UNIVERSITETET I AGDER

Classification with Multiple Classes using Naïve Bayes and Text Generation with a Small Data Set using a Recurrent Neural Network

TORE ELIAS GJERVIK REITEN

SUPERVISOR
Morten Goodwin

University of Agder, 2017
Faculty of Science and Technology
Department of ICT

University of Agder

Master's Thesis

Classification with Multiple Classes using Naïve Bayes and Text Generation with a Small Data Set using a Recurrent Neural Network

Author:
Tore Elias Reiten

Supervisor:
Morten Goodwin

A thesis submitted in fulfilment of the requirements for the degree of Masters in ICT

in the

Department of ICT

May 2017

Acknowledgements

Completing a master's degree in ICT has been a challenge. With my background in electronics, the transition to pure software and computer science with specialization in artificial intelligence has not been an easy task. It has, however, been an educational adventure.

I would like to use this opportunity to express my gratitude to my supervisor Associate Professor Morten Goodwin for assisting me with technical details as well as the formalities required for writing a good thesis.

I also thank my fellow students who have been great discussion-partners through the period of my masters degree.

Abstract

In this thesis, text classification and text generation are explored using only a small data set and many classes. This thesis experiments with text classification, and show how it is able to find the most similar output compared to the input even with thousands of classes. Furthermore, text generation is explored on a small data set to create a unique output. By using Naïve Bayes text classifier combined with a Recurrent Neural Network language-model, it is possible to use new deviations as input before an original suggestion for a measure is generated as the output.

Contents

List of Figures

List of Tables

1 Introduction

Text classification and text generation are familiar topics for research, in the fields of machine learning. Machine learning requires training- and validation data for the computer to be able to learn. For the testing to give good results, a fair amount of data is needed. It is, therefore, interesting to see how machine learning can be used on relatively small data sets and to see if the output can still be reliable.

Naïve Bayes (NB) is known as a very famous and powerful classification technique, and is commonly used as a text classifier [1]. The *20 newsgroups* data set, consisting of 20000 documents divided over 20 classes [2], is very popular for testing classification techniques. However, using many classes (thousands) and short texts for the sake of text classification is not as commonly explored, and is tested and validated in this thesis. Recurrent Neural Networks (RNNs) can create its outputs based on the input given. It can remember sequences and generate, based on these sequences, similar ones without being copies. This thesis explores the use of RNN on a small data set, and we will see that without enough data, it is difficult for the RNN to learn and create its own output.

As the small data set deviation forms will be used. In short, a deviation form is a form which needs to be filled out when something unusual (a deviation) occurs in the workplace. Later on, a person - typically the HSE-responsible (Health, Safety and Environment) at the company - takes a look at the deviation and writes a suggestion for a measure, so the deviation does not happen again.

The goal of this thesis is to demonstrate the use of text classification with text generation, combined in a novel way. This thesis examines the use of small data sets for classification and generation, specifically.

1.1 Problem Statement

Usually when using machine learning; hundreds or thousands of megabytes are considered as a fair amount of training data. When the amount of data is too small, the algorithm might not have enough data to be able to learn properly - which will lead to the results being bad, or even useless.

This thesis will research the use of Multinomial Naïve Bayes (MNB) on a small data set with several classes. In addition to MNB, research will be done on a Recurrent Neural Network (RNN) to generate new text based on the best suggestions for a measure to create an even more appropriate advice.

1.1.1 Research Questions

1. Is it possible to use Naïve Bayes to find a deviation similar to the input?
2. Can the RNN combine the best suggestions for a measure to create a consistent output with the given data?

1.1.2 Hypotheses

1. Naïve Bayes can find the most similar deviations even with a small dataset and many classes, where the output will resemble the graphical representation as shown in Figure 1.1.

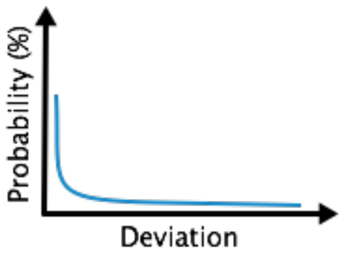

FIGURE 1.1: A graphical representation of the expected output when using NB.

2. MNB can find similar deviations, but the RNN might need more data to create a consistent output.

1.2 Contributions

This thesis covers the use of MNB, with several classes, for text classification, as well as covering the use of an RNN for text generation on a relatively small data set.

1.3 Thesis Outline

Further, the report will cover theoretical background, in Chapter 2 for better apprehending the entity of the implementation and results, which are shown in Chapter 4 and 5, respectively. In between the theory and implementation, Chapter 3 will cover State-of-the-Art, which consists of other people's work. Finally there will be a discussion in Chapter 6 before Conclusion and Future work is represented in Chapter 7.

2 Theoretical Background

This chapter describes some theory used in this thesis to grasp the entity of it better. The main subjects are text classification using Naïve Bayes (NB), and text generation using an RNN.

2.1 Classification

Classification is a systematic arrangement in groups or categories according to a certain established criteria [3]. In computer science it is used to categorize any set of data. The data can be of any form i.e.; text, image and sound. A common method to implement this is to convert the data to points in a space with n-number of dimensions which represents a specific characteristic of the sample.

Figure 2.1 shows a simple example of how classification works. Based on some parameters, which in this case are humidity and temperature, points are assigned within the dimensions of the diagram. Each point represents a day, where the blue points are days with sunny weather and the red points are days with rainy weather. There are two parts for the classification - the training phase and the classification phase. In the training phase, a set of samples is fed to the algorithm. These samples are called training data. Based on the said training data, the algorithm constructs a model which

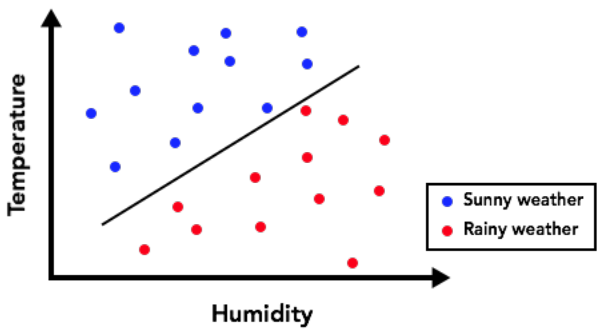

FIGURE 2.1: A simple example, showing how linear classification can be used for weather prediction.

in this case is a line. When the classification phase starts, the model can predict the class of samples of which the class is unknown. E.g. for Figure 2.1 if the new samples are measured to be above the line, it is classified as sunny, and vice versa below the line.

2.1.1 Text Classification

The aim of text classification is to assign a given text into one or more classes in a predefined set of classes [4]. The given text could be a document, email, tweet, news post etc. Text classification includes categorizing all given text into topics. The user decides how many topics the text could be classified into.

Popular learning algorithms for text classification are decision tree learning, support vector machine, regression methods (e.g. Naïve Bayes), neural network and rule learning methods [5].

2.2 Naïve Bayes

NB is an algorithm which uses Bayes' Theorem. Bayes' Theorem is a formula that calculates a probability by counting the frequency of given values or combination of values in a data set [6]. If A represents the prior events, and B represents the dependent event then Bayes' Theorem can be stated as in equation 2.1 [7].

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.1}$$

For further explanation of Bayes' Theorem, Figure 2.2 illustrates a Venn diagram where the pink color represents those who have a deadly disease while the purple represents those who tests positive for a deadly disease.

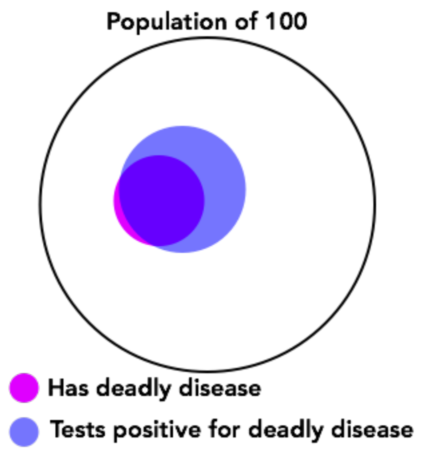

FIGURE 2.2: A venn diagram of people who have a deadly disease and people who test positive for a deadly disease.

Out of 100 people, 5 have a deadly disease. If the test is 85% accurate, the intuitive answer is that one person has 85% chance of getting the correct diagnose, i.e. has a deadly disease, or has not a deadly disease. From the diagram in Figure 2.2 it is clear to see that this is not the fact. By using Bayes' Theorem (equation 2.1) where the number of people who has a deadly disease is $P(A) = 0.05$, and the number of people with a deadly disease who actually tests positive is $P(B \mid A) = 0.85$. Further, let 6% of the population test positive even without having a deadly disease, then $P(B)$ will be as stated in equation 2.2.

$$\begin{aligned} P(B) &= 0.85P(A) + 0.06(1 - P(A)) \\ P(B) &= 0.0425 + 0.057 \\ P(B) &= 0.0995 \end{aligned} \tag{2.2}$$

Bayes' Theorem will then be indicated as in equation 2.3.

$$P(A \mid B) = \frac{0.85 \times 0.05}{0.0995} = 0.427 \tag{2.3}$$

The result from equation 2.3 states that given the above criteria, the probability of having a deadly disease when testing positive is 42.7% and not 85% which is the intuitive answer.

2.2.1 Multinomial Naïve Bayes

To compute the probability of binary outcomes, like the likelihood of getting heads or tails, when flipping a coin (i.e. there are only two possible outcomes) is called binomial distribution [8]. To compute the probabilities in situations where there are more than two possible outcomes, multinomial distribution is used.

As this thesis considers the frequency of words to find the most fitting deviation form, the multinomial distribution is used. By comparing how many times a word occurs in every deviation, it is possible to statistically find the existing deviation which is most similar to the input.

