a phrase, where *hits*(*phrase*) indicates one search query, can be rewritten as

$$SO(phrase) = \log_2\left[\frac{hits(phrase\ NEAR\ \text{"excellent"})hits(\text{"poor"})}{hits(phrase\ NEAR\ \text{"poor"})hits(\text{"excellent"})}\right] \tag{10}$$

Zero division can be avoided by adding 0.01 to all the hits (Liu 2011). In this thesis, Bing search engine was used for querying the semantic orientations because currently it is the only mainstream search engine offering a free *API* access for doing the searches through queries which is an important aspect. Bing search engine was also chosen because the previously mentioned AltaVista search engine does not exist anymore as it was during the time when Turney’s research was done. Bing also offers the

NEAR operator in the searches, however it is not supported in the API, so the *AND* operator was used instead. The differences between AND and NEAR operators were experimented by Turney (2001). The algorithm was tested using 80 synonym test question from *Test of English Language as a Foreign Language* (*TOEFL*) and 50 synonym test questions from a collection of tests for students of *English as a Second Language* (*ESL*). In the TOEFL tests AND operator was about 10% less accurate than the NEAR operator, still resulting in better scores than just arbitrarily choosing between the two options. The results in ESL tests were much lower for AND operator resulting in about 48% accuracy, whereas NEAR was still above 60%.

The biggest limitations Turney (2001) noted for this algorithm were the high network traffic caused from querying the search engine, which may not be so problematic anymore due the evolved connection speeds.

One possible solution Turney (2001) proposed was using a hybrid system that incorporates a local search engine for searching the most common words and then, relying the big search engines when the search term is not so commonly known. This is a viable solution that could be accomplished quite easily these days. Powerful search server software like, for example, *Sphinx* and *Apache Lucene*, that index and allow searching within local documents and databases could be used for this tasks if a good set of documents are available.

The problem that might exist now is the access to the search engines which is becoming limited. The only efficient and legal way of doing automated queries with a computer system is by using the API’s. They often charge the user by the amount of sent queries which can get costly with a system that heavily relies on using an external search engine. Of course once the query is done, the calculated score can be stored into a database along with the phrase for later use. In the beginning when the lexical database that stores these results is generated, the time required to query all the phrases can be quite long, because often there are set limitations how often one should query the search engines during short period of time.

4.4 *K*-fold cross-validation

Described by Russell and Norvig (2011), the idea of *k-fold cross-validation* is to evaluate how well a learning system performs, by first splitting the used dataset in to *k*-subsets. Each of these subsets then serves, both as the training data and test data. On each *k*-round one subset is chosen for testing, and the other subsets are used for teaching. Each subset is used only once for testing. Calculating the average score received from these test rounds is proposed to give better estimation than using just one score. However, this evaluation method is mainly usable when labeled datasets are available because it requires prior knowledge about the data.

5 APPROACHING THE PROBLEM

This thesis searches for a solution for document-level opinion mining from the documents belonging to various domains. With sentiment analysis the given documents can be classified into opinion categories. The commonly used ones are positive and negative. Sometimes neutral is also included which may be considered as having no opinion or that there is same amount of positive and negative opinions. The approach is not domain dependent which means the attempt is to find features that are common in all the domains, for example, no matter if the article is a movie review, a blog post, tweet or news article. This can be complicated task because opinion is often subjective, and some of these domains are hard to classify due their nature. The biggest complications may rise from social media, forum and blog posts that can often contain more than one entity and have multiple opinion holders, making document-level sentiment analysis hard and in worst case the results are irrelevant when examining the document content. However there are some features that may be used to give us a good overview.

The thesis work started by studying the existing techniques that have been researched and tested in the past. Multiple books and research papers, some that are mentioned in the chapter 1, were read to find features that are commonly used. Previously, multiple studies have been done in the area that incorporates supervised machine learning. Choosing Support Vector Machines as the classifier would have been a valid choice based on their good performance in sentiment classification as described in chapter

3.3.1. However for this thesis unsupervised approach was chosen, due to the fact that it is hard and time consuming to create a good set of labeled training documents that would cover various domains. Creating a set of labeled training documents is a task that is necessary if supervised learning is used. *Kohonen Self-Organizing Maps* was chosen as the unsupervised classifier as it is well known for being able to classify high-dimensional data into clusters according their similarity without need to have prior knowledge about the data.

Another idea might be building a hybrid system that uses different opinion mining methods and combines them together to provide more accurate classification; this can lead to better results as demonstrated by Prawobo and Thelwall (2009).

Creating a functional sentiment analysis system is a big task and requires a lot of work. The assignment given by Cluetail Ltd. was to design a system which could be used to classify documents according to their opinion orientation by using Python programming language. Sets of existing documents are used for training and teaching the system, and also used to confirm the results how well the system performs.

5.1 Choosing document features

It became clear quite early in the work that having a good feature set which can describe the opinion information is important. It is perhaps the most important thing in sentiment classification process. Choosing the used features was not an easy task, and some of the ideas required complex systems to be made. So far there is no universal solution that would work for everything.

The first chosen feature was *term frequency* which is simply the count on how often positive and negative words appear within the given document. This is the simplest of all the features and alone will not provide good results. It is a naive approach and does not represent any deeper context nor provide "understanding" about the content. However when combined together with other features it may give a good but sometimes unreliable direction. This is a lexicon based feature and the accuracy can get better together with the increasing amount of positive and negative words that are known. The second used feature is expanding set of over 4000 *bigrams*. These bigrams were extracted by using Part-of-Speech rules introduced in chapter 4.2 and their semantic orientation (Turney 2001) is calculated with PMI-IR algorithm as described in the chapter 4.3. Later on, new terms were automatically extracted and added to the database when they were discovered from the documents.

5.2 Generating lexical database

Building the lexical database could have been considered both a supervised and unsupervised approach. A corpus of positive and negative opinion words composed over the years by Hu and Liu (2004) was used for term detection. Because the words had been tagged a priori, the task was just to transfer them to the used database. A total of 6785 words were extracted and stored according their polarity.

In case of the bigrams; all the bigrams matching the given POS tag rules were extracted and two attributes calculated for each bigram. First the overall appearance in the dataset was calculated. Next, the amount of documents that contained each bigram was calculated. This process was inspired by *Term Frequency – Inversed Term Frequency (TF-IDF)* technique used in data mining to evaluate how important a given term is within a set of documents, however the process was not the same because the aim was to find the most commonly occurring items within the dataset.

The dataset where bigram rules were extracted is a set of over 5.8 million Amazon product reviews collected by Jindal and Liu (2008) for their paper “Opinion Spam and Analysis.” This dataset consists of individual product reviews extending over 20 categories. The reviews come with rating scores ranging from 1.0 to 5.0; however all the reviews with 3.0 rating had been removed. That was a small downside because including the so called *neutral* ones would have been preferred, but it did not affect building the lexicon.

