UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Geraldine King Granada

Improving the Automatic Text Classification Algorithm of Siav, A Case Study

Master's Thesis (30 ECTS)

Supervisors:
Fabrizio Maggi, PhD (University of Tartu)
Daniele Turato (Siav)

Tartu 2018

Improving the Automatic Text Classification Algorithm of Siav, A Case Study

Abstract:

Siav is an IT service company that provides products for electronic document management, workflow management and the preservation of digital documents. One of their projects is to create an automatic text classifier suitable for use in business contexts. The primary aim of this thesis is to improve the current accuracy and confidence reliability of the text classifier using neural networks. In order to accomplish these goals, the baselined implementation is analysed and a number of approaches from linguistic processing and neural networks are proposed to address limitations in the current technology. The proposed techniques are then implemented and the performance results are compared against the existing metrics. Finally, observations are made regarding the proposed solution and its suitability for business use compared to the existing one.

Keywords:

Text classification, machine learning, natural language processing, neural networks

CERCS: P170, Computer science, numerical analysis, systems, control

Ettevõte Siav automaatse tekstiklassifikaatori täiustamine

Lühikokkuvõte:

Siav on ettevõtte, mis pakub digitaalsete dokumentide haldamise- ja säilitamise- ning töövoogude juhtimisele keskenduvaid infotehnoloogiateenuseid. Üheks firma projektiks on ärilises kontekstis kasutatava automaatse tekstiklassifitseerimise teenuse loomine. Antud lõputöö eesmärgiks on parandada praeguse klassifikaatori täpsust ja usaldusväärsust läbi tehislike närvivõrkude. Olemasolevat lahendust analüüsitakse ja selle kitsaskohtade parandamiseks pakutakse välja mitu edasiarendust, mis kasutavad lingvistilist eeltöötlemist ja tehisliki närvivõrke. Pakutud lahendused teostatakse ja nende jõudlust võrreldakse olemasoleva lahendusega. Lõpetuseks arutletakse väljapakutud lahenduse ja selle konteksti sobimise üle.

Võtmesõnad:

Teksti klassifitseerimine, masinõpe, keeletehnoloogia, tehisnärvivõrgud

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

Equations, Tables, and Figures

1 Introduction

The applications of Text Classification are extensive, including web search or web browsing, topic identification, content filtering and spam detection, among others [1]. The increasing availability and use of digital documents, and the subsequent need to organize them in a systematic way have thus also increased the prominence of document management related tasks in the field of information systems [2].

One of the simplest approaches to automated text classification is to have domain experts define rules that could be used in queries to a database of documents. These queries could then be applied in parallel instead of sequentially, solving the scale issue. However, maintaining and updating these queries based on changing requirements is also very labour-intensive, as each rule would have to be changed manually, for example whenever categories were added or removed.

The advent of machine learning has allowed an alternative option – to intuitively learn rules based on manually pre-classified examples (*supervised learning*) [2]. Using them became very advantageous, as they are easy to create and modify, use only the information given to them and can be customized very easily to suit the needs of the organization. Furthermore, these automatic text classifiers often have accuracy comparable to those defined by human experts and are much cheaper to maintain because they do not need constant upkeep [2].

However, automated text classification with machine learning came with its own challenges. Before a classifier can interpret the text, it must first be transformed, through an indexing procedure, into a mapped representation of its content [2]. How the text is represented greatly influences the overall result of the text classification [3], thus the choice of representation must also be carefully considered. The easiest way to represent the text is to make each unique word a separate feature, but different words may mean the same thing (*synonyms*), or conversely, words with the same spelling may also have different meanings (*polysemous words*) [3]. As a result, one of the main areas of research in natural language processing and machine learning is in the development of methods to extract more meaningful *feature representations* from the text.

Following the feature representation, a second challenge is in tackling the problem of the high dimensionality of text. Texts could be as short as a phrase, and as long as a thousand-page novel, and as such, could consist of thousands, or even tens or hundreds of thousands of unique features [4]. This makes the computational cost for the classifiers incredibly high, as each feature would naturally be one input variable for them [4]. In order to ameliorate this problem, a number of techniques are performed, including but not limited to *linguistic processing* and *feature selection*.

One ongoing project in Siav is to develop one such automatic text classifier suitable for use in business contexts. In order to accomplish this aim, a bag-of-words (BOW) model combined with *Term Frequency-Inverse Document Frequency* (TF-IDF) was used for feature representation and selection. The resulting vectors were then used as the input of various machine learning algorithms, and their individual performance was compared with their performance as an ensemble. The results of Siav's approach had so far been unsatisfactory for business use, thus they were interested in trying some new approaches in order to improve their text classifier's performance. Two areas of particular interest to the company were in using additional linguistic processing techniques and neural networks for text classification, thus the focus of the study was on the use of the aforementioned approaches.

Using the current approach as a baseline, a review of the state of the art was performed for linguistic processing and text classification, including approaches in feature representation and selection. Based on the results, a combination of approaches was then identified, described in further detail in Section 3. Following the selection, proposed changes were then implemented, and the results collected and evaluated against the results of the baseline approach.