From equation 2.1, let us say that **D** represents a document and **C** represents a class.

$$C_{MAP} = argmaxP(C \mid D) \tag{2.4}$$

MAP stands for *"maximum a pistori"*, which, in this case, means the most likely class. By using Bayes Rule and dropping the denominator the formula can be written as in equation 2.5.

$$C_{MAP} = argmaxP(D \mid C)P(C) \tag{2.5}$$

Moreover, when document D is represented as features $(x_1, ..., x_n)$, the formula can be written as in equation 2.6.

$$C_{MAP} = argmaxP(x_1, x_2, ..., x_n \mid C)P(C) \tag{2.6}$$

This method of MNB is called the "bag of words assumption", which assumes that position does not matter. Figure 2.3 shows how a bag of words is represented before any operation is done.

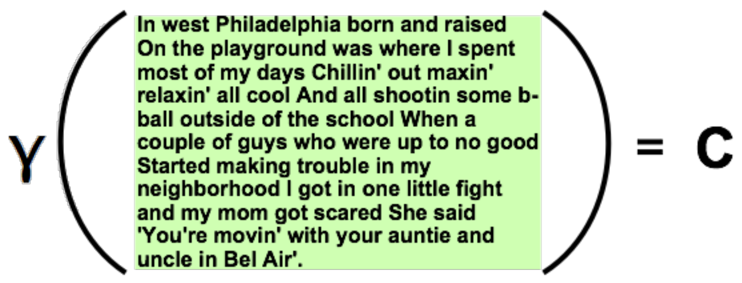
Y
In west Philadelphia born and raised On the playground was where I spent most of my days Chillin' out maxin' relaxin' all cool And all shootin some b-ball outside of the school When a couple of guys who were up to no good Started making trouble in my neighborhood I got in one little fight and my mom got scared She said 'You're movin' with your auntie and uncle in Bel Air'.
= C

FIGURE 2.3: Bag of words assumption before any operations.

To make the process of classification better, the most frequent words are deleted. Figure 2.4 shows how the bag of words is represented after the most frequent words are removed from the "bag". Since NB uses a statistical approach to the input, based on how often a word appears given the database, it is necessary to remove the most common words to decrease processing time, as well as increasing the chance of finding a more reasonable suggestion.

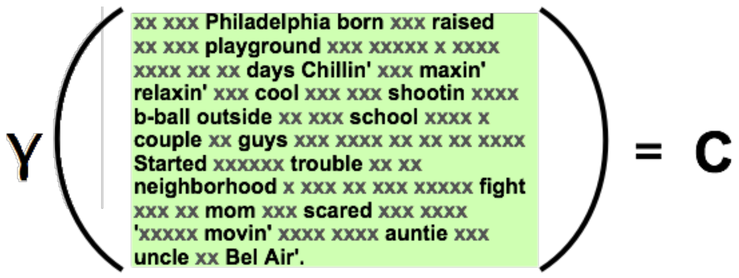
Y
xx xxx Philadelphia born xxx raised xx xxx playground xxx xxxxx x xxxx xxxx xx xx days Chillin' xxx maxin' relaxin' xxx cool xxx xxx shootin xxxx b-ball outside xx xxx school xxxx x couple xx guys xxx xxxx xx xx xx xxxx Started xxxxxx trouble xx xx neighborhood x xxx xx xxx xxxxx fight xxx xx mom xxx scared xxx xxxx 'xxxxx movin' xxxx xxxx auntie xxx uncle xx Bel Air'.
= C

FIGURE 2.4: Bag of words assumption after removing frequent words.

Table 2.1 shows how the bag of words is represented when using it in the implementation. This will be thoroughly explained in Chapter 3.

Philadelphia	1
born	1
raised	1
playground	1
...	...

TABLE 2.1: How the bag of words are represented in an array.

The bag of words model - also well known as the *unigram* model - does not care about the order of the words.

2.3 Artificial Neural Network

?? As the name implies, an Artificial Neural Network (ANN), comes from and is inspired by how the neural network (NN) in a brain is functioning[1]. The human brain is excellent at seeing patterns and using connections from previous experience to decide whether a symbol represents e.g. the letter *g*, or the number *9*. This is done by a system consisting of several billions of neurons in the brain which makes the human able to learn [10]. A neuron collects signal spikes of electrical activity from a connection of

[1]Much is still unknown about how the brain trains itself to process information. The NN model is therefore still a theory in development [9]. it is, however, this theory ANN is based on.

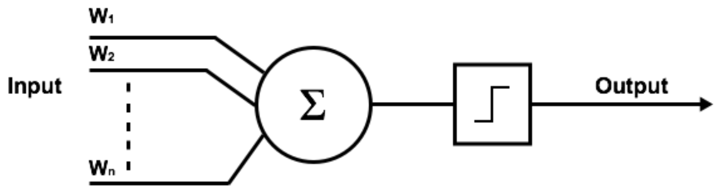

FIGURE 2.5: A perceptron with n number of inputs.

thousands of other neurons. When it receives excitatory input that is sufficiently large, compared with its inhibitory input, it sends an electrical spike. The learning occurs by changing the effectiveness of the synapses. In that way, the influence of one neuron on another change [9].

2.3.1 The Perceptron

In 1957, Frank Rosenblatt invented the perceptron. A perceptron is the simplest possible form of an NN - a computational model of a single neuron [11, 12]. There can be several inputs on a perceptron, but only one output. A perceptron follows the *feed-forward* model which is explained in Section 2.3.2. Figure 2.5 represents a perceptron; N number of inputs with different weights are summed up before it gives a binary output, typically 1 or 0. Perceptrons , as shown in equation 2.7 where x is the input, w is the weight, while b is a bias[2].

$$f(x) = \begin{cases} 1 \quad if \quad w \times x + b > 0 \\ 0 \qquad otherwise \end{cases} \tag{2.7}$$

2.3.2 Feed-Forward ANN

In feed-forward ANNs, the information flows in only one direction - forward. There are no feedback loops i.e, it does not remember previous operations [14, 15]. To explain how the feed-forward ANN learns, a perceptron will be used as an example.

[2]The bias is an offset which, in effect, allows the user to shift the activation function to the left or right [13].

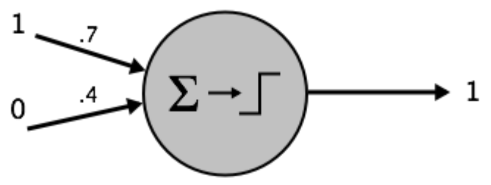

FIGURE 2.6: An example of a perceptron with two inputs.

Figure 2.6 shows a perceptron with the two inputs 1 and 0, with the respective weights of 0.7 and 0.4. This gives the output 1 since $1 \times 0.7 + 0 \times 0.4 = 0.7$.

$$w_{new} = w_{old} + \alpha(desired - output) \times input \tag{2.8}$$

Like the NN in a brain, ANNs are large collections of simple neurons (artificial neurons, that is)[3]. Each neuron is connected with many others, which links and can enhance or inhibit the activation state of any other adjoining neuron. In short, a simple neuron is a device with several inputs, and only one output. Figure 2.7 shows how every input are weighted and summed up before going through an *activation function* - typically a sigmoid function, which determines the output value of the neuron between 0 and 1, or -1 and 1 [16].

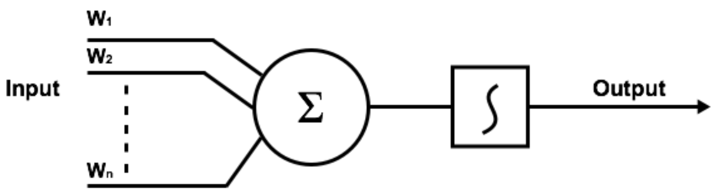

FIGURE 2.7: A simple neuron summing n number of weighted inputs, before going through an activation function and a single output is given.

Figure 2.8 shows how a feed-forward ANN works. Each connection has a weight - a number which controls the signal between the two neurons connected together. The network can either generate a *desired* or *undesired* output. Desired output means there is no need for adjustment on the weights, while undesired output means the system alters the weights in order to improve subsequent results - the system learns based on the generated output [14].

[3]Artificial neurons are from here on out referred to only as neurons.

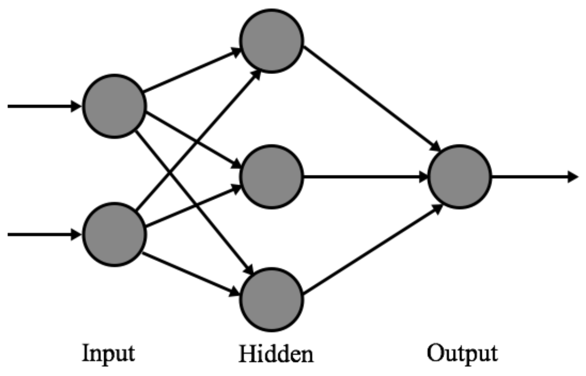

FIGURE 2.8: A small ANN. Each node represents a neuron while the arrow shows the pathway for the flow of information. This example has only one hidden layer between the input and output.

2.3.3 Recurrent Neural Network

According to the definition, recurrent means *returning or happening time after time* [17]. A RNN is, as the meaning of the word recurrent implies, an ANN that uses a feedback loop which makes the network capable of remembering past events allowing information to persist. Figure 2.9 shows an example of a simple RNN. The connections between units on the same layer allow mapping the history of previous inputs to the output vectors. The activation a_k, for each unit k, depends on the inputs $\{x_1, x_2, ..., x_n\}$ with their respective weight connections $\{w_1, w_2, ..., w_n\}$ It can be mathematically described as shown in Equation 2.9 [18, 19]:

$$a_k = f(\sum_{i=1}^{n} w_i x_i + b_k). \tag{2.9}$$

The most common activation functions are sigmoid-, hyperbolic tangent, and linear functions. The aim, during the training phase of an RNN, is to update the weights for a given input to be able to produce an output that minimizes a loss function which is to measure the similarity between the network and the desired output [19]. There are three major steps for training an RNN [18, 19]:

1. Initialize the weights (a small value between -0.1 and 0.1).
2. Forward pass: compute the activation a_k for all units in the RNN.
3. Backward pass: update the weights using a gradient descent for minimizing the loss function between the output of the RNN and the desired output. Backpropagation is used to efficiently compute the gradient and update the weights.