A set of over 100 000 terms was extracted from the previously mentioned Amazon product review set, and they were stored in a database. From these, all the terms with less than average appearance value were directly eliminated, which left a set of around 14 000 terms. After doing more manual cleaning in the dataset, about 4400 bigrams were left and the PMI-IR algorithm was used to calculate the semantic orientation for these bigrams which then served as the base lexicon. All the scores were normalized and scaled so scores with *semantic orientation* over 2.0 and less than -2.0 were updated to match these set boundaries. An implementation was also made which extracted and calculated semantic orientation for all the unknown bigrams that matched the part-of-speech rules. This way the system was enabled to expand its knowledge because clearly, 4400 bigrams is far from enough to cover all the necessary terms in English-language.

The reason why product reviews were used to create the lexicon (although, this thesis is presenting a domain independent approach), is because product reviews are good sources for finding strong opinions. By extracting the most often appearing opinionated bigrams from multiple domains within the reviews, a set of the most general opinion phrases may be generated and then scoring for these can be calculated.

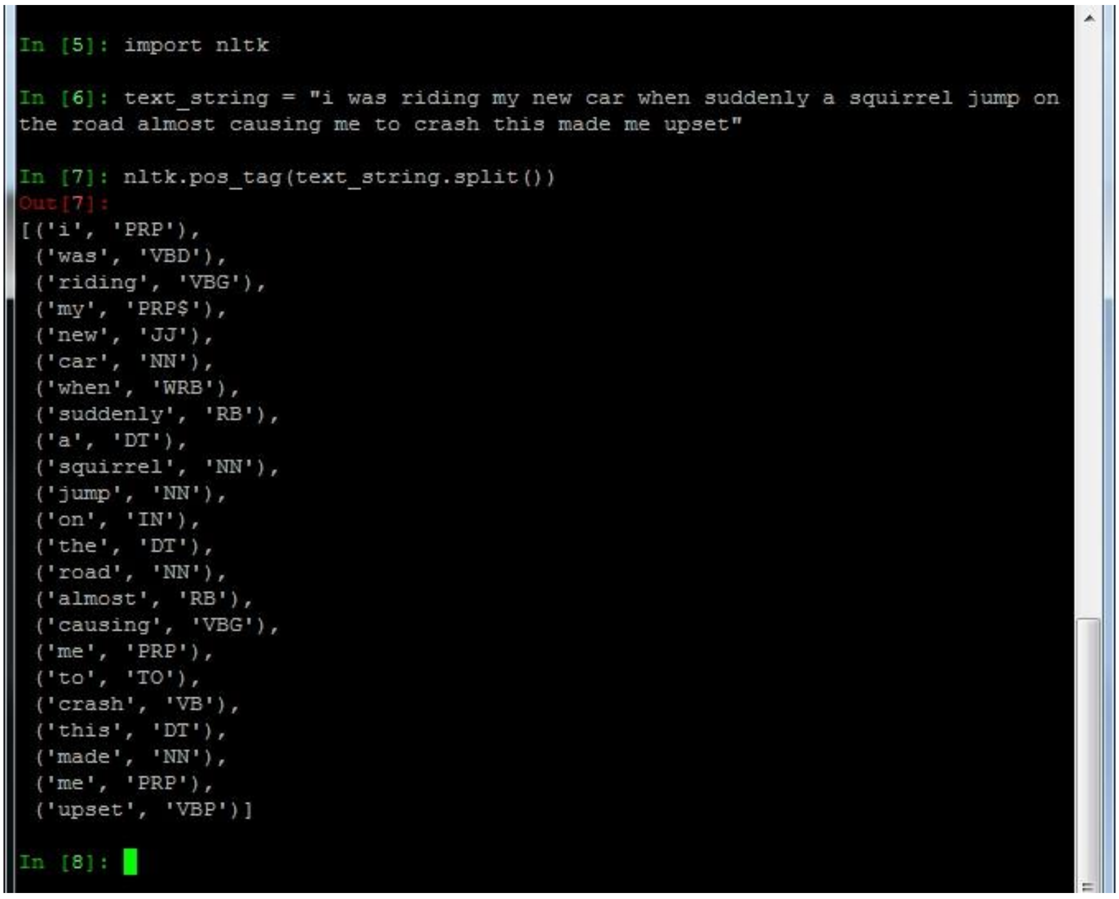
5.3 Extracting the document features

The first task in extracting the document features is cleaning the used documents. It was known from the beginning that the system will always have access to relatively clean documents; therefore, the only thing left for the system to do was to remove all the special characters like dots, commas, quotation marks etc. Removing the characters was done in Python by using regular expressions and string operations that allow to search, substitute and remove unwanted characters within a given string.

The next task was extracting features introduced in chapter 5.1.1. First all the terms t within the given document D where $t \in D$ are extracted, possible negations are checked and the words are added to set T_+ *or* T_- according terms positive or negative orientation. Deciding whether a term is positive or negative is first done by querying word lexicon described in chapter 5.1.1, and then checking the possible negation. All the unknown terms are ignored.

To formulate a document-level feature vector, three different features are determined (roughly) as follows. The feature $score_1$ represents term appearance. It is calculated from the amount of positive term appearances T_+ and negative term appearances T_- found in the given document D.

Next, the part-of-speech tagging is done by using the NLTK-module’s POS tagging function. It happens by splitting the given text in to a list which is used as an input for the NTLK's "pos_tag"-function that attempts to tag the given words according to Penn Treebank tagset which is described in chapter 4.3. The function returns a list of pairs that consist of the word and the corresponding POS-tag. An example of how to use the part-of-speech tagging can be seen in figure 11.



```
In [5]: import nltk

In [6]: text_string = "i was riding my new car when suddenly a squirrel jump on
the road almost causing me to crash this made me upset"

In [7]: nltk.pos_tag(text_string.split())
Out[7]:
[('i', 'PRP'),
 ('was', 'VBD'),
 ('riding', 'VBG'),
 ('my', 'PRP$'),
 ('new', 'JJ'),
 ('car', 'NN'),
 ('when', 'WRB'),
 ('suddenly', 'RB'),
 ('a', 'DT'),
 ('squirrel', 'NN'),
 ('jump', 'NN'),
 ('on', 'IN'),
 ('the', 'DT'),
 ('road', 'NN'),
 ('almost', 'RB'),
 ('causing', 'VBG'),
 ('me', 'PRP'),
 ('to', 'TO'),
 ('crash', 'VB'),
 ('this', 'DT'),
 ('made', 'NN'),
 ('me', 'PRP'),
 ('upset', 'VBP')]

In [8]:
```