Goal and Problem Statement

The aim of this thesis is to improve the performance of the current text classifier through the use of state of the art approaches, with a particular focus on the use of additional linguistic processing techniques and neural networks. The research questions for this study were defined as follows:

> RQ1: What other linguistic processing techniques can we use to improve the baseline's performance?
>
> RQ2: How does the selected technique compare to the previous one when the same classification algorithms are used?
>
> RQ3: What is the state of the art in neural networks applied to text classification?
>
> RQ4: What are the results when Neural Networks approaches are used?

In order to address these questions, a review of the state of the art was performed for both fields of research. In addition, we also examined in particular how feature representation and feature dimensionality reduction is approached in each of the different studies. The contribution of the thesis is thus an automatic text classifier that improves on the existing approaches.

2 Related Work

This section describes the state of the art for linguistic processing techniques and machine learning algorithms used in text classification. The section is further divided into four subsections: linguistic processing techniques, feature representation, feature selection, and supervised machine learning approaches to text classification.

2.1 Additional linguistic processing approaches

Pekar [5] performed a comparative study on the effect of four different linguistic processing techniques on text classification accuracy. Firstly, he compared using stemming and lemmatisation with retaining the original form of the word. Secondly, they experimented with decomposing words by their morphology – using either only their root form or by adding new features for the prefix and suffix of each word. Lastly, they attempted to reduce the feature space further by removing rare words. In general, his findings were that most of the techniques were particularly helpful in terms of improving classification accuracy, however, they were quite advantageous in terms of reducing the dimension of the feature space. In contrast, he observed that while the morphological decomposition of words allowed the feature vector to hold more semantic information, it also greatly increased the feature space and thus had a negative effect on the performance of the classifier. He noted that this last technique would be more effective if used in conjunction with a feature selection algorithm to remove affixes that do not provide useful information.

2.2 Approaches to Feature Representation

Feature representation in the context of machine learning refers to a numerical representation of an object. This representation contains important information about the object that can be used in statistical analysis. The following subsections present various forms of feature representation for text that have been used in previous studies.

2.2.1 N-grams

The most popular approach is the *bag-of-words* (BOW) model described by Sebastiani [2]. In this approach, each word is considered an individual feature. Bag-of-words models do not preserve the order which the words appear in, which makes it problematic for some applications [6]. It is possible to have two words (*bigrams*), three words (*trigrams*) or even more (*n-grams*) in one feature, however previous experiments have shown that increasing the number of words in a feature do not drastically improve the effectiveness of the model, and so n-grams of more than two to three words are not cost-effective [1, 6].

2.2.2 Word Embeddings

Bengio et. al. [8] described a method using shallow neural networks to dynamically extract the features from the text. In addition, instead of using the word itself, each word is represented as a feature vector, wherein the vector contains various aspects of the word. These vectors are popularly known as *word embeddings* and have been successfully used in various applications related to natural language processing [9], such as part-of-speech (POS) tagging, named entity recognition, parsing, as well as text classification.

Word2Vec

Mikolov et. al. (2013) [10] further developed two model architectures to generate word embeddings, collectively known as Word2Vec[1]. The first, the *Continuous-Bag-of-Words*

[1] https://code.google.com/p/word2vec

model (CBOW) attempts to predict the probability of a word based on the other words surrounding it, while the second, *Skip-gram* does the opposite: It attempts to predict the probability of the context based on the word. Words that are related to each other semantically end up being grouped closer together within the vector space. In addition, Mikolov et. al. (2013) [11] also experimented on training Word2Vec models using n-grams. Alternatively, Rui et. al [12] further build on Word2Vec by first clustering the Word2Vec vectors together using the k-means algorithm and then choosing one of the words to represent the cluster.

Word2Vec extensions

A number of different models have also been built to extend the initial Word2Vec design. Bojanowski et. al. (2017) [13] describe a model wherein each word is treated as a sum of its n-gram embeddings. This algorithm was developed to support rare words or words that were not included in the original training data since they would still be able to find similarities with other words through shared character n-grams. The following paper by Joulin et. al. (2017) [14] describes how they developed a model architecture to generate these new word embeddings, the implementation of which they call *FastText*[2]. In contrast, instead of breaking down a word into its sub-words, Le and Mikolov [15] devised a third extension of Word2Vec called *Doc2Vec* where instead of generating feature vectors per word, they instead generate paragraph vectors to represent the entire document.

2.3 Approaches to Feature Selection

Feature selection refers to the process of selecting which information about the object is the most important. This aims to accomplish two things: to reduce the time required to train a model by reducing the amount of information to process and to reduce overfitting through generalisation.