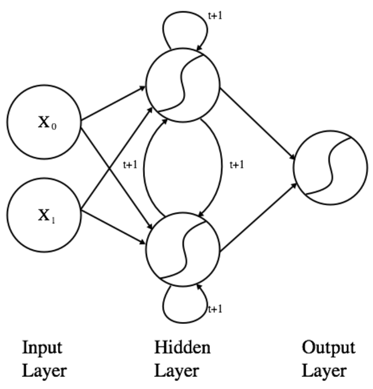

FIGURE 2.9: An example of a simple RNN.

These three steps are repeated until a minimum of the loss function is reached. The solution converged might, however, represent a local minimum.

3 State-of-the-Art

This chapter presents existing NB classification techniques and other common text classification algorithms as well as difficulties and how the said difficulties are addressed by others for text generation, using RNN.

3.1 Text Classification

There are multiple ways of classifying text. This thesis considers text classification using machine learning. As explained in Chapter 2, there are multiple text classification algorithms. This section explores the most common ones, and reflect on the results of other people's experiments.

Support Vector Machine (SVM) is a very popular algorithm for learning text classifiers from examples. In 1998, Thorsten Joachims analyzed the particular properties of learning with text data and identifies why SVMs are appropriate for this certain task [20]. Joachims gives arguments on SVMs' high dimensional input space and that text categorization problems are mostly linearly separable which give theoretical evidence that SMVs should perform well for text categorization. In his experiments he compares SVM with other text classification algorithms like k-NN (which will be explored later in this section) and his results show, with empirical evidence, that SVMs consistently achieve good performance on text categorization tasks.

A popular data set for doing experiments in text applications of machine learning techniques, such as text- classification and clustering is called *The 20 Newsgroups data set*. The 20 newsgroups data set is a collection, originally collected by Ken Lang, and consist of approximately 20000 newsgroup documents divided evenly across 20 different news groups [2]. Shuo Xu uses the 20 newsgroups data set for text classification and experiments with different models of NB, i.e. Multinomial-, Bernoulli- and Gaussian event model. Xu concludes that the multinomial event model is more accurate than the other models [21].

In 2001, Jason D. M. Rennie, proposed a thesis on how to improve multi-class text classification with NB [1]. He states that Naïve Bayes is the de-facto standard text classifier, and discuss different variants of using NB, depending on the data used and the wanted output. Rennie writes about the Maximum Likelihood Naïve Bayes (MLNB), which is a way of formulating Naïve Bayes which chooses the parameters that produce the largest likelihood of the training data [1].

Furthermore, in 2003, Rennie and others, write a paper [22] where they compare SVM with a slightly modified NB. According to the report, the modified NB is a fast, easy-to-implement, near state-of-the-art classification algorithm, while SVM supposedly is the fastest algorithm for text classification.

In a report by Ge Song and others in 2013 [23], short text classification research has been done. They state that the most traditional methods for text classification (such as SVM, NB and KNN) are all based on the similarity of term frequency. They also state that most of the said classification techniques may fail to obtain high accuracy if the labeled information is insufficient. The processing of text classification is generally in Vector Space Model (VSM), which has the underlying assumption that the relationships of words are independent, neglected the correlation between text. Song [23] brings up a good point regarding the semantics of text - short texts has a weaker capacity of semantic expression. Semantic analysis pays more attention to the concept, and to the correlation of texts to obtain the logic structure, understanding the contents rather than using a statistical approach. Zelikovitz [24] applies an algorithm based on semantic analysis that proposed to deal with short text classification. Quiang Pu etc [25] combine Latent Semantic Analysis (LSA) and Independent Component Analysis (ICA) [26, 27] together. LSA transforms the vector space into semantic space, and based on statistical method, LSA extracts, and quantify the semantic structure, eliminating the correlation between terms [23]. Song etc [23] state that LSA can reduce the high-dimensional vector matrix to construct the low-dimensional subspace which can effectively describe the relationship of term-document. Some Many Dimension reduction methods in LSA are proposed by Deerwester etc [28] and Landaues [29] such as Singular Value Decomposition (SVD), Semi-Discrete Decomposition (SDD) and Non-negative Matrix. Ge Song etc. [23] explores SVD. In Song's report [23] it is discovered that short text classification is a challenging field because many technologies are in the initial stage.

A method for obtaining better undestanding of context it is possible to use a higher level of n-gram language models. In 2003 Fuchun Peng and Dale Schuurmans [30] experiments on how n-gram language models can be used as text classifiers. Peng and Shuurmans' results show that using the word models, using n-gram with $n > 1$, does not demonstrate any significant improvement, and when $n > 2$ the results are even more inaccurate [30].

3.2 Text Generation

Ilya Sutskever and others [31] states that RNN is a powerful tool because of their capability of having a high-dimensional hidden state with non-linear dynamics that remember and process past information. They demonstrate, in the report, a character based language model with an RNN, trained with the Hessian-Free (HF) optimizer for generating text. HF, also known as truncated-Newton, is a 2nd-order optimization approach, which is relatively new in the fields of machine learning [32]. The goal in the report by Sutskever etc [31] is to demonstrate the power of large RNNs trained with the new HF-optimizer by applying them to the task of predicting the next character in a streaming text.

Following is the formalization of the standard RNN: Given a sequence of input vectors $(x_1, x_2, ..., x_T)$, the RNN is able to compute a sequence of hidden states $(h_1, h_2, ..., H_T)$ in addition to a sequence of outputs $(o_1, o_2, ..., o_T)$ by iterating equation 3.1 for $t = 1$ to T.

$$\begin{aligned} h_t &= tanh(W_{ih}x_t + W_{hh}h_{t-1} + b_h) \\ o_t &= W_{ho}h_t + b_o \end{aligned} \tag{3.1}$$

W_{ih} is the input-to-hidden weight matrix, W_{hh} is the hidden-to-hidden (aka recurrent) weight matrix while W_{ho} is the hidden-to-output weight matrix. b_h and b_o are the biases.

By using back-propagation through time, the gradients of the RNN are easy to compute [33, 34]. The relationship between the parameters and the dynamics of the RNN is, in reality, highly unstable which makes gradient descent ineffective. Hochreiter [35] and Bengio [36] formalizes the intuition that it may seem that RNNs are easy to train with gradient descent. They proved that the gradient decays exponentially as it is back-propagated through time. These results were used to argue that RNNs can not learn long-range temporal dependencies when gradient descent is used for training. Furthermore, there is a tendency of the pack-propagated gradient to exponentially blow-up greatly. This increases the variance of the gradients, which makes the learning unstable.

There are some ways to deal with the inability of gradient descent to learn long-range temporal structure in a standard RNN. One way is to modify the model to include "memory"-units which are designed to store information over long time periods. This approach is known as the *Long-Short Term Memory*(LSTM) [37]. LSTM has been successfully applied to complex real-world sequence modeling tasks [38]. LSTM makes it possible to handle data sets which require long-term memorization.

Another way to avoid back-propagation through time problems is the Echo State Network (ESN) [39]. ESN forgoes learning the recurrent connections all together, and trains only the non-recurrent output weights. This makes the learning task easier, moreover, it works surprisingly well, provided that the recurrent connections are carefully initialized.

In 2013, Alex Graves [40] write an article about generating sequences with RNNs. He shows how LSTM RNN can be used to produce complex sequences with long-range structure, simply by predicting one data at a time. The approach he uses is demonstrated for text and online handwriting - where the focus is on handwriting in particular. Graves states that in most cases, text prediction, or language modelling, is formed at the word level. However, because the number of words often exceeds 100,000 it becomes a problematic task to realize. Having so many classes, in addition to requiring many parameters to model demands an enormous amount of training data to adequately cover the possible contexts for the words [40]. Furthermore, there is a difficulty in the high computational cost of evaluating all the exponentials during training. Besides, word-level models are not applicable to text data containing non-word strings, such as multi-digit numbers or web addresses.

According to Graves [40] Character-level language modelling with NN is found to give slightly worse performance than equivalent word-level models [31, 41]. Graves states that predicting one character at a time, however, is more interesting from the perspective of sequence generation, as it allows the network to invent original words and strings [40].

In a report by Razvan Pascanu and others[42] they address two issues for training an RNN, i.e. the *vanishing* and the *exploding* gradient problem, which is detailed in Bengio's report [36]. They look at previous solutions to the problem, where one of them is Doya [43], who proposes, in 1993, to pre-program the model or to use *teacher forcing*. The downside is that it is not always possible to know the required asymptotic behaviour, and, even if it is known, it might not be trivial to initialize the model accordingly. Pascanu reviews Hochreiter and Schmidhuber [37] and Graves *et al.*[38] who proposes the LSTM model to deal with the vanishing problem. This solution does not, however, address the exploding gradients problem explicitly. Another way to deal with the two said problems, is to use the Hessian-Free optimizer in conjunction with *structural damping*, which is proposed by Sutskever *et al.* [44]. According to Pascanu etc [42] the Sutskever *et al.* approach [44], regarding the vanishing gradient problem works because in high dimensional spaces there is a high probability for long-term components to be orthogonal to short term ones. Regarding the exploding gradient, Pascanu *et al.* [44] take curvature into account. The Jacobian matrices $\frac{\partial x_t}{\partial \theta}$ are forced, by the enhancement called structural damping, to have a small norm, hence further helping with the exploding gradient problem. Pascanu *et al.* [42] put, after looking at previous solutions, forward their assumption stating that gradients explode, there is a cliff-like structure in the error surface and devise a simple solution based on this hypothesis, i.e. clipping the norm of the exploding gradients. This solution [42] provide some indirect empirical evidence towards the validity of their hypothesis, even though further investigations are required for more evidence.