FIGURE 11. NLTK POS tagging

After the tagging is complete, all the bigrams are listed by using "bigram" function in the NLTK-module. Then all the found bigrams matching the described POS tagging rules are extracted from the tagged document. Bigrams are chosen according to the rules that define the word category and order in which they appear. Only the matching ones are extracted, third word following the bigram is also checked but is not extracted. Complete list of part-of-speech rules used in this work is seen in table 2. The listed rules are based on the list suggested by Turney (2002) with one additional rule "RB + VBP ^ *". The complete list of all Penn Treebank part-of-speech tags can be found in chapter 4.2**.**

TABLE 2. Part-of-Speech extraction rules

First Word	**Second Word**	**Third Word**
JJ	NN or NNS	any word
JJ	JJ	not NN nor NNS
RB or RBR or RBS	JJ	not NN nor NNS
NN or NNS	JJ	not NN nor NNS
RB or RBR or RBS	VB or VBD or VBN or VBG	any word
RB	*VBP*	*any word*

Calculating the positive and negative bigram feature vectors are performed by calculating the average semantic orientation $score_2$ for the set of positive bigrams P (see formula 9 in chapter 4.4, how to calculate SO for each bigram b), and $score_3$ for the set negative bigrams N. Both sets P and N are subsets of the extracted bigrams B found in the given document. All the bigrams $b \in B$ with $SO(b) > 0$ are considered positive and bigrams with $SO(b) < 0$ are considered negative. With the three scores, a document-level feature vector **x** can be presented as a triplet

$$\mathbf{x} = (score_1, score_2, score_3) \tag{11}$$

This feature vector is calculated for each document within the dataset and used to describe the document-level opinion features.

5.4 Training SOM and identifying the clusters

Two different Kohonen Self-Organizing Map implementations were experimented for this thesis. The first one is called “competing-layer” found in NeuroLab library (see chapter 2.2 for more details). Using this implementation resulted in fast training times and good clustering, however documentation did not provide good enough knowledge about the implementation and how everything was calculated so an alternative was also made.

Creating the custom implementation later called as *custom SOM* gave free hands to experiment with the learning algorithms and also gave an opportunity to understand better how self-organizing maps really work, because sometimes it is easier to learn from the hands on experiences. Full source code for the implementation in Python can be found from appendix 1.

The network training is first done by extracting the document-level feature vectors from documents in the used dataset. Then manually or arbitrarily chosen set of documents with calculated feature vectors are used for training the SOM. The network is then set to organize in to a given number of separate clusters that may be decided based on the knowledge about the dataset. A good rule of thumb is that each individual opinion category should be presented with at least one cluster. Therefore if it is known that the used dataset has documents from three categories *Positive, Neutral* and *Negative*, there should be at least three cluster units, presenting each of these categories. The training set should contain a good amount of samples from each category, which can be found by testing. Having too small amount of training samples might not provide enough information and having too many training samples can lead in *overfitting*. Overfitting may decrease networks capability to generalize outside the used training set and it is a common problem for all the learning models, also the time required to train the network increases with the amount of training samples.

After training the network, another challenge will arise; how to identify in which category each formed cluster belongs. The following method can be used to identify any number of formed clusters units. However if the presented feature vectors change, this solution is not guaranteed to apply anymore. The solution assumes, based on the knowledge about the chosen features, that the most *positive* cluster unit, or reference unit as denoted in chapter 3.2.3, $\mathbf{m}_i$ has the highest value from $score(\mathbf{m}_i)$, given by

$$score(\mathbf{m}_i) = \mathbf{m}_{i1} + (\mathbf{m}_{i2} - \mathbf{m}_{i3}) \tag{12}$$

The most *negative* cluster will have the lowest value from formula 14. By using the given function, knowing these two extremities, and doing simple reasoning, we can label the clusters according the calculated scores.

5.5 Testing Process

Testing the performance was done by using *k-fold cross-validation* method (see chapter 4.4) which happens by partitioning the used dataset in to *k*-subsets. Then one of the subsets is chosen for testing and the rest of subsets are used to train the system. This procedure is repeated for *k*-times. From each round the calculated scores are remembered and after all the subsets have once been used for testing, the average accuracy score is calculated by using the scores from all *k*-rounds. All the presented results were done with *5-fold cross-validation*, which means the used documents were split in five separate groups.

All the tests were run with the Kohonen Self-Organizing Map implementations found in NeuroLab library and with the custom implementation which was made along the process (see appendix 1). A comparison was made how well the system was able to classify according to document opinion groups and how well the decisions followed the provided ratings within the used datasets. For this the amount of neurons was adjusted to reflect the used ratings; for example if the training data has only two classes, SOM is trained to represents those two classes. In all the tests both SOM's used "Winner-Takes-All" training algorithm and the networks were set form during 2000 epochs on each round. For the testing two separate datasets were used and are explained in the coming chapter.

5.5.1 Used datasets

The first dataset for testing the system was the previously presented Amazon Review set originally collected by Jindal and Liu (2008). Training the classifiers was done by setting them to organize into four clusters, because the set was known to have negative reviews with ratings 1.0 and 2.0 and positive reviews with ratings 4.0 and 5.0. No neutral documents were included. Knowledge about the ratings was not used in the training process due the unsupervised approach; however the information was used in evaluating how well the system performed. Of course one could argue that since the same dataset was first used to construct the bigram lexicon, that makes it connected to the domain, and evaluating the results with the same dataset do not alone prove that the system can generalize to other domains.

These presented four clusters that were formed in the training process were dealt with in two different ways. First it was assumed that out of the four clusters, two of them presented the positive documents, one corresponding to ratings 5.0 - “Very Positive” and the other one to 4.0 - “Positive”. The two negative clusters then indicated the documents with ratings 1.0 - “Very Negative” and 2.0 - “Negative”. This way it was possible to evaluate how well the system was able to recognize two similar classes from each other. In the second test, classifiers were set to detect *Positive* and *Negative* polarities. The same amount of clusters were used. This time the opinion strengths between the clusters within the same sentiment polarity were not evaluated. Results were just measured by how well the system was able classify polarities.

Amazon Reviews set was extracted into individual files according to ratings where reviews with rating > 3.0 were added in positives, and rating < 3.0 added in negatives. This structure made it is easier to handle the data. Only the ratings and text content were stored. The output files used a simple XML-scheme, which helped processing the files later on. To calculate the document-level feature vectors text content was extracted from the files, but no information about the ratings was exposed to the system. A Mixed set of 2000 documents containing 1000 *Positive* and 1000 *Negative* reviews was first created from six different domains. The documents were split and distributed pseudo-randomly to 5 different groups for 5-fold testing. In other words, each group had a varying ratio of positive and negative documents.