A paper by Yang and Pedersen [4] compared different feature selection algorithms used in text classification. They found that using Chi-test (X2) and Information Gain (IG) gave the best results among the five algorithms that were chosen. In addition, using Document Frequency (DF) with a threshold also gave good results, and was more efficient than the first two algorithms as its computational cost was lower. In contrast, Term Selection (TS) only worked well with smaller vocabulary sets, and Mutual Information (MI) did not perform well at all. A similar work done by Rogati and Yang [7] compared 100 variants of popular feature selection algorithms (DF, IG, Chi-test, Mutual Information & Binary Information Gain). Their results were that Chi-test based methods were consistently the best out of all the algorithm variants tested.

2.4 Supervised Machine Learning Algorithms for Text Classification

In this section, we look at different machine learning approaches that have been used in text classification in previous studies.

Yang and Liu [16] compare five different text categorization methods: Support Vector Machines (SVM), k-Nearest Neighbour (KNN), Neural Networks, Linear Least-squares Fit (LLSF) mapping and Naïve Bayes. Their results showed that while all the methods were effective when used on an unbalanced dataset where there were more negative training samples, SVMs, kNNs, and LLSF performed better than a simple neural network and Naïve Bayes.

[2] https://fasttext.cc/

Xu et. al. [17] used a modified Random Forest algorithm to classify text into different topics. Random Forest is a popular ensemble classification method that bases its predictions on the results of an ensemble of individual decision trees. However, the problem with building trees on text data is that there is a high possibility of important information being missed, resulting in "weak" trees. To solve this problem, they use term-weighting to evaluate the performance of each individual tree and then exclude weak trees from the forest. When compared to an SVM, NB and KNN classifier, their results showed that their improved random forest algorithm had superior results.

The focus of more recent research has been on the application of deep neural network models to text classification tasks. Popular models include Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) which are described in further detail in the next section.

3 Baseline Approach

This section discusses the current approach used by Siav. Figure 3-1 shows the entire process, using a collection of text as input and generating a set of predictions from several text classifiers.