4 Implementation

As mentioned earlier, this thesis consist mainly of two topics, namely text classification with multiple classes and text generation with a small dataset.

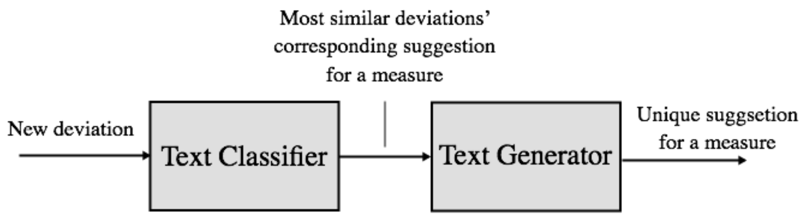
Most similar deviations' corresponding suggestion for a measure
New deviation
Text Classifier
Text Generator
Unique suggsetion for a measure

Figure 4.1: A graphical illustration of the system.

Figure 4.1 shows a graphical illustration of the implementation. The input is a new deviation and is sent to a text classifier which gives an output of the most similar deviations. The similar deviations send in their corresponding suggestions for a measure into a text generator which in the end give a unique suggestion for a measure, which is a combination of the said suggestions for a measure. This chapter goes through the approaches used for text classification and text generation.

4.1 Text Classification

There are multiple algorithms for classifying text using machine learning. For the classification, *SKlearn's* libraries for text classification are used [45]. To use SKlearn for classification the input needs to be in an array format. That is, each deviation needs to be represented as an array. If there is a document saying "*a man buys a nice car*", it will then be presented as shown in Table 4.1[1]. The representation of each deviation needs to be represented with an equally long array as there are unique words in all the documents.

[1]This representation is the *bag of words* representation, as described in Chapter 2

Word representation	a	man	other	words	buys	nice	car
Array representation	2	1	0	0	1	1	1

TABLE 4.1: An example of how the text documents is represented in an array format.

Algorithm 1 shows how the pseudocode for making the data ready for training, using SKlearn's libraries for text classification.

Algorithm 1 Pseudocode for preparing the data to be used with SKlearn's libraries.

```
1: Initialize variables
2: Import deviations from file
3: Initialize i = 0
4: for $deviation \in deviations$ do
5:     deviationArray[i] = {}
6:     for $word \in deviation$ and $word \notin stopwords$ do
7:         $deviationArray[i] \leftarrow word$
8:         if word.isUnique then
9:             $UniqueWords \leftarrow word$
10:        end if
11:    end for
12:    i += 1
13: end for
14: number = 0
15: for $word \in UniqueWords$ do
16:     wordInNumberArray = word.mapWordToNumber(number)
17:     Initialize number += 1
18: end for
19: for $deviation \in deviationArray$ do
20:     $deviationAsnumberArray \leftarrow deviation.representedAsArray$
21: end for
```

The *stopwords* in line 6 are words which are very common (words like it, is, then, the, etc.). They are removed for making the classification better, as well as making the training phase faster.

The data for classifying is circa 450 KB (kilobytes), consisting of approximately 2000 deviation forms. The length of the deviations varies from a couple of words to several sentences. Within the 2000 deviations, all together, there are approximately 7300 unique words which means that each deviation will represent an array of 7300 elements. The final array will, therefore, be a 2000 $\times$ 7300 matrix.

The classifier does not have any predefined classes, and since the deviation forms contains no specific topics, every deviation gets its own class, i.e. there are corresponding circa 2000 classes. When the implementation of the code takes an input, the output will be the deviation which is most similar to the input.

Input	**Modified Input**
the	...
man	man
did	...
not	...
remember	notremember
to	...
wear	wear
his	...
sunglasses	sunglasses

Table 4.2: The left column displays the word arrangement before any operations, while the right column shows how the input might look like when the stop words have been removed, and the *"not"* has been combined with the following word.

Deviations are descriptive texts. It is, therefore, relevant when a sentence has the word *"not"* in it. *"The man did not remember to wear his sunglasses"* means the exact opposite of *"the man did remember to wear his sunglasses"*. The word *not* is, however, one of the stop words, which means that it is not taken into consideration. To solve this problem, I combine the *not* word with the next word in line. That is, if a sentence represents the example with the sunglasses, then the algorithm from Chapter 3 will fetch the words, as shown in Table 4.2.

When the input is classified, the output is ready to be inserted in an RNN for training and generating an unique suggestion for a measure.

4.2 Text Generation

For the text generation two RNNs are used, namely a char-RNN and a word-RNN.

Andrej Karpathy has made a char-RNN code which implements multi-layer RNN for training/sampling from character-level models [46]. That is, it takes a text file as input and trains an RNN that learns to predict the next character in a sequence. The implementation made by Karpathy makes it possible for the users to adjust parameters for tailoring the RNN to fit the data used.

In Table 4.3 the essential parameters are displayed. The *RNN size* decides the size of the LSTM internal state, *number of layers* decides how many hidden layers there should be in the LSTM, *sequence length* sets the number of time steps to unroll for, while *batch size* is the number of sequences to train in parallel. There are other parameters, like what the fractions for training- and validation data should be[2], but

[2]training- and validation data is divided into 80% and 20% respectively.

Essential parameters			
RNN size	number of layers	sequence length	batch size

Table 4.3: The essential parameters in the char-RNN.

they do not have a significantly impact of the output (according to Karpathy [46]). The two most important parameters that control the model are the RNN size- and number of layers. As this thesis is all about generating text using a small amount of data, the parameters will be adjusted thereafter. In Chapter 5 we will experiment with different values of the essential parameters.

The word-RNN (made by Sung Kim[47])is mostly reused code, which was inspired from Andrej Karpathy's char-RNN.

5 Results

This chapter covers all the results from the two parts of this thesis i.e. the text classification and the text generation. The input and verification data used in this thesis is written in Norwegian. It is not important for the sake of the results what language it is. It is, however, important that the output is as expected. The results, of which the output text is relevant, will be translated into English.

5.1 Naïve Bayes Classifier

As described in Chapter 4, the implementation for text classification can be used for SKlearn's libraries. The ones tested are Multinomial Naïve Bayes (MNB), Support Vector Machine (SVM) and K-Nearest Neighbor (KNN), concerning the expected output and performance. When using MNB, all classes will get a probability of similarity[1]. SVM and KNN do not use a statistical approach meaning there will not be a distribution for the most similar ones, and only for the single "best" one.

Furthermore, the MNB classifier is the fastest of all three approaches. Table 5.1 shows a comparison of how much time each of the algorithms need for training the data. SVM is the slowest (by far), while MNB beats the KNN in a fraction of a second. All algorithms returns the same output, i.e. the expected output[2].

Algorithm	**Time Training**
SVM	32.9 seconds
KNN	3.0 seconds
MNB	2.8 seconds

Table 5.1: A small table comparing the the training duration between SVM, KNN and MNB.

[1]The sum of all probabilities for the deviations will result in 1.0 or 100%.

[2]Section 5.1.1 gives a detailed explanation of the expected output.

Based on the training time in Table 5.1 and the fact that MNB uses a statistical approach which gives a probability output for every deviation (making it possible to easily gather the five most similar suggestions for a measure to generate an unique measure, using RNN later on), the text classifier chosen for this thesis is MNB.

The tests considered are divided into the following: "*Proof of Concept*" , "*Proof of Concept using Modified Inputs*" and "*Validation of the Classifier*". The section with "*Proof of Concept*" will use already existing deviation as input, and the goal is for the output to be the same as the input. For the section with "*Proof of Concept using Modified Inputs*", the input consist of existing deviations as well, but slightly modified. The final section, "*Validation of the Classifier*", will consist of random deviations created by someone else than the author of this thesis.

5.1.1 Proof of Concept

In these tests already existing deviation are used, of which the output should be equal to the input. This is solely to see if the algorithm can find the same output to be equal to the input, then the classifier works as intended.

When searching for a deviation, which is exactly the same as an existing one, the algorithm always finds the output to be exactly the same as the input. Even though the distribution varies from a few percents to 99%[3], the "correct" i.e. the expected answer is found. However, when the input text is only a couple of words, the algorithm becomes very uncertain.

Table 5.2, shows an example of when the input is intentionally put as one of the already existing deviations. In this example, the input is only a short sentence. In English the input means: *"The card reader on pump 1 is bad."* Even though the most likely deviation is below 0.5%, it finds the expected one. The second most likely deviation has, more or less, the same probability, but due to the output is (in English): *"The card reader on pump 1 seems bad"*, there is no wonder the probability is almost the same.

[3]It will most likely never reach 100% due to the removal of the stop words.

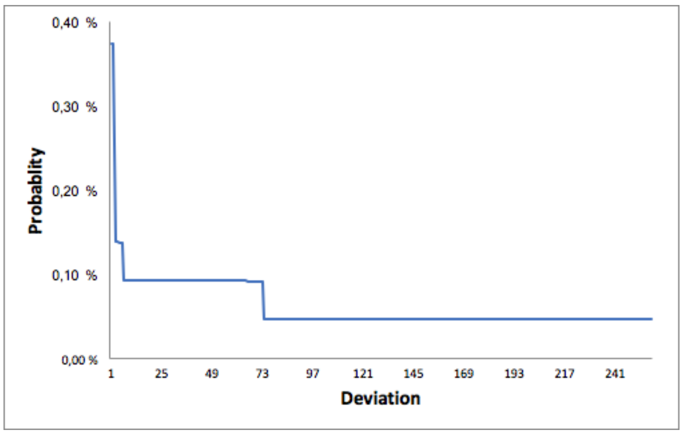

FIGURE 5.1: A graphical representation of the output of the 250 top suggestions when the input is as in Table 5.2. The y-axis represents probability while the x-axis represents deviations.