The second included dataset was chosen from IMDB movie review corpus originally collected by Pang and Lee (2005). The set is called *scale_dataset v1.0* and it contains a large amount of movie reviews labeled according to scaled ratings. This set comes with labels for three and four class schemes that have been derived from normalized numerical ratings that also were included.

For the first test, a set consisting of 900 documents with three class label ratings (*Positive*: 312, *Neutral*: 349, *Negative*: 239) was made. These documents were randomly split and distributed in to five groups for 5-fold cross-validation like the Amazon review set. For simplicity, documents were first extracted to separate files that used the same XML-scheme as the Amazon reviews. This set was tested with the three class label scheme to see how well system differentiates between these three classes. Both

SOM classifiers were set first to form 3 clusters that would represent the 3 classes, otherwise all the settings were kept same as with the Amazon reviews.

The second test with the same IMDB document set was run after removing all the neutral documents. That left 550 documents after all the neutrals and one positive were removed from the set, to make all the groups even. SOM classifiers were set to form two clusters to represent each polarity.

5.5.2 Results

Reading the result tables is done in following manner: The first column *Classifier* indicates the used classifier, which in this case is either *NeuroLab* or *Custom* self-organizing map implementation. The second column *k-fold* tells in how many subsets the dataset was split and from how many test rounds the average scores were with-drawn. The third column *Group* shows which opinion groups were detected; *Pos/Neg* indicates *Positive - Negative* opinion groups and *VP/Pos/Neg/VN* represents *Very Pos-itive*, *Positive, Negative* and *Very Negative* opinion groups. In the table 4 *Pos/Neu/Neg* represents the three categories *Positive, Neutral* and *Negative*. *Documents* column shows the total number of documents in the used dataset. *Min-%*, *Max-%* and *AVG-%* are the minimum, maximum and average accuracies calculated from the test rounds. AVG-% column is the most important and it shows the average accuracy, which may be a better performance estimator than a percentage from individual round.

First, the testing was done by using the Amazon dataset. A positive result straight from the beginning was the overall difference in the two SOM's performance which was minimal. It suggested that both implementations worked correctly and there were no errors that could have affected the results. As seen in table 3, the custom imple-mentation had a slightly better Max-% and AVG-% compared to NeuroLab when just opinion polarity was used.

TABLE 3. Amazon test results (2 and 4 classes)

Classifier	*K-fold*	*Group*	*Documents*	*Min - %*	*Max - %*	*AVG - %*
NeuroLab	5-fold	Pos/Neg	2000	84.00	86.75	84.60
Custom	5-fold	Pos/Neg	2000	84.00	**87.00**	**85.40**
NeuroLab	5-fold	VP/Pos/Neg/VN	2000	**39.75**	**45.00**	**41.85**
Custom	5-fold	VP/Pos/Neg/VN	2000	38.75	43.25	41.65

The results from detecting opinion alignment between 4 classes were much worse for both classifiers. Here NeuroLab had a little bit higher accuracies from all the measurements, that are most likely to flatten out if the tests were run multiple times and the averages taken. They both scored just above 41 % on average and NeuroLab peaked 45%. This clearly indicates that when measuring document-level opinions, features that were chosen may not be enough in detecting differences between similar opinion categories.

To confirm that the system was not affected by the bigram lexicon, generated from the same dataset which was now used for testing, another testing set was included in the process. First a set consisting of 900 reviews collected from IMDB (see previous chapter for more details) and labeled in three classes; positive, neutral and negative was used to show how well the system can differentiate between three distinct classes. Then on the second round, all the documents with the neutral label were removed leaving 551 documents. One of the positive documents was then also removed to even all the 5 groups.

As can be seen from the results in table 4, using three classes lead to quite low performance, which may be reflecting the fact that the used techniques have been developed for detecting opinion polarities that often are quite distinct cases. Both SOM's performed equally, following the trend which was already seen in the previous tests.

TABLE 4. IMDB test results (2 and 3 classes)

Classifier	*K-fold*	*Group*	*Documents*	*Min - %*	*Max - %*	*AVG - %*
NeuroLab	5-fold	Pos/Neu/Neg	900	**41.12**	**50.00**	**44.12**
Custom	5-fold	Pos/Neu/Neg	900	38.89	48.33	42.89
NeuroLab	5-fold	Pos/Neg	550	60.00	**78.18**	68.36
Custom	5-fold	Pos/Neg	550	60.00	77.27	**68.54**

Using just two sentiment classes clearly affected how well the system performed. Average results went up ~24%, which lead to accuracy better than what is achieved by arbitrarily guessing long enough. However the system did not reach similar accuracies than with the Amazon reviews, but it is clearly capable of making a distinction between positive and negative document-level opinions.

The overall performance that the system demonstrated was satisfying. Calculated average accuracies created a clear view that the used features are good for detecting document-level opinion polarity. However some additional feature(s) may be required to classify similar documents like, positive and very positive. Some additional tests would be needed to be sure how the performance is affected if the documents were withdrawn from totally different domain. Adjusting and experimenting with the SOM classifiers, by running the test with different amount of clusters may also help to make more accurate choices in border cases. The generalization that happened in the training process was quite strict. Therefore adding more clusters may help in recognizing nuances and would also use the available strengths that are found in using the self-organizing maps.

5.6 Empirical Work

Designing and implementing the described system may be considered its own sub-problem along the whole opinion mining. It may be implemented in various ways and one method is suggested. It may not be the best and there are many things like, security that needs to be carefully considered when creating a software system. The design process started, by thinking of the different tasks that are needed to be performed in order to fulfill the given requirements. Since the only requirement, was to design a system that can classify documents (preferably using Python), almost complete freedom was given to approach this problem.

At this point, it has to be noted that the way the system is implemented has barely any part in how accurately the document-level opinion-mining is performed. The classification accuracy is related to the quality of chosen features including extracted bigram lexicon and available datasets. It is also related to how well the classifier is able to generalize what it has learned, how the clusters that are formed in unsupervised manner are classified, and the used part-of-speech tagger may affect the accuracy.

All the main *modules* or *components* used in the design can be seen in figure 12. *Document Handler*, later referred as *DH*, is the main component that connects all the parts together. It receives unlabeled document as input and passes it to the *Feature Vector Extractor* (*FVE*) which extracts all the document-level feature information. FVE sends the extracted *feature vector* back to DH that passes it next to the *SOM Wrapper*. SOM Wrapper acts as an interface between NeuroLab / Custom SOM and the Document Handler, providing unified functions that are required to use the classifiers. SOM Wrapper sends the received feature vector to the used classifier which responds by returning the opinion orientation matching closest to the given feature vector. The received classification is then forwarded back to document handler that passes it back to caller.