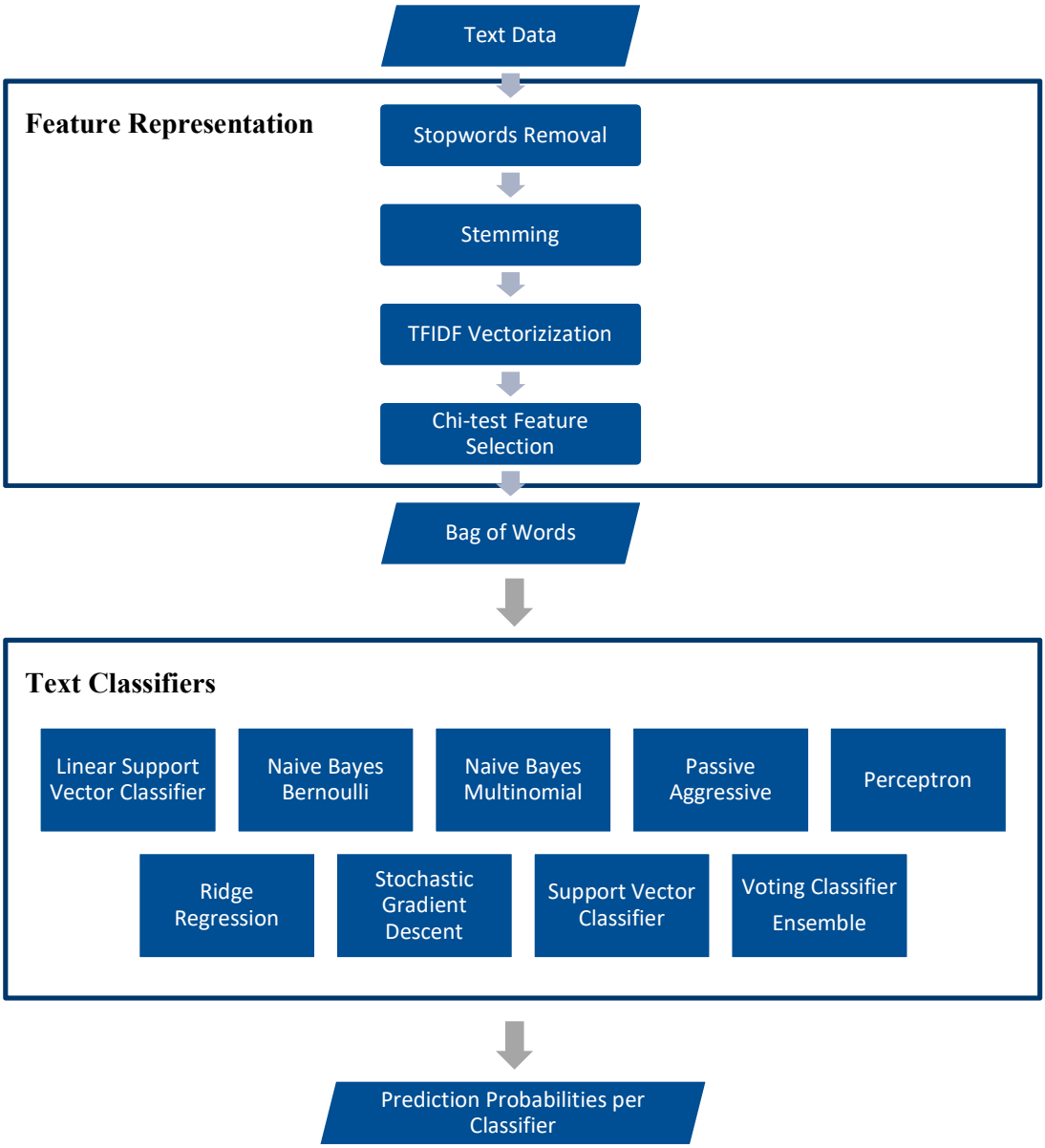

Figure 3-1 Siav's Baseline Approach

Before converting the text into vectors, Siav uses two linguistic processing techniques in order to reduce the dimensionality of features. They first remove *stopwords*, which are commonly used words and words that contain no semantic meaning, such as prepositions and articles [18, pp. 22-26]. Secondly, they use a process called stemming to reduce words into their base form by removing the ends of words. For example, 's' characters would be removed to reduce plural words to their singular form, and '-ing' and '-ed' would be

removed from verbs. The stoplist used by Siav is a collection of stopwords available from the Natural Language Toolkit[3] (NLTK), and the stemming algorithm used is an improved version of the Porter stemmer algorithm, referred to by its creator as Porter2[4] and is available as a part of the SnowballStemmer[5] collection of stemming algorithms.

Following stopwords removal and stemming, each text in the collection is then converted into Bag-of-Words (BOW) feature vectors. The TF-IDF algorithm is used to assign weights to these vectors, and the highest scoring vectors were selected using the Chi-test. A more detailed description can be found in 4.1 Feature Representation and Selection.

These weighted vectors are then used as input into several supervised machine learning-based classification models to obtain the prediction probabilities. The final prediction for each classifier is decided by which class has the highest probability. The only exception is with the Voting Classifier ensemble, which first takes the prediction probabilities of the other classifiers and generates a final prediction based on their mean prediction probabilities. We discuss these classification models in greater detail in 4.2 Supervised Machine Learning Algorithms.

[3] https://www.nltk.org/
[4] https://snowballstem.org/algorithms/english/stemmer.html
[5] https://snowballstem.org/algorithms/english/stemmer.html

4 Background

This section describes the technical background behind the methods used in the experimental implementation of the tool. Section 4.1 discusses various feature representation and selection techniques and Section 4.2 discusses different supervised machine learning algorithms used for the study.

4.1 Feature Representation and Selection

Two models for feature representation were used in the experimental scenarios: the bag-of-words model and GloVe Word Embeddings.

4.1.1 Bag-of-Words (BOW)

The most popular approach is the bag-of-words (BOW) model. In this approach, a dictionary was first created from all of the unique words in the collection of texts. Each text was then broken down into tokens and a map of all the occurrences of each word in the text was created. As an example, we present the following sentences:

Sentence 1: *The dog barks at the cat*

Sentence 2: *Cats and dogs don't get along well*

Table 4-1 Bag-of-Words (BOW) Model

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Dictionary	The	dog	barks	at	the	cat	Cats	and	dogs	don't	get	along	well
Sentence 1	1	1	1	1	1	1	0	0	0	0	0	0	0
Sentence 2	0	0	0	0	0	0	1	1	1	1	1	1	1

As can be seen in Table 4-1, retaining the sentences as they are leads to each sentence being represented as a word vector containing 12 unique features. Larger corpora would then result in larger and increasingly sparse word vectors.

Continuing our example from Table 4-1, performing stopwords removal and stemming on the sentences would result in a greatly reduced vector. The reduced vector is shown in Table 4-2.

Table 4-2 Bag-of-Words (BOW) Model with Stemming

Index	**0**	**1**	**2**	**3**	**4**	**5**
Dictionary	The	dog	bark	cat	Cat	And
Sentence 1	1	1	1	1	0	0
Sentence 2	0	1	0	0	1	1

In addition to removing stopwords and stemming, Siav used Term Frequency – Inverse Document Frequency (TF-IDF) to score feature words based on their importance to the document, and CHI^2 to retain only the 5000 most important features. TF-IDF is a combination of two statistics: the term frequency, which calculates how often a term (word) occurs within a particular text, and inverse document frequency which proportions this data against how frequent the term is throughout the entire corpora. The idea behind this algorithm is that words that are commonly found throughout the corpora are relatively useless, as they cannot be used to make the text unique, while words that are rare in a specific text could not be reliably used to generalize throughout a group of texts.

4.1.2 Word Embeddings: GloVe Embeddings

Another form of Word Embeddings called Global Vectors for Word Representation (GloVe[6]) was developed by Pennington et. al. [19]. In contrast to WordVec which uses neural networks in order to predict the words that would appear within the context of another word (or vice versa), GloVe embeddings consider how often words appear near to each other (co-occurrence counts).