Input: Kortleser på Pumpe 1 er dårlig	
Probability	**Output**
0.3749%	Kortleser på Pumpe 1 er dårlig
0.3747%	Kortleser på pumpe 1 virker dårlig
0.1402%	Pumpe nr 1 ga melding om ”ingen pumpe tilgjengelig” selv om pumpen var ledig. Skjedde søndag ettermiddag
0.1401%	Kunde Odd Steinar rotet olje fra sin egen bil mellom pumpe 1 og 2 og ved pumpe 7. Var hull under bilen.

TABLE 5.2: An example where the input is the same as an existing deviation. The column to the left shows the probability of its respective output, which is found in the right column.

The example in Table 5.2 is great to show how the algorithm is able to find the expected output, using only a few words. One problem occurs, however, when the input text is slightly changed. When giving the input: *“The card reader is not bad”*, the output is the same as in Table 5.2. It makes no sense writing a deviation about a card reader not being bad, but it shows how short deviations are subject to potentially obtaining the exactly opposite suggestion for a measure than expected. For longer texts this problem is slightly avoided.

Table 5.3 and 5.4 are examples of how the probability changes by adding the *not*-word for describing a, more or less, opposite situation of the original one. Table 5.3 shows an original deviation and its corresponding output while in Table 5.4 the deviation has added the *not*-word to see if there are any changes. The algorithm finds the same deviation, but with a lot smaller probability (from 25% to 14.5%).

Input: Skulle krane ei skinnelengde som lå vanskelig til. Lengden glapp slik at enden spratt opp og nesten traff basen	
Probability	**Output**
25.2%	Skulle krane ei skinnelengde som lå vanskelig til. Lengden glapp slik at enden spratt opp og nesten traff basen
0.394%	Skulle kranevekk ei skinnelengde da ei skinne i enden på lengden datt av og traff hytta på bilen og forårsaket en liten bulk

Table 5.3: An example where the input is the same as an existing deviation. The column to the left shows the probability of its respective output, which is found in the right column.

Input: Skulle krane ei skinnelengde som lå vanskelig til. Lengden glapp slik at enden ikke spratt opp og nesten traff basen	
Probability	**Output**
14.5%	Skulle krane ei skinnelengde som lå vanskelig til. Lengden glapp slik at enden spratt opp og nesten traff basen
0.45%	Skulle kranevekk ei skinnelengde da ei skinne i enden på lengden datt av og traff hytta på bilen og forårsaket en liten bulk

Table 5.4: A short text example where the input is the same as an existing deviation, but with adding *not* to the sentence, making it somewhat opposite of the example from Table 5.3. The column to the left shows the probability of its respective output, which is found in the right column.

Furthermore, Table 5.5 shows an example where there is an even longer input. The most likely deviation has 88.2% probability of being the correct one while the rest is below 1%. Looking at Tables 5.2 and 5.5 and after doing further empirical tests, it is easy to see that the more words used in the input, the more certain the algorithm is of finding the most likely deviation based on the given data.

Input: Verkstedet har hatt inne 8508 på full sjekk! Jobben som er gjort er meget bra og det er nå tomt for feilkoder! T.o.m abs er i orden og ingen blinkende lamper!!! BRA!!! :-))) DVS ABS feil er kommet tilbake nå da, men men...	
Probability	**Output**
88.2%	Verkstedet har hatt inne 8508 på full sjekk! Jobben som er gjort er meget bra og det er nå tomt for feilkoder! T.o.m abs er i orden og ingen blinkende lamper!!! BRA!!! :-))) DVS ABS feil er kommet tilbake nå da, men men...
<1%	Nå fungerer henger til 1066 som den skal.Ingen feil med ABS eller SAM
<1%	Det skulle skiftet abs føler på en bil. det ble skiftet føler på feil side, selv om det sto riktig på notaplan.Bilen må inn igjen får og få skiftet rett føler.

TABLE 5.5: A long text example where the input is the same as an existing deviation. The column to the left shows the probability of its respective output, which is found in the right column.

5.1.2 Proof of Concept using Modified Inputs

This section will show the results when experimenting with inputs which are similar to already existing deviations. This section's purpose is the same as in the previous section. That is, to find the expected output. The difference is, in this section the texts are slightly modified, by removing parts of the text.

Table 5.6, shows what the input is and what the expected output should be, in addition to showing what the actual output is, as well as the probability of the corresponding result based on the given data. The examples in Table 5.6 consist of both short and long texts. The short text gives a probability of scarce 0.2% and the long texts get between 4- and 5% which all are considered small, but they all find the expected output. In the bottom row, the deviation is modified as much as possible, while still making, somewhat, sense and it still was able to find the expected output.

Input	**Expected Output**	**Actual output**	**%**
fyringsolje påfylling ikke merket	fyringsolje påfylling ikke merket ved 1-2-3 Hafrsfjord	fyringsolje påfylling ikke merket ved 1-2-3 Hafrsfjord	0.19
Det var gått ett snøras på vegen mellom festøya i Ørstad i Møre og Romsdal	Sjåfør ble stående i timevis og vente til vegen skulle åpne! Det var gått ett snøras på vegen mellom festøya iØrstad i Møre og Romsdal.	Sjåfør ble stående i timevis og vente til vegen skulle åpne! Det var gått ett snøras på vegen mellom festøya iØrstad i Møre og Romsdal.	4.39
Tusen takk til han som orka å høre på me å komme me forslag på natta	Tusen takk til Øystein som orka å høre på me å komme me forslag på natta når fryseren stoppa:)	Tusen takk til Øystein som orka å høre på me å komme me forslag på natta når fryseren stoppa:)	4.95
Lekkasje av når du losser. Lekkasje nederst i kasse.	Lekkasje av 95 i Ørsta når du losser,det ligger absorberende kluter nederst i kasse fra før.Lekkasje nederst i kasse mellom glasset og bunnen.	Lekkasje av 95 i Ørsta når du losser,det ligger absorberende kluter nederst i kasse fra før.Lekkasje nederst i kasse mellom glasset og bunnen.	0.56

Table 5.6: Caption

5.1.3 Validation of the Classifier

In this section the MNB text classification algorithm will be put to the test regarding new deviations. The new deviations are created by a person without insight in this report to get an objective perspective for the validation. The purpose of these tests are to see if it is possible to use the existing data set to find similar suggestions for a measure when a, more or less, random input is given. The deviations to validate are shown in Table 5.7, which will be classified based on the given data set with approximately 2000 classes.

Figure 5.2 shows the distribution of the output of the 250 most similar deviations, using the deviations from Table 5.7. All the curves resemble the graph, which is in the hypotheses from Chapter 1. Taking a closer look at Figure 5.2, all the graphical representations of the probability distributions, are giving the most similar deviations probabilities of 1% or less, except for "deviation 3". Finally, Table 5.8 displays the output of their respective inputs from Table 5.7.

Nr	Input
1	Jeg ble stoppet på veien av det som så ut som en regelmessig kontroll. Politiet sa at ene lyset bak på bilen ikke virket, og at jeg måtte få det fikset så fort som mulig
2	Det var bestilt opp feil type rekkverk og det var umulig for oss å gjøre noe som helst.
3	Jeg fikk ikke til å sette bilen i revers da jeg skulle parkere bilen. Jeg prøvde å slå bilen av og på igjen, men fikk fortsatt ikke sette bilen i revers. Jeg prøvde å sette bilen i første gir, og kjørte et lite stykke fremover før jeg prøvde å sette bilen i revers igjen. Det gikk heldigvis da, og fikk parkert bilen. Da jeg skulle starte bilen igjen fikk jeg fortsatt ikke til å sette bilen i revers.
4	Vi hadde en EU-kontroll av en Peugeot GT-Line og den manglet kun en refleksvest, men vi skrev på kontrollen at kunden måtte sende inn bilen for å reparere girkassa.
5	Antennen fikk ikke til å ta inn nøyaktige signal, så etter at jeg registrerte sporene på rekkverket ble det vist helt feil i systemet. Det var gult signal på antenna, og det trodde jeg var bra nok.

Table 5.7: A list of five deviations which are used for validating the text classification.