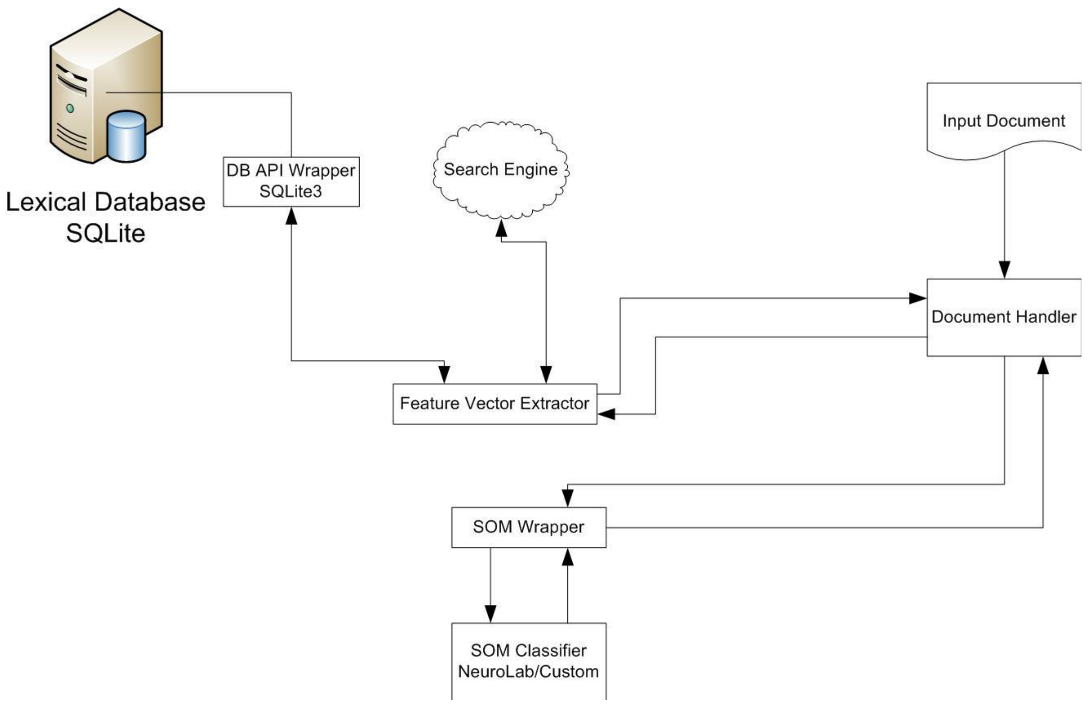


FIGURE 12. Sketch for component structure

To provide a simple access within any Python application, the system may be constructed in a simple client - server model, see figure 13 for a simplified diagram how this model work. The server functions as the “Sentiment Analysis Service”, which receives unlabeled documents as input from the connected client. Received document is then passed through the document handler to the FVE. The process continues as previously described. When the document handler receives the classification, it passes it to the server that responds to the connected client with the opinion orientation, which passes it back to the application that requested classification.

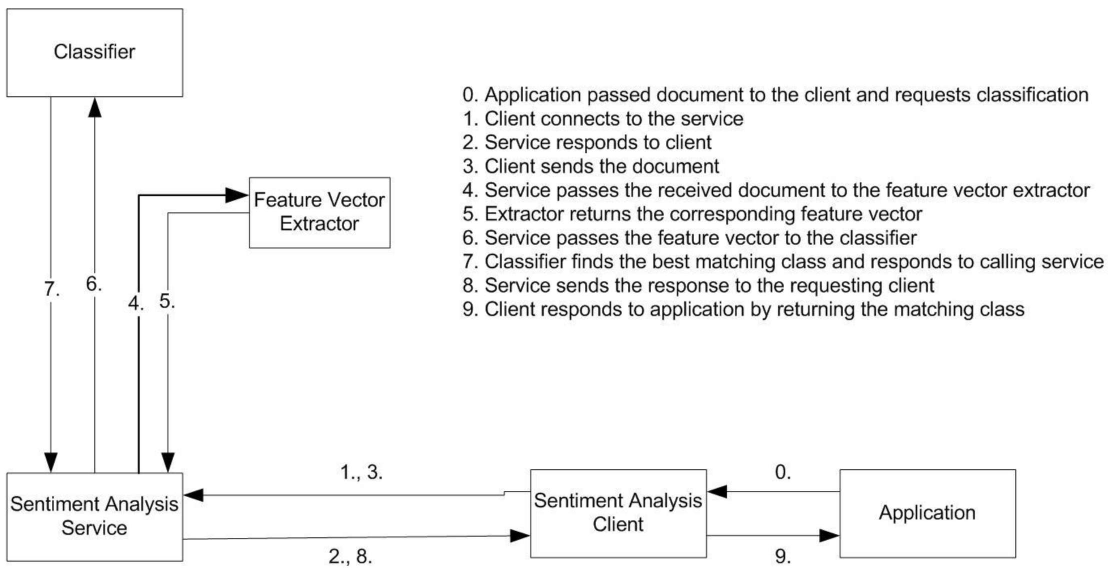

Classifier
Feature Vector Extractor
Sentiment Analysis Service
Sentiment Analysis Client
Application
7.
6.
4.
5.
1., 3.
2., 8.
0.
9.
0. Application passed document to the client and requests classification
1. Client connects to the service
2. Service responds to client
3. Client sends the document
4. Service passes the received document to the feature vector extractor
5. Extractor returns the corresponding feature vector
6. Service passes the feature vector to the classifier
7. Classifier finds the best matching class and responds to calling service
8. Service sends the response to the requesting client
9. Client responds to application by returning the matching class

FIGURE 13. Client - Server model

All the inter-process communication between the client and server can be done locally, for example, by using *socket* module which is a *low-level networking interface* found in Python. More about socket can be read from Python 2.7.3 Online Documentation (2012).

This implementation would require that the classifier is either re-trained every time it is restarted or that it has the capability to "save" and "load" all the necessary information about the network which depends on the used SOM implementation. Thus it was also one of the reasons why client - server model was chosen. This way the network may stay "online" as long as the hosting machine is on and just be restarted to re-train the network when it is needed, for example, if the used training dataset is updated.

5.7 Issues

There were some issues that arose in the thesis process. The first one was the realization how hard it is to collect a corpus of opinionated documents that could cover all the general domains. This clearly sets some boundaries for testing and generating lexicon, however, thanks to the datasets collected by Hu and Liu (2004), Jindal and Liu

(2008) and Pang and Lee (2005), this problem was not so crucial. In the beginning, It was not realized how big a task it would be. Opinion mining itself is a higher level process that is composed from many underlying techniques, which help making it possible. Therefore being able to use great modules like NeuroLab and Natural Language Processing Toolkit, clearly demonstrated that advancements in natural language processing and machine learning have happened. Building the introduced system from the scratch may be enormous task, which might be the case in some languages where the existing implementations are not available.