Given two word vectors, the GloVe algorithm defines that their scalar product is equal to the logarithm of the frequency of their co-occurrence. This relationship is defined by Equation 4.1, where $w_i^T \tilde{w}_j$ refers to the scalar product of words i and j, and X_{ij} refers to the frequency of their co-occurrence. In addition, they add bias terms to provide a base output value.

Equation 4.1 GloVe co-occurrence

$$w_i^T \tilde{w}_j + b_i + \tilde{b}_j = \log X_{ij}$$

Furthermore, they consider that words that co-occur rarely or infrequently may be buried by the noise of frequently co-occurring words, thus they add weights to make sure that the model is able to learn from both. Words that appear together only once or rarely are weighted much more heavily than words that appear together more often, and conversely, the weight is kept small for frequently co-occurring words. Equation 4.2 defines the final model, which takes the original relationship, uses the residual sum of errors and adds a weight function $f(X_{ij})$.

Equation 4.2 Weighted Least Square Regression for GloVe

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

The weight function is further expanded into Equation 4.3, which returns a value between 0 and 1 based on the frequency x and introduces a limit so it just returns 1 if it goes beyond a defined maximum value.

[6] https://nlp.stanford.edu/projects/glove/

Equation 4.3 Weighting function for Equation 4.2

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\propto} & if\ x < x_{max} \\ 1 & otherwise \end{cases}$$

4.2 Supervised Machine Learning Algorithms for Text Classification

In 2 Related Work, we discussed different machine learning approaches to text classification that were not used in neither the baseline nor in the experimental approaches. This subsection describes approaches used in the latter.

4.2.1 Linear Support Vector Classification

Support Vectors are the data points that lie closest to the area in the hyperplane that separates positive and negative examples of a class. Linear Support Vector Classification focuses on finding the most optimal function to separate the two using the Support Vectors.

4.2.2 Bernoulli and Multinomial Naïve Bayes

Bernoulli (Bernoulli NB) and Multinomial Naïve Bayes (Multinomial NB) are variants of classifiers based on the Naïve Bayes algorithm. Naïve Bayes algorithms assume that all features are independent of each other. The distinction between the two is that Bernoulli NB only checks if a term is present or not present in a document, which severely limits its effectiveness when used with longer texts. On the other hand, Multinomial NB also takes into account the frequency of its appearance in the document.

4.2.3 Perceptron

The Perceptron algorithm is one of the oldest artificial neural networks algorithms. It can be used in both online (where the classifier is trained incrementally over time), and offline (where the classifier is trained with all the training examples at once) learning. The main idea in the perceptron algorithm is the weight vector. Every time new data is added, the current weight vector is used during classification. If the classification is correct, the weight vector is retained, but if it is incorrect, the weight vector is aggressively changed to a value that is close to the original weight vector but also satisfies the loss function or the error difference between the original and modified weight vector.

4.2.4 Passive Aggressive

Passive Aggressive algorithms are online learning algorithms based on the Perceptron algorithm. The difference is that Passive Aggressive classifiers include a regularisation parameter to keep the penalty from being too high.

4.2.5 Ridge Regression

Ridge Regression is a linear classifier algorithm, thus it attempts to find the function of the line that is best able to separate positive and negative samples within the vector space. Ridge is an improvement over the Ordinary Least Squares method or Linear Regression algorithm because it tries to reduce overfitting by adding a penalty.

4.2.6 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variant of Gradient Descent which is an algorithm used in conjunction with linear regression algorithms. For classification tasks, it focuses on

finding the line function that best separates two classes of data. It works by minimizing the cost function or error rate between the current line function and the data points. The difference between Gradient Descent and SGD is that while the former uses all the data points to find the best weight value, SGD only uses a random subset of the data to reduce the performance and time cost of training the algorithm.

4.2.7 Voting Classifier

Voting Classifier is a form of ensemble classification that chooses the best prediction out of the individual predictions of different classifiers. There are two ways in which it chooses the best prediction. The first, called 'hard' voting, selects the prediction based on the majority rule. In the case of a tie, it selects the first option. In comparison, 'soft' voting takes the argument of the maxima (argmax) of the mean confidence probabilities of the individual classifiers. This allows for the consideration of the confidence of each classifier, so predictions with overall higher confidence values would have greater weight.

4.2.8 Neural Networks and Deep Learning

Neural networks are models that were inspired by how neurons work in the human brain. Basic neural networks generally have four layers: the input layer, the weight layer wherein the input is multiplied by a weight value, the bias layer where the input is added to a bias value, and a calculation layer that evaluates the continuous-value result to produce a discrete label prediction. The aim in training the neural network is to find the weight and bias value that fits the best to the training data, or that has the smallest possible error or cost between the predicted values and the actual values. In order to do this, the neural network uses gradient descent algorithms to shift the weight and bias values incrementally until it has reached the lowest possible error value.

Deep Learning is the popularized term for large neural networks. They are called "deep" because they consist of multiple "hidden" layers in addition to the initial set. With the volume of data available today, and the increased computational power of modern computers, deep learning has been growing in popularity in machine learning, as it has a greater ability to scale in comparison to traditional machine learning algorithms.