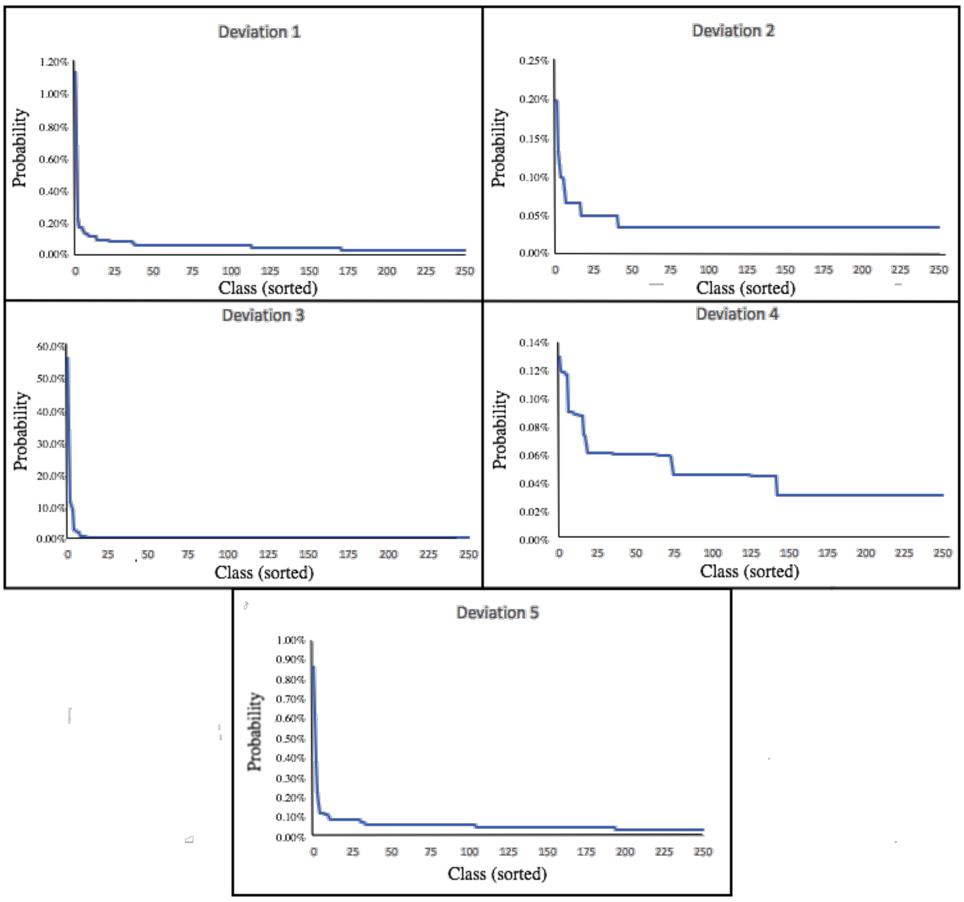

Figure 5.2: Five graphs representing the probability distributions of the five deviations from Table 5.7. The x-axis shows the the 250 most similar deviations (i.e. the classes) to the input, while the y-axis is the corresponding probability.

Nr	Output	Prob
1	Kom kjørende på RV65 ved Svorkmo på svingete vei da jeg møtte en tømmerbil i venstresving. Passerte hytte på bilen hans så smalt plutselig speilet inn og i ansiktet på meg. Må ha truffet noe bak på hans bil. Holdt på å besvime, var utom veien på høyre side men klarte å komme innpå igjen. Bare flaks at bilen kom på veien igjen, hvis ikke hadde det gått rett i fjellet. Fikk stoppet på en lomme rett foran, ringte Solheim som videre varslet 113 og 112 før han kom tilbake. Blødde fra ansiktet. Ble kjørt i ambulanse til sykehus, hvor de plukket glass og stelte sår. Politiet stoppet også tømmerbilen fra Vuttudal som bare hadde kjørt videre uten å merke noe. Politiet har opprettet sak.	1.13%
2	Rekkverk over undergang hadde feil lengde i forhold til mur, feil bolter og for langt rekkverk med feil vinkel	0.20%
3	Sku hente Atego'n til G-lag på Molde. Hadde dårlig tid på ferga, så droppet å fylle selv om det var lite på tank. Det var ingen varsellampe om at bilen var på reservetank, så regnet med at det holdt til Åndalsnes. Bilen stoppet på kul'n før Nebbatunellen, så fikk trillet bilen på statoil og fylt diesel. Bilen ville ikke starte etter fylling, og fant heller ingen fødepumpe eller utstyr til å jekke hytte ned. Ringte 3 mann fra G-lag uten å få svar, og verksmester 3 ganger pluss at jeg sendte han sms. Ingen respons... Satt igjen bilen på Statoil. - G-lag ringte opp igjen etter at jeg hadde dratt, og kunne fortelle at det ikke fins fødepumpe på bil og at han må ha trykkluft på tank for å få diesel frem til motor.	55.8%
4	kjørte gjenom lærsta tunelen og fik fartsmåling over 80. da speedopmeteret viste rett over 70. speedometeret viser nesten 10 km/t forlite på grunn av at verkstedet har sotte på alt for store hjul. så kan få ei dyr overaskelse til jul, vist eg får fartsbot for alle fotoboksene eg har pasert dei siste 6 ukene. og får eg prikkbelastning så kan eg bli lapplaus!!! detta er ein uakseptael feil, som verkstedet var klar over da dei sette på hjula og sa me måtte tenke over hjulbuan og heve bilen på lufta...(som då var deira løsning)må kjøre med bilen heva på lufta hele tida, og det hørtest tøft ut nor eg måtte avlaste boogien for å få meir tyngde på driven.	0.13%
5	For dårlig signal fra satelitter. Store problemer på Lesja/Lora under telling. Varierende forhold. Ofte fikk man bare rødt signal om man stod å prøvde aldri så lenge. Men om man kom tilbake 10 timer senere kunne man oppleve å få grønt signal umiddelbart.Sist uke under telling med gtac mistet den kontakten med altus. Måtte slå av/på så virket den. Skjedde 2 ganger.Etter oppdatering til versjon 2,5.Fungerte fint en stund, så kom denne beskjed midt i en telling:" En feiloppsto:Kunne ikke starte Altus B lag. Sjekk Altus og altus-adresse og prøv på nytt. Slo av/på antennen mange ganger. Samme feil kommer.Slår til slutt av både gtac og antenne, da får gtac kontakt med altus å kunne fortsette telling.	0.87%

Table 5.8: The output of the algorithm when the input is as shown in Table 5.7.

Analyzing the input from Table 5.7 compared with the corresponding output from Table 5.8, it is easy to see that deviation 1,3,4 and 5 has no correlation. Using the suggestion for a measure for the said deviations will not make any sense. On deviation

2, however, the subject is not just the same, but the problem is almost identical. In Table 5.9 the five top outputs from Deviation 2 in Table 5.7 are given. By analyzing the content, it is fairly easy to see that all these deviations has the same subject, and a combination of all the suggestions for a measure for all the said deviation could potentially create a unique suggestion for a measure to the input, which should make sense. In Section 5.2 we will see if we are able to use RNN to create a reasonable suggestion for a measure using the corresponding suggestions for a measure from the output in Table 5.9.

Input: Det var bestilt opp feil type rekkverk og det var umulig for oss å gjøre noe som helst.	
Nr	**Output**
1	Rekkverk over undergang hadde feil lengde i forhold til mur, feil bolter og for langt rekkverk med feil vinkel
2	VikØrsta leverte feil rekkverk mot ordre.AG lossa bakskinne/påler/sk. som skulle lenger nord. Skal ikke brukes nå - må flyttes.Tegninger som var sendt med var feil.Bestilt feil lakkert sigma (feil lengde og hullbilde)
3	Var registrert feil type rør på bruen. Må ha nøyaktig beskrivelse når det ikke er vanlig rekkverk
4	Laget ble trekt i lønn for å ha satt opp rekkverk på feil sted, ble så trekt i lønn for dette. De mener dette er urettferdig, og at de da skal ha kompensasjon når andre gjør feil som rammer de.
5	Rekkverksmateriellet stemte ikke overens med rekkverksmateriellet. Feil avstand mellom boltegrupper i forhold til rekkverksmateriell. Feil radie på handlister. Feil hullbilde på handlister, feil lengde på handlister, feil lengde på sprosser. Brukte mye ekstra tid pga dette. Etter litt om og men så skulle vi prøve så godt som mulig å montere materiellet. Måtte kappe, rette sveise og tilpasse handlist. Kappe og tilpasse sprosser og vinkler samt borre opp nye hull.

Table 5.9: The five top outputs when the input is deviation 2 from Table 5.7.

5.2 Text Generation with RNN

The purpose of this section is to see if it is possible to create unique suggestions for a measure which makes sense, given the input deviation. As described in Chapter 4, two approaches are explored, namely word- and char-RNN.

5.2.1 Proof of Concept

First we will see that the algorithm works, using data set known as *"tinyshakespeare"*, which is a document consisting of a subset of works, made by Shakespeare, before the data set with all the deviations will be used. The example from Table 5.12 shows how

RNN size	**Number of layers**	**Sequence length**	**Batch size**
128	2	50	50

Table 5.10: The parameters used for training with the Shakespeare and deviation data set using the char-RNN.

RNN size	**Number of layers**	**Sequence length**	**Batch size**
256	2	50	25

Table 5.11: The parameters used for training with the Shakespeare and deviation data set using the word-RNN.

powerful the RNN can be when used on a proper amount of data. The left column displays the original text while the middle and right column show the text generated using char-RNN and word-RNN respectively, after training for a while (approximately 30 minutes)[4]. Taking a closer look at Table 5.12[5], the char-rnn's output does not make any sense. The word-rnn's output, on the other hand, looks real.

The parameters used for training with the Shakespeare and deviation data set are the default values, shown in Table 5.10 and 5.11. These tests are mostly for proof of concept, and I have therefore not performed any more empirical tests for the two said data sets.

Table 5.12 shows how great the RNN is at learning and generating its own text, with a reasonable amount of data. It is therefore interesting to see if it is possible to generate equally good text using only a small set of data[6].

As the example used in this thesis are deviation forms, we will see if it is possible to create an original deviation based on the deviation forms in the data set. There are about 2000 existing deviations, consisting of approximately 440KB of text (which is less than half of the amount of data compared to the Shakespeare data). Table 5.13 shows two examples (one long, and one short deviation) of original- and generated deviations using the said data set. The generated deviations in the right column of Table 5.13, does not make much sense.

[4]The training loss stopped decreasing after 25 minutes or so, meaning that the generated text would not become any better at this point. That is, it would not be necessary to train the RNN any further.

[5]Note that the input is only clear text, as well as the output. I have made the names bold for cosmetic intentions only.