Another thing that showed up is that the covered fields are sometimes lacking a common language. This can be seen with the name sentiment analysis which can be also called as opinion mining or subjectivity analysis depending on author's opinion. Bing Liu (2011) has tried to standardize the terms he uses, which really helps. The same issue came up when dealing with neural networks, the information about them is covered well, however there is small variation in terminology between the authors that were mentioned in this thesis, for example, artificial neuron can also be called as a node or unit. Of course, some of the sources go back over 10 years so these things have hopefully improved. Overall this also shows the fact that understanding the underlying techniques used in data mining, machine learning and natural language processing is important. It builds the ground where tasks like sentiment analysis can be approached even though it may have evolved to its own field of study.

The most difficult task in using unsupervised learning and Kohonen Self-Organizing Maps might be the process of connecting the formed clusters under opinion labels. In this work it was relatively easy because of the assumptions that were made. However, this may not be the case if some other features are used, as also can be seen in the results, part of the failed classifications may come from incorrectly identified clusters. One possible solution could be exploiting human understanding, for example, by graphically representing the clusters and labeling them manually after carefully examining the used training set. Another way would be by studying the used features and the results to see possible characteristics that may exist in documents with different opinions then engineering the system to differentiate results according to these rules. This makes fine tuning the system a crucial process.

5.8 Extending to other natural languages

Extending the work to other natural languages might be possible but have not been tested. In order to do that, at least the following things are required. First, the opinionated word corpus would have to be generated in the target language. This could be done using machine translation from English to the target language, how well it may work might totally depend on the language. There can be unique words and expressions that have opinionated meanings and those may not be included, if it is generated only from the existing corpus.

Generating the bigram database in other languages might be a difficult task. First a part-of-speech tagger that works for the specific target language is required. This may be solved by using existing POS taggers, or in the worst case it has to be developed. The used part-of-speech rules might not apply, so they have to be engineered to match the grammar and structure. Using *PMI-IR* algorithm to calculate semantic orientation can be possible, especially with the most common languages. The lack of results may cause a problem with less used ones. Here the previously introduced local search engine approach may be used, if there is access to a documents corpus written in the target language. If these previously mentioned problems are solved, it may be possible to apply this document-level opinion mining approach in other languages. Clearly expert knowledge is a crucial aspect, because, languages can have unique features that may get overlooked if only the existing knowledge about another language is applied.

Another easier solution may be using machine translation to translate the target documents into language which is modeled in the system. Using the approach might work, but there is always a risk. Important contextual information may be lost which cannot be exactly translated in to the target language. If the translations were done manually by a person, this approach might be more reliable. This method was quickly tested with a few reviews written in Finnish language and translated in English using Google Translator. The results were promising; however deeper and more thorough testing would be required to confirm these findings.

5.9 Further work

There are multiple ways of how the described document-level opinion mining system could be improved. Some of these ideas for further development are presented. Firstly, how the calculated semantic orientations are normalized is important, because it greatly affects how well the SOM classifier performs. SOM uses Euclidean distance to calculate the distances between given input vectors and reference vectors, and it is completely blind to the information that might be in within the vectors. That is why all the choices and balancing between feature vectors has to be done before feeding them to a classifier. This can be important in order to find all the opinion categories correctly.

Handling the document-level neutrality has to be carefully thought out due to the neutrality that can be seen in more than one way. The true neutrality, when there are no opinions expressed, is probably the most common idea one has when thinking about neutral opinions. Another type of neutrality comes from differing opinions that are in balance. This neutrality can also make it difficult to find the relevant document-level opinion if it is used incorrectly.

The individual opinion categories were represented in this work with only one cluster unit. Differences between similar opinion categories may be detected by doing the classification, using multiple cluster units to form clusters that represent individual opinion categories. An example of this is how the shallow differences between very positive and positive categories can be found more accurately. This approach would utilize the strengths found in using SOM's. The formed clusters could then be labeled in a supervised manner by examining the differences within formed clusters and the used training data.

Adding the handling for negation, comparison and connection words may also be used to increase the accuracy. By detecting negation words like *no, not, never* etc., together with opinion term, the term opinion polarity can be switched. Applying word stemming may also improve the system, but was not done in this thesis. With word stemming the lexicon size could be kept smaller. Detecting accuracy may also improve due to many of the grammatical differences that exist, for example, "fail, fails, failed, failure", could be recognized by the root word "fail", and only the information about a single term needs be included in the lexicon.

6 CONCLUSION

The subject for this thesis came from Cluetail Ltd. The company needed a research to be done about sentiment analysis; more precisely, about the possibilities in document-level opinion-mining and how such system could be implemented in Python. Studying the subject led to an understanding about the problem and the available solutions. It also helped to understand how sentiment analysis is a high-level application, coming from an actively studied research field that incorporates techniques from data mining, machine learning and natural language processing. Finding the document-level opinion is often done using discovered linguistic features. This can be achieved by using existing or manually assembled corpora, which in the latter case can be a time consuming process. The suggested solution comes from combining supervised and unsupervised learning methods. Compared to its counterpart, unsupervised learning may not have as many learning techniques available; however, using it can lead to good results with less work. That can be a key factor if the research resources are limited. The opinion mining process consists of two key elements. The first one is choosing the set of features that represent the opinion information. The second one is the document classification based on the information discovered from the chosen features. Classification can be done by using machine learning methods, for example, Bayes networks, Neural Networks or Support Vector Machines. These last two are explained in chapter 3.

Two different features were chosen to represent the document-level opinion information; a term frequency and a bigram based feature. Positive/negative term frequency measures the amount of positive and negative terms found within a document (see chapter 5.1). Bigram feature is formed by using three separate methods. It starts by tagging the processed document with part-of-speech tags (see chapter 4.2). This is followed by extracting the part-of-speech tagged words as bigrams (see chapter 4.1). The bigrams are checked against a set of rules, suggested by Turney (2002) (see table 2 in chapter 5.3 for the list of used rules) and the matching bigrams are extracted for further processing. Unsupervised learning algorithm PMI-IR (see chapter 4.3) is used, and a probability based opinion orientation score is calculated for each individual bi-gram.

Kohonen Self-Organizing Map (SOM) was chosen as the used classifier. It was chosen because of its ability to form clusters from untagged datasets that are used in the training process. Compared to supervised neural networks, training unsupervised neural networks does not require labeled documents with known opinion categories; this can save a great deal of work in the process. The formed clusters are detected and an algorithm is used to label the cluster units under the given opinion categories, for example, positive and negative. The amount of clusters depends on how many categories are known, or assumed to be within the training set. Two different SOM implementations were used to do the classification, and the results were compared with each other. NeuroLab library's "competing layer" was chosen as one of the used classifiers. The second one (referred in this thesis, as *custom SOM*) was built to learn and to have free control over the network and how it works. The source code for this implementation can be found in appendix 1.