Multilayer Perceptrons

Multilayer Perceptrons are one of the simplest forms of deep neural networks consisting of multiple layers of Perceptrons. Clark, et. al. [20] developed a binary classification model to categorize emails into spam and not spam. They used a bag-of-words model assigned with weights calculated using Information Gain (IG) and Variance (V). They compared this model with k-Nearest Neighbours, Stacking, Stumps and the Treeboost algorithm and found that their model performed well when used with IG for feature selection.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are normally used for image recognition tasks but have also been used in text classification, such as in the study done by Kim [21] on sentence classification. The author trained CNN models using different variations of word embeddings. The first scenario consisted of a CNN model with word embeddings that were randomly initialized, and modified during training. The second scenario used pre-trained Word2Vec embeddings that were trained on 100 billion words from Google News[7]. The author had expected that the second scenario would outperform the first, but they were

[7] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/view

surprised to observe that the second scenario was at least 4% more accurate across all the datasets they experimented on. This suggests that pre-training the word vectors such as done with Word2Vec is an integral component of deep learning for natural language processing.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are another type of neural network model used frequently for tasks where order and sequence of the data are important, such as in natural language processing. Nowak et. al. [22] compared different variations of RNNs on their performance for classifying short texts and sentiment classification. The authors used BOW models to represent the text, then compared Long Short Term Memory (LSTM), Bidirectional LSTM, and Gated Recurrent Unit (GRU) models to see how well they worked to classify the texts as spam or not spam. They found that LSTM and Bidirectional LSTM models performed well compared to GRUs, however, they noted that because of their simpler design, GRUs may be better suited for simpler datasets.

Combining the properties of CNN & RNNs

It is also possible to use combinations of CNNs and RNNs to build text classification models. Zhou et. al. [23] designed a neural network model (C-LSTM) consisting of a CNN layer connected to an LSTM layer and compared its performance in binary and multi-class classification to several state of the art models. For feature representation, they used Word2Vec models pre-trained on 100 billion words from Google News, similar to the work done by Kim [21]. The authors found that while there were several models that outperformed their C-LSTM model, the results they achieved with this experiment were still promising, and comparable to several of the state of the art models.

Alternatively, Lai et. al. [24] proposed a model called Recurrent Convolutional Neural Networks (RCNN). The model consists of concatenated LSTM layers and receives as input the left and right context words in addition to the one being analysed. The overall structure of the model thus resembles that of a CNN while being able to use the ability of LSTMs to take word order into consideration. Furthermore, the model used Word2Vec embeddings trained on English and Chinese language Wikipedia corpora for feature representation. The authors compared it to several popular text classification models such as Support Vector Machines (SVMs) and Logistic Regression, as well as with CNNs. The authors found that their model outperformed both the traditional methods and CNNs.

5 Contribution

In order to answer the research questions, several scenarios were designed using various linguistic processing techniques and neural network architectures. These scenarios are summed up in Table 5-1 List of Scenarios.

Table 5-1 List of Scenarios

Scenario No.	Scenario Name
Scenario 0	Baseline
Scenario 1	Baseline+LP
Scenario 2	BOW+MLP
Scenario 3	WE+CNN
Scenario 4	WE+LSTM
Scenario 5	WE+CNNLSTM
Scenario 6	WE+RCNN
Scenario 7	Voting Classifier Hard
Scenario 8	Voting Classifier Soft

Scenario 0 is Siav's current approach which we mark as our baseline, and was discussed in 3 Baseline Approach. The first and second research question deal with the effect of linguistic processing techniques, thus we designed Scenario 1 Baseline+LP, which applies the selected techniques to the baseline approach. The remaining scenarios then deal with answering the third and fourth research questions related to using neural network architectures.

The following subsections subsequently describe each scenario in greater detail.

5.1 Baseline+LP

Figure 5-1 shows the additional linguistic processing techniques added during the creation of the feature vector. We first expanded contractions such as "don't" and "won't" into their original forms (do not and will not respectively), then removed all the punctuation marks. A full list of the contractions can be found in Table 6-2 Contraction replacements. In addition, we also lemmatized each remaining word in the text after removing the stopwords.