[6]The Shakespeare data set is approximately 1 MB, which actually already is considered as a small data set.

Training data	char-rnn output	word-rnn output
DUKE OF AUMERLE: Yea, at all points; and longs to enter in. **Lord Marshal:** The Duke of Norfolk, sprightfully and bold, Stays but the summons of the appellant's trumpet. **DUKE OF AUMERLE:** Why, then, the champions are prepared, and stay For nothing but his majesty's approach. **KING RICHARD II:** Marshal, demand of yonder champion The cause of his arrival here in arms: Ask him his name and orderly proceed To swear him in the justice of his cause.	**GLOUCESTER:** Then mercy! let their shepherd, whose place, I access you for her. **Son:** Even uncle. I perceived so much uneven With thyself on my beg, and cat all guilty: Ha, with a have; be revel I allow. **HENRY BOLINA:** A swallow, nurse, I see her royal heart! Hath wit, e Turn unform'd arrestory this is his watch. And little dukedication. The wrong up, Bein thy earthoment, no foregen his gentleman, Desparted unhoolesh ones shows to rite are drunk.	**CLARENCE:** The noble masters, Clarence for the cause of that I had thee! To make the orator as still as I am bigger, but let thy dauntless mind Still ride in fury with the ireful part of any of our meeds, He **Shepherd:** Why, let the pebbles on the hungry beach Fillip the stars; I'll slay the helpless king. Lord of mine love the Duke of York is slain! **EDWARD:** O Warwick, Warwick! that Plantagenet, Which will you grant me with the selfsame king? 'Tis time the murderer will be slain, That let the mutinous winds Strike the Duke of York was slain!

TABLE 5.12: An example which compares some of the original text (input) with some of the generated text (output) after training char-, and word-RNN to write Shakespeare.

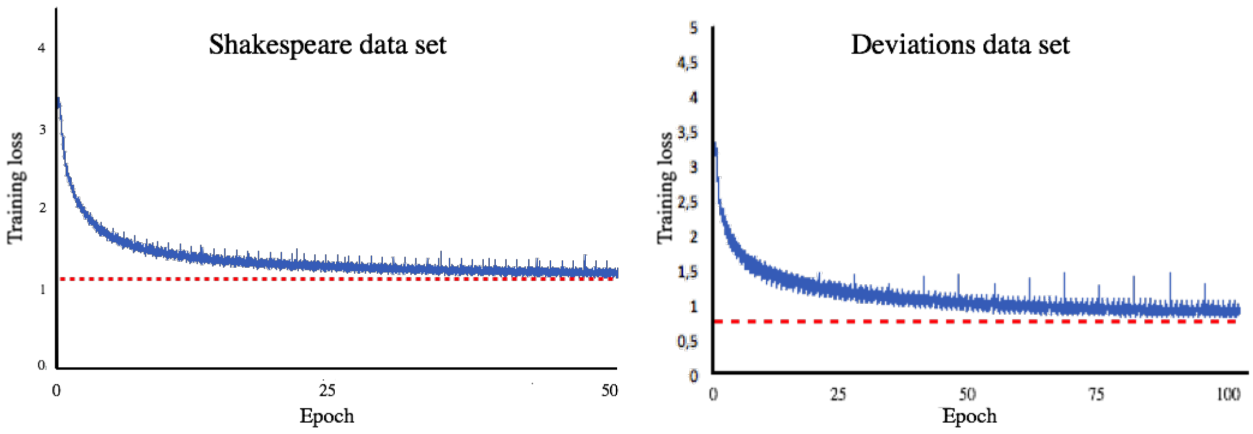

FIGURE 5.3: A graphical representation of the char-RNN training loss on the data set with Shakespeare and all the deviations, respectively.

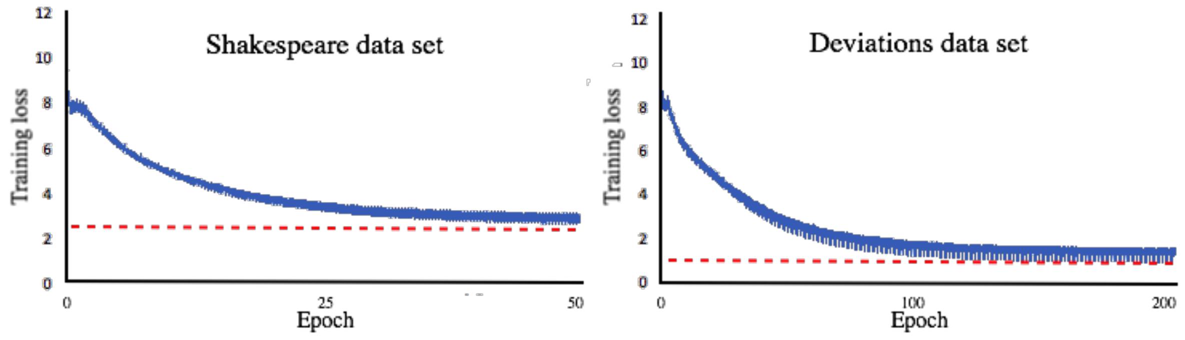

FIGURE 5.4: A graphical representation of the word-RNN training loss on the data set with Shakespeare and all the deviations, respectively.

Training data	**char-rnn output**	**word-rnn output**
Juli: I Juli leide vi inn en gammel Heggem- sjåfør. Han er ikke vant til å tenke på at han skal kjøre etter kjøre og hviletid fordi han kjører tipp-bil til vanlig. Derfor ordner pausene seg selv grunnet faste mat- pauser. Han har i løpet av Juli hatt en del brudd, ikke store men litt her og der. Han har flere steder ikke fått god-kjente 45 min, pauser. Han kjører ikke for oss nå så prob-lemet er ikke så stort. Men hvis han skal kjøre hos oss ig-jen skal han få opplæring på rutinene, lovene og reglene!	Sjåfør glemte å ta ut under fylling, blir det seg ukjørte me dagen og fått hjulponinits stoppet av C26627 jeg trodder på obe soflirmidd. Verkste og forsyner dette var. Ringte seg ikke fylle hjøretiden. Da alt varm og virke. soven på vaske-maskin. pumpekontorene til F4 å som godkall god i - metinga til Statoil Sunndve-len slåg, da skilles med grun-net vi hadde den tilfaker på volamen som holdt på ca kort meg på truckbra:	det var på måndalen Kunden ringte Monica og pølsedisken. under Autotank og den ankom Statoil Valderøy, fikk ikke sikkerhetsmargin i denne. Etter skift av toppakning på verksted ringte det som av-talt så jeg skulle tilpasse en kjetting til forrige jobb :) Da sto den inn på verksted å sa at jeg skulle levere på hvil mange timer døgnhvil. Jeg ringte igjen og da kom det en annen plass!
Jeg skal gjøre mitt for at jeg selv skal bli bedre. Dette er en fest. Eller er det test ?	Telemer for står døgnhvil må skjøres og pent blir trukte og vannpå.	Jeg har ikke vært på en haug og gi fantastisk plan og prob-lemet er kjempe flink.

TABLE 5.13: An example which compares some of the original text (input) with some of the generated text (output) after training char-, and word-RNN to write unique deviations.

Looking at the results from Table 5.12 and 5.13 as well as Figure 5.3 and 5.4 it seems like the char-RNN is bad at handling small amounts of data to generate unique outputs that makes sense. From this point the tests will therefore be done using only the word-RNN algorithm.

For the next tests we will try to use the data, as explained in Section 5.1.3, to create an unique suggestion for a measure.

RNN size	**Number of layers**	**Sequence length**	**Batch size**	**Valid**
200	2	10	10	no
200	2	3	3	no
100	2	10	10	no
70	2	3	3	no
55	2	3	3	no
55	2	1	1	no
50	2	5	5	no
50	2	4	4	no
50	2	3	3	no
50	2	2	2	no
50	1	5	5	no
50	1	4	4	no
50	1	3	3	no
50	1	2	2	no
50	1	1	1	no
30	2	3	3	no
30	2	3	3	no
25	2	4	4	no
25	2	3	3	no
20	2	1	1	no
10	2	3	3	no
10	2	2	2	no
10	2	1	1	no

TABLE 5.15: The parameters used for training with the Shakespeare and deviation data set using the char-RNN.

Nr	**Corresponding suggestion for a measure from Table 5.9**
1	Det er veldig viktig å bestille riktig rekkverk på første forsøk. Det kan ta tid å bestille på nytt.
2	Sendt klage og Faktura til Vik.
3	Viktig å passe på at det blir bestilt riktig type rør, ettersom det kan da ta lang tid å få bestilt nye.
4	Det er ikke meningen at ansatte skal straffes for sånne årsaker. Det vil bli tatt opp på neste styremøte.
5	Dette er noe de som registrerer må være oppmerksom på. Siden dette er et gjenntakende problem må vi kanskje se på om vi må endre rutinene for registrering av brurekkverk. Dette er mye mer spesielt og stiller helt andre krav til tilpasning enn vanlig autovern.

TABLE 5.14: The input and output of Deviation 2 from Table 5.7.

In Table 5.14 is corresponding suggestions for a measure for the deviations in Table 5.9. The total amount of data, in this case is 627 bytes. That is, more than 15 000 times less data than in the Shakespeare data set. t

Overfitting	**Underfitting**
Det vil bli tatt opp på neste styremøte. Dette er noe de som registrerer må være oppmerksom på. Siden dette er et gjenntakende	bestille på Sendt klage klage klage klage klage klage klage klage klage Vik til Vik til

TABLE 5.16: An example of overfitting and underfitting the 627 bytes of data.

Data used	Valid
1KB	no
5KB	no
10KB	no
20KB	no
50KB	yes

TABLE 5.17: A list over different amount of data sets used for generation, and if it is enough to be able to generate reasonable output.