Testing the suggested system was performed by using two separate datasets with multiple setups. The first test set was assembled from a corpus of Amazon reviews, composed by Jindal and Liu (2008). The corpus was also used to generate the initial lexicon of the most commonly appearing bigrams that were used to detect bigram features. Second test set was a collection of IMDB reviews collected by Pang and Lee (2005). All the tests were run using 5-fold cross-validation method, which happens by splitting the dataset in 5 subsets and then using each individual subset once for testing and the other subsets for training. This method leads to five individual rounds where the average accuracy is calculated. All the test results can be seen in in chapter 5.5.2.

These results gave a clear view that the system is capable of detecting document-level opinion polarities (positive - negative) with over ~68 percent accuracies. When neutral sentiment (positive - neutral - negative) and/or more fine-grained sentiment categories (very positive - positive - neutral - negative - very negative) were introduced, the average accuracy decreases below 50 percent. This accuracy is far from enough to do reliable classification. It must be remembered that the results came from highly opinionated reviews; therefore the accuracy may be even lower if the documents have more subtly expressed opinions. The design introduced in chapter 5.6 was locally implemented to test the idea. It proved to be a working solution. The client - server model can also be used to provide the means to implement the system in other programming languages that support socket networking. These results show that a lot of good progress has been done and a foundation has been laid. This presented suggestion is our first step towards sentiment analysis in document-level and a lot of further development is still needed. More about possible improvements can be read in chapter 5.9.

The research for this thesis inspired me to think about the document-level opinion mining approach. There are certain problems (introduced in the earlier chapters), that makes document-level opinion mining difficult and in some cases the results irrelevant. This may happen when there are more than one opinion holders and/or more than one subject within the opinion mined document. Calculating the overall opinion from documents like this may not provide useful information. Reviews and other precisely opinionated documents are in my opinion the best subjects for document-level opinion mining. To detect opinions from blogs, news articles etc. may require more developed approaches. This approach could be a hybrid-model, based in aspect-based opinion mining and opinion holder detection (both introduced in chapter 4). Using these two approaches together; by finding the individual opinion targets and expressed opinions connected to the opinion holders could provide detailed information about the documents. This solution may provide more valuable and exact information about the documents. It would also give a possibility to compare the found results.

The idea of finding a document-level opinion is difficult because how individuals perceive the information is subjective. This could be solved in various ways. Specifically engineering the system for each domain, because as stated in the chapter 4, similar expressions may have different meanings depending on the domain they are used. This may involve more work to acquire needed knowledge about the covered domains; however it can lead to better results. Another idea could be taking in the consideration the opinions that the target users (to whom the results are targeted) may have, later referred to as *opinion hunter*. Based on my personal observations about the studies regarding opinion mining, they are often focused on looking for the opinions that the opinion holders are expressing. However, the opinion of the opinion hunter is left out. I believe that the opinions that the opinion hunter holds are important. Using this factor could provide more accurate results. For this I suggest an idea about *personalized opinion mining*. This may help to adapt the opinion mining system to provide more personalized results, endowing it with the capability to see the opinions through the users own eyes.

Overall, a lot of important knowledge about sentiment analysis was gained. The possibilities and problems in opinion mining are now understood much more clearly. It provided more perspective, compared to where everything started. It also gave encouragement to learn more about the techniques used in data mining, machine learning and natural language processing. The mentioned fields alone do not provide a solution for sentiment analysis, which is its own field; however, knowing them gives a foundation of where to begin. These findings also led to thinking more clearly of how this task should be done. The test results were not as good as expected originally, but they gave a lot of good information. I believe that with further development, the suggested system can be made more accurate. Considering all the ways it can be used more thoroughly helps to find tasks where the suggested system can bring further benefits. For me personally, this whole process gave valuable experience and opened up a starting point to learn more about sentiment analysis. Clearly, there are more things left uncovered than what was presented. This work was mainly focused on learning about unsupervised document-level opinion mining. It used a few chosen techniques, and it is just one possible approach among many others. I hope that this thesis could serve as a basic overview about the things that are needed to be understood in order to do opinion mining, and can be useful for anyone who has an interest in sentiment analysis and might want to learn more.

BIBLIOGRAPHY

Bird, Steven, Klein, Ewan, Loper, Edward 2009. Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit. United States of America: O'Reilly Media.

Fausett, Laurene 1994. Fundamentals of neural networks: architectures, algorithms, and applications. United States of America: Prentice-Hall Inc.

Graupe, Daniel 2007. Advanced Series on Circuits and Systems – Vol. 6: Principles of artificial neural networks 2nd edition. Singapore: World Scientific Publishing Co. Pte. Ltd.

Hu, Mingqing Liu, Bing 2004. Mining and Summarizing Customer Reviews. Published in Proceedings of the ACM SIGKDD International Conference on knowledge Discovery and Data Mining (KDD-2004). New York: ACM.

Kohonen, Teuvo 1997. Self-Organizing Maps 2nd Edition. Berlin: Springer-Verlag.

Liu, Bing 2011. Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data 2nd Edition. Berlin: Springer-Verlag.

Neurolab Online Manual 2012. Online-manual. http://packages.Python.org/neurolab/. Last updated 2011. Referred 18.4.2012.

Pang, Bo, Lee, Lillian, Vaithyanathan, Shivakumar 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. Published in Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10. Philadelphia: ACL, 79-86.

Pang, Bo & Lee, Lillian 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. Published in ACL '05 Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Philadelphia: ACL.

Prawobo, Rudy & Thelwall, Mike 2009. Sentiment Analysis: A combined Approach. Published in Journal of Informetrics 3 (2009) 143 - 157. United States of America: Elsevier Ltd.

Python 2.7.3 Online Documentation: SQLite3 – DB-API 2.0 interface for SQLite databases. Online Documentation. http://docs.Python.org/library/sqlite3.html. Last updated 18.4.2012. Referred 18.4.2012.

Python 2.7.3 Online Documentation: socket – Low-level networking interface. Online Documentation. http://docs.Python.org/library/socket.html. Last updated 18.4.2012. Referred 19.4.2012.

Russell, Stuart & Norvig, Peter 2010. Artificial Intelligence, A Modern Approach Third Edition. United States of America: Pearson Education Inc.

Santorini, Beatrice 1990. Part-of-Speech Tagging Guidelines for the Penn Treebank Project. University of Pennsylvania, Department of Computer and Information Science. Technical Report.

Turney, Peter D 2001. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. Published in Proceedings of the Twelfth European Conference on Machine Learning. Berlin: Springer-Verlag, 491 – 502.

Turney, Peter D 2002. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. Published in Proceedings of the 40th Annual meeting of the Association for Computational Linguistics (ACL). Philadelphia: ACL,417-424.