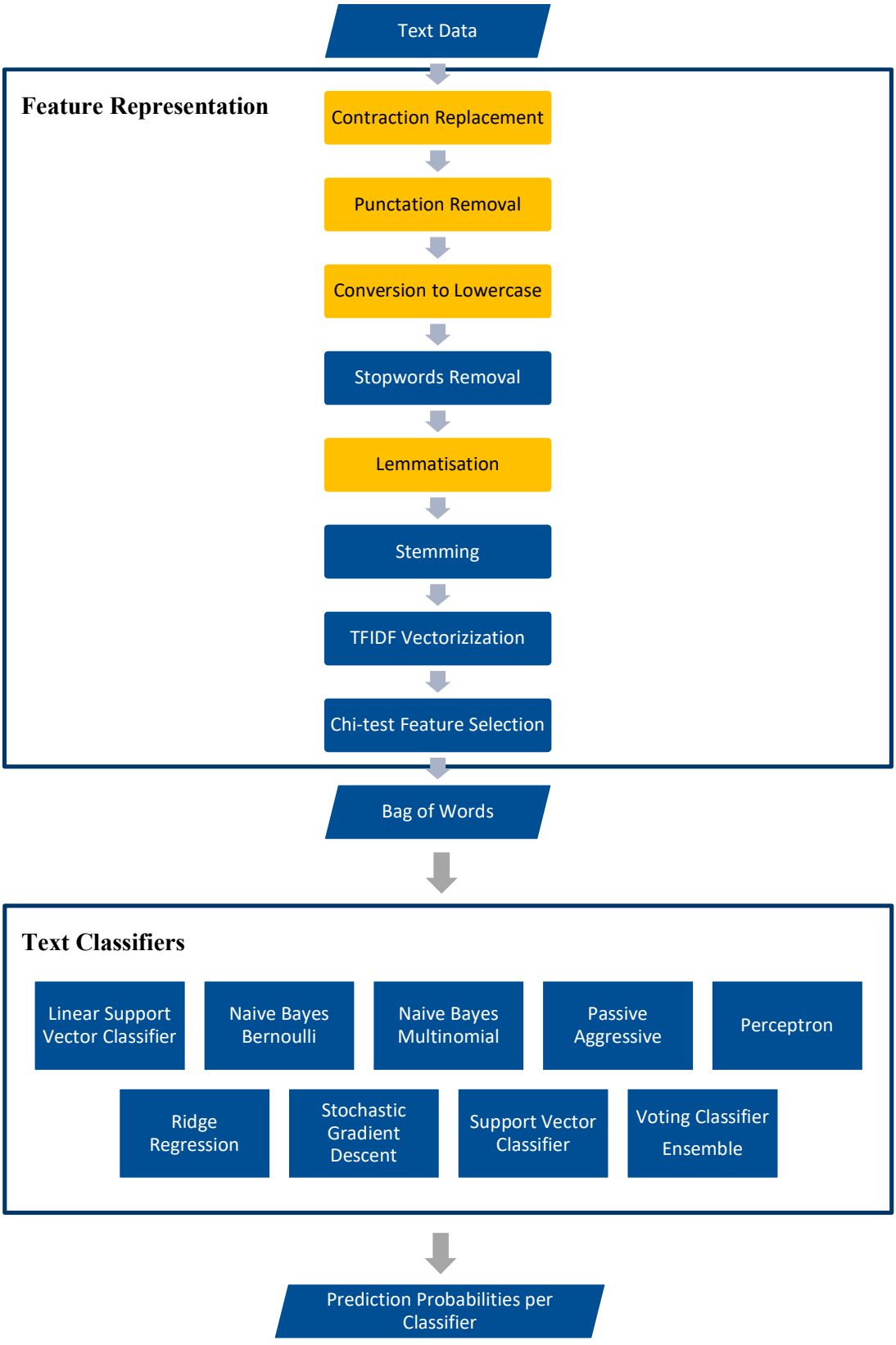

Figure 5-1 Baseline+LP

5.2 BOW+MLP

In this scenario, a Bag-of-Words model was used as feature representation together with a Multilayer Perceptron architecture. Figure 5-2 below describes the entire process.

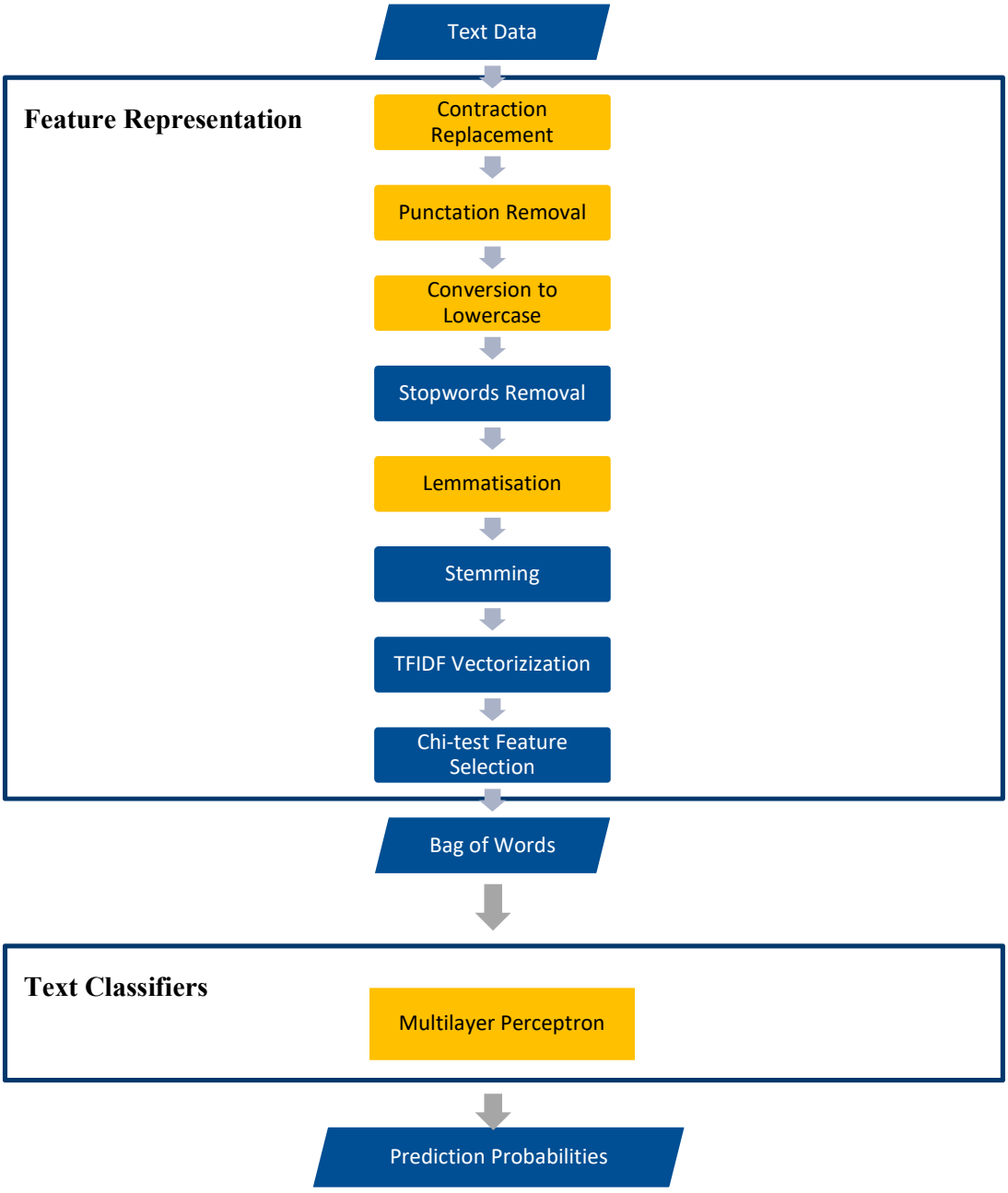

Figure 5-2 BOW+MLP

The selected linguistic processing techniques that were applied to the Baseline+LP approach were retained, however, instead of the machine learning algorithms from the Baseline approach, a standard Multilayer Perceptron model with two fully-connected (dense) layers were used. The first fully-connected layer applies a Rectified Linear Unit (ReLU) activation

function and the second applies the softmax activation function. Figure 5-3 MLP Network Architecture below describes shows the architecture design.

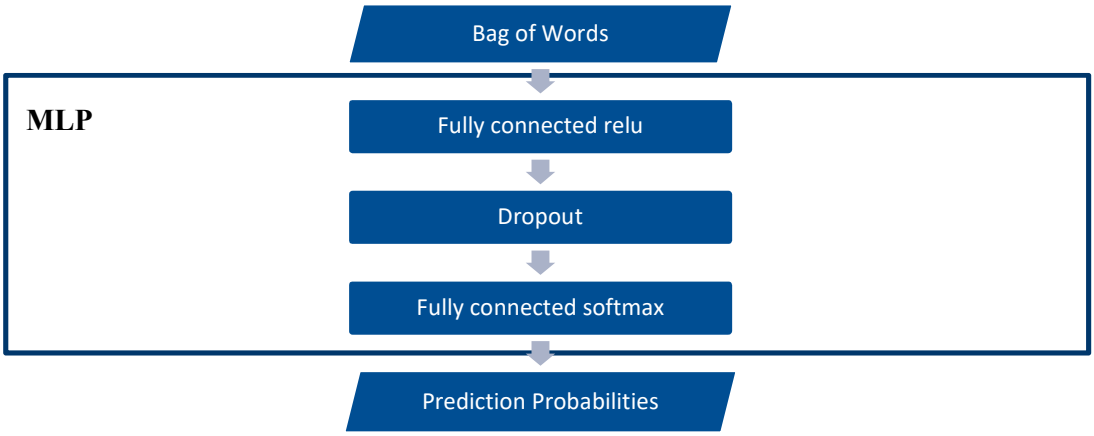

Figure 5-3 MLP Network Architecture

5.3 CNN, LSTM, CNNLSTM, and RCNN

This subsection describes the scenarios for all remaining neural network architectures. As was done with the BOW+MLP approach, the additional linguistic processing techniques were retained. However, the stemming process was removed, and instead of a bag-of-words model, GloVe Word Embeddings (WE) were used for feature representation. After lemmatisation, the remaining texts were tokenised and passed through an embedding layer that assigns weights based on pre-trained embeddings. Previous studies have remarked that word embeddings are more effective when they have been trained on large volumes of data, thus it was decided to use 300-dimension pre-trained GloVe embeddings[8] made available by Stanford's Natural Language Processing group. These embeddings contain 6 billion tokens and have a vocabulary size of 400 thousand words.

[8] http://nlp.stanford.edu/data/glove.6B.zip