Generated output with the 50KB Shakespeare data
PROSPERO: Thou dost, and then and, For the very son in the purpose hurried tell thou camest first, I have not Lucentio. **GREMIO:** Ay, and the purpose hurried tell I will not shift my bush; and houses Cursed thou have each together capable

TABLE 5.18: The output from the word-RNN when the input is a 50KB chunk of the Shakespeare data set.

Table 5.15 shows, in the first four columns, the parameters tested, while the fifth column represents says if it is makes sense or not, i.e. if it gives a valid output. Looking at the output when training using the parameters in Table 5.15 and the 627 bytes, the empirical results turns out like in Table 5.16 every time. That is, either it is overfitted, or it gets underfitted.

Using $< 1KB$ inn an RNN clearly is too little data. It is therefore interesting to see how much data is needed to be able to get a valid output. For the following tests, I will take chunks from the Shakespeare collection, to see if how little data it is able to create similar data without copying the text. I will use empirical tests to try to find the best parameters for the short texts.

Table 5.17 shows that the minimum[7] need of data to be able to generate unique text which makes some sort of sense. As seen in Table 5.18 the text does not make much sense, but a lot of Shakespeare's texts are difficult to understand anyway, so I decided to validate this output.

[7]It might be possible to use even less data, with some other parameters by someone more experienced with RNNs.

6 Discussion

The results from Chapter 5 are somewhat divided. Looking at the proof of concept of the NB classifier it is able to find the most similar deviation, even though it has to find the most similar deviation, by searching through 2000 classes. The proof of concept shows that even if the probability is low, it is able to find the expected output.

There is a problem regarding the negatives in a sentence. I used an approach which combines the *not*-word with the following word to be able to divide the typical example as explained in Section 5.1.1. Using this approach is a good idea for all sentences which does not end with the Norwegian word for "not". Furthermore, in Section 5.1.2, the results show that because of the *not*-word, the algorithm becomes more uncertain regarding the most similar output. The algorithm does not, however, change its mind. If there were a lot more data, the issues mentioned above gets somewhat dismissed. The *not*-word technique would work better as well as it would be easier to find the most similar deviation. If there were more data, then maybe a higher level of n-gram models should be tested, even though Peng and Shuurmans [30] (see Chapter 4) shows with their empirical values that there were no significant increase in accuracy using n-gram models with $n > 2$.

In Section 5.1.3, about the validation of the classifier, it is evident to see that there is too little data to find similar deviations based on random suggestions. Out of five random deviations, created by an unbiased person, only one of them found a similar deviation which is subject to using its corresponding suggestion for a measure. The deviation who scored highest out of the five, however (55.8% at that), has no correlation with the input. This is most likely due to the weak link in the *bag of words* model. That is, the order of words does not matter. Because of this assumption, the MNB compares the number of times a word appears in a sentence and checks it with all the classes to find the most fitting. This gives MNB a tendency of choosing the classes with most text.

Both the char-RNN and word-RNN works well with the Shakespeare data set, and are able to generate unique Shakespeare data. At first, I was surprised at how much better the word-RNN was compared to the char-RNN in writing Shakespeare text. But then I realized that because word-RNN uses words as sequences, it does not give any rubbish as output (meaning that the phrase is not making sense). The char-RNN is much more of a subject to making rubbish, as it needs to generate its words based on the characters in a text. Looking at the results (in the text generation) from the training with the Shakespeare and deviation data set, I decided to use only the word-RNN because it gave the best results. In addition, my intuition says that using char-RNN on data sets < 1 KB, is a bad idea. Furthermore, I tried generating a new suggestion for a measure based on the five which corresponded to the five most similar deviation regarding the given input. The data used were only 627 bytes, which intuitively was too little data, and some experiments approved the validity of my intuition.

With further testing, it seems like approximately 50KB is the least amount of data needed to use for training a text generating RNN.

7 Conclusion

This thesis consist of empirical testing regarding the use of, mainly, Multinomial Naïve Bayes text classification with many classes, and text generation with only a small data set using a Recurrent Neural Network. Looking at the results in Chapter 6, we can see that even if the probability is low for the output in the MNB classifier, the expected output is found, and the probability distribution is as I foresaw in my hypothesis.

RNN, more specifically, word-RNN feels almost like magic with the correct tuning of the essential parameters. With big enough data set, it can generate its own unique output texts, based on the input. When the data set becomes smaller, the word-RNN struggles with the training phase, ending up with overfitting or underfitting.

This thesis shows, with empirical results, how great NB is for categorizing, even with many classes. Moreover, it shows that combining just a few suggestions for a measure, is insufficient amount of data for the word-RNN to be able to create a original and consistant suggestion for a measure.

Bibliography

[1] J. D. M. Rennie, "Improving multi-class text classification with naive bayes," September 2001.

[2] K. Lang, "Newsweeder: Learning to filter netnews," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[3] M. Webster. (2017) Merriam webster dictionary. [Online]. Available: https://www.merriam-webster.com/dictionary/classification

[4] (2015) Text classification: Step 1 of 5, data preparation. [Online]. Available: https://gallery.cortanaintelligence.com/Experiment/Text-Classification-Step-1-of-5-data-preparation-3

[5] F. Sebastini, "Machine learning in automated text categorization," 2002.

[6] J. E. Oracle, *Oracle Data Mining Concepts*, release 1 (11.1) ed., Oracle, May 2008.

[7] J. V. Stone, *Bayes' Rule: A Tutorial Introduction to Bayesian Analysis.* Sebtel Press, 2013.

[8] D. M. Lane *et al.*, "Introduction to statistics," pp. 208–210.

[9] C. Stergiou and D. Siganos, "Neural networks." [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[10] B. Voytek. (2013, May) Are there really as many neurons in the human brain as stars in the milky way? [Online]. Available: https://www.nature.com/scitable/blog/brain-metrics/are_there_really_as_many

[11] D. Shiffman, *The Nature of Code.* Free Software Foundation, 2012.

[12] (2016) Perceptrons - the most basic form of a neural network. [Online]. Available: https://appliedgo.net/perceptron/

[13] (2010) Role of bias in neural networks. [Online]. Available: http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

[14] Artificial intelligens - neural networks. [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm

[15] S. R. Kishan Mehrotra, Chilukuri K. Mohan, *Elements of Artificial Neural Networks.* MIT Press, 1997.

[16] N. Sarten. (2016, March) Simple artificial neural networks with fann and c++. [Online]. Available: https://genbattle.bitbucket.io/blog/2016/03/19/Simple-Artificial-Neural-Networks-with-FANN-and-C

[17] (2017) Merriam webster. [Online]. Available: https://www.merriam-webster.com/dictionary/recurrent

[18] O.-C. G. Mehdi Ben Lazreg, Morten Goodwin and J. Radianti, "Addressing the crisis responders' needs with information abstraction from crises related tweets using recurrent neural network," 2017.

[19] A.Graves, "Supervised sequence labelling," 2012.

[20] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," 1998.

[21] S. Xu, "Bayesian naïve bayes classifiers to text classification," 2016.

[22] J. T. Jason D. M. Rennie, Lawrence Shih and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," 2003.

[23] X. D. X. H. Ge Song, Yunming Ye and S. Bie, "Short text classification: A survey," 2013.

[24] M. F. Zelikovitz, S, "Transductive learning for short-text classification problems using latent semantic indexing," in *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, 2005.

[25] Q. Pu and G.-W. Yang, "Short-text classification based on ica and lsa," in *Advances in Neural Networks - ISNN 2006*, 2006.

[26] J. M. I. BACH F, "Kernel independent component analysis," in *The Journal of Machine Learning Research*, vol. 3, 2003.

[27] L. H, "Text classification retrieval based on complex network and ica algorithm," in *Journal of Multimedia*, vol. 8, 2013.

[28] F. G. W. L. T. K. Deerwester S., Dumais S. T. and R. Harshman, "Indexing by latent semantics analysis," in *Journal of the American Society for Information Science*, vol. 41, 1990.

[29] L. D. Landaues T. K., Foltz P. W., "An introduction to latent semantic analysis," in *Discourse Processes 25*, 1998.

[30] F. Peng and D. Schuurmans, "Combining naive bayes and n-gram language models for text classification," 2003.

[31] J. M. Ilya Sutskever and G. Hinton, "Generating text with recurrent neural networks," 2011.

[32] J. Martens, "Deep learning via hessian-free optimization," 2010.

[33] H. G. Rumelhart, D.E. and R. Williams, "Learning representations by back-propagating errors," in *Nature*, 1986.

[34] P. Werbos, "Backpropagation through time: What it is and how to do it," 1990.

[35] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen. diploma thesis," Ph.D. dissertation, Technische Universitat Munchen, 1991.

[36] S. P. Bengio, Y. and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," 1994.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural Computation*, 1997.

[38] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," 2009.

[39] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication." in *Science*, 2004.

[40] A. Graves, "Generating sequences with recurrent neural networks," 2013.

[41] A. D. H. L. S. K. T. Mikolov, I. Sutskever and J. Cernocky., "Subword language modeling with neural networks," 2012.

[42] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[43] K. Doya, "Bifurcations of recurrent neural networks in gradient descent learning," *IEEE Transactions on neural networks*, vol. 1, pp. 75–80, 1993.

[44] J. Martens and I. Sutskever, “Training recurrent neural networks with hessian-free optimizaiton,” 2011.

[45] Scikit-learn *Machine learning in python*. [Online]. Available: http://scikit-learn.org/stable/index.html

[46] A. Karpathy. (2016) char-rnn. [Online]. Available: https://github.com/karpathy/char-rnn

[47] S. Kim. (2017) word-rnn-tensorflow. [Online]. Available: https://github.com/hunkim/word-rnn-tensorflow