Witten, Ian H., Frank, Eibe, Hall, Mark A. 2011. Data Mining: Practical Machine Learning Tools and Techniques 3rd Edition. United States of America: Morgan Kaufmann Publishers.

APPENDIX 1 Kohonen self-organizing map implementation in Python

```
# -*- coding: utf-8 -*-
#Kohonen Self-Organizing Map
#Required modules: Numpy (Tested with ver. 1.6.1)

from numpy import random, zeros, sqrt, arange, square, sum, log, exp, array


class KohonenNetwork:
        """Kohonen Self-Organizing Map"""

        def __init__(self, network_layers=1, network_output=4, network_input=3):
                #The amount of layers in the network
                self.layer_in_net = network_layers
                #The amount of output units in the network
                self.output_in_net = network_output
                #The number of input vectors
                self.input_in_net = network_input
                #Initialize network weights
                self.weights = random.uniform(0, 1,
                                                (self.layer_in_net,
                                                self.output_in_net,
                                                self.input_in_net))
                #Distances between input x and network clusters
                self.Y = sum(self.weights, 2)
                #Current learning time step
                self.timestep = 0.0
                #Learning rate in the beginning
                self.learning_rate_original = 0.9
                self.learning_rate = self.learning_rate_original
                #The overall timesteps
                self.epochs = 1
                #Matrix used update winner unit
                self.help_maxtrix = zeros((self.layer_in_net,
                                                                self.output_in_net,
                                                                self.input_in_net))
                #Used to store minimum unit i,j index values
                self.min_i = 0
                self.min_j = 0
                self.minimum = 10000000000000000000
                self.neighborhood_ratio_orig = self.output_in_net / 2
                self.error_delta = 0.0
                self.neighborhood = self.neighborhood_ratio_orig

        def __calculate_distances(self, input_vector):
                """calculate distances between network units & given input """

                self.Y = sqrt(sum(square(input_vector - self.weights), 2))

        def __update_network(self, input_vector):
                """update learning rate and calculate new weights"""
                self.__update_learning_rate()
                self.weights = self.weights + self.learning_rate *\
                                                self.help_maxtrix * (input_vector - self.weights)

        def __update_learning_rate(self):
                """update the learning rate

                degrates lower over time.

                """
                #self.learning_rate = self.learning_rate_original+
                #(0.01-self.learning_rate_original) * (self.timestep / self.epochs)
                #Kohonen - > 1997 p. 88
                self.learning_rate = self.learning_rate_original *\
                                                        (1.0 - self.timestep / self.epochs)

        def __update_neighborhood_ratio(self):
                """update neighborhood ratio

                gets lower over time.

                """

                #Gradient descending
                sigma = max(self.input_in_net, self.output_in_net) / 2.0
                lbda = self.epochs / log(sigma)
                self.neighborhood = sigma * exp(-self.timestep / lbda)

                #Linear descending alternative
                #self.neighborhood = self.neighborhood_ratio_orig +\
                #(1.0 - self.neighborhood_ratio_orig) * (self.timestep / self.epochs)

        def __find_winner_unit(self):
                """finds the winner unit

                stores the index to min_i, min_j

                """
                self.minimum = 9999
                self.min_i = 0
                self.min_j = 0
                for i in arange(self.layer_in_net):
                        for j in arange(self.output_in_net):
                                temp_sum = self.Y[i, j]
                                if temp_sum < self.minimum:
                                        self.min_i = i
                                        self.min_j = j
                                        self.minimum = temp_sum

        def __choose_updated_units(self):
                """choose updated neighboring units

                uses a hard limit
                indicates which units get update

                """

                #Hard limit, only the matching unit gets update
                self.__update_neighborhood_ratio()
                for i in range(self.layer_in_net):
                        for j in range(self.output_in_net):
                                distance = sqrt(((i - self.min_i) ** 2) +
                                                        ((j - self.min_j) ** 2))
                                if distance > self.neighborhood:
                                        self.help_maxtrix[i, j, :] = 0
                                else:
                                        self.help_maxtrix[i, j, :] = 1

        def __choose_updated_units_gauss(self):
                """choose updated neighboring units

                indicates which units get update
                uses gauss method, creating a soft limit

                """
                self.__update_neighborhood_ratio()
                for i in range(self.layer_in_net):
                    for j in range(self.output_in_net):
                        distance = sqrt(((i - self.min_i) ** 2) +\
                                        ((j - self.min_j) ** 2))

                        dist = exp((sum(- distance)) / self.neighborhood)
                        if distance > 0:
                            self.help_maxtrix[i, j, :] = square(dist * 1)
                        else:
                            self.help_maxtrix[i, j, :] = dist * 1

            def train_network(self, training_data, epochs=1500):
                """train the network

                takes training dataset and the amount of training steps as input

                """
                self.epochs = epochs
                td_samples = training_data.shape[0]
                for i in range(epochs):
                    self.error_delta = 0.0
                    self.timestep += 1.0
                    random.shuffle(training_data)
                    for j in range(td_samples):
                        self.__calculate_distances(training_data[j])
                        self.__find_winner_unit()
                        error = sum(abs(self.weights[self.min_i,
                                                     self.min_j] - training_data[j]))
                        self.error_delta += error
                        #Uncomment to enable hard threshold
                        #self.__choose_updated_units()
                        #Comment to disable gauss
                        self.__choose_updated_units_gauss()
                        self.__update_network(training_data[j])
                    if (self.timestep % 5) == 0:
                        print "Error on epoch {0} -> {1}".format(self.timestep,\
                                            self.error_delta / float(td_samples))
                print self.weights

            def __return_winner(self):
                """indicates the winner unit

                returns an output array

                """
                for i in range(self.layer_in_net):
                    for j in range(self.output_in_net):
                        self.help_maxtrix[i, j, :] = 0
                self.help_maxtrix[self.min_i, self.min_j, :] = 1
                return sum(self.help_maxtrix, 2) / self.input_in_net

            def lookup(self, input_vector):
                """finds the best matching unit

                takes testing vector as input

                """
                self.__calculate_distances(input_vector)
                self.__find_winner_unit()
```

```
		#Returns the location of winner unit
		return self.__return_winner()

if __name__ == "__main__":
	#Create new network with 1 layer, 2 outputs and 2 inputs
	network = KohonenNetwork(1, 2, 2)
	#Create 4 training samples - uses numpy.array
	training_data = array(((1.0, 1.0), (0.9, 0.9), (0.0, 0.0), (0.2, 0.2)))
	#Train the network with training samples: training_data
	network.train_network(training_data, 500)
	#Test network, prints the location of winner unit
	print network.lookup(array((0.2, 0.2)))
	print network.lookup(array((0.7, 0.7)))
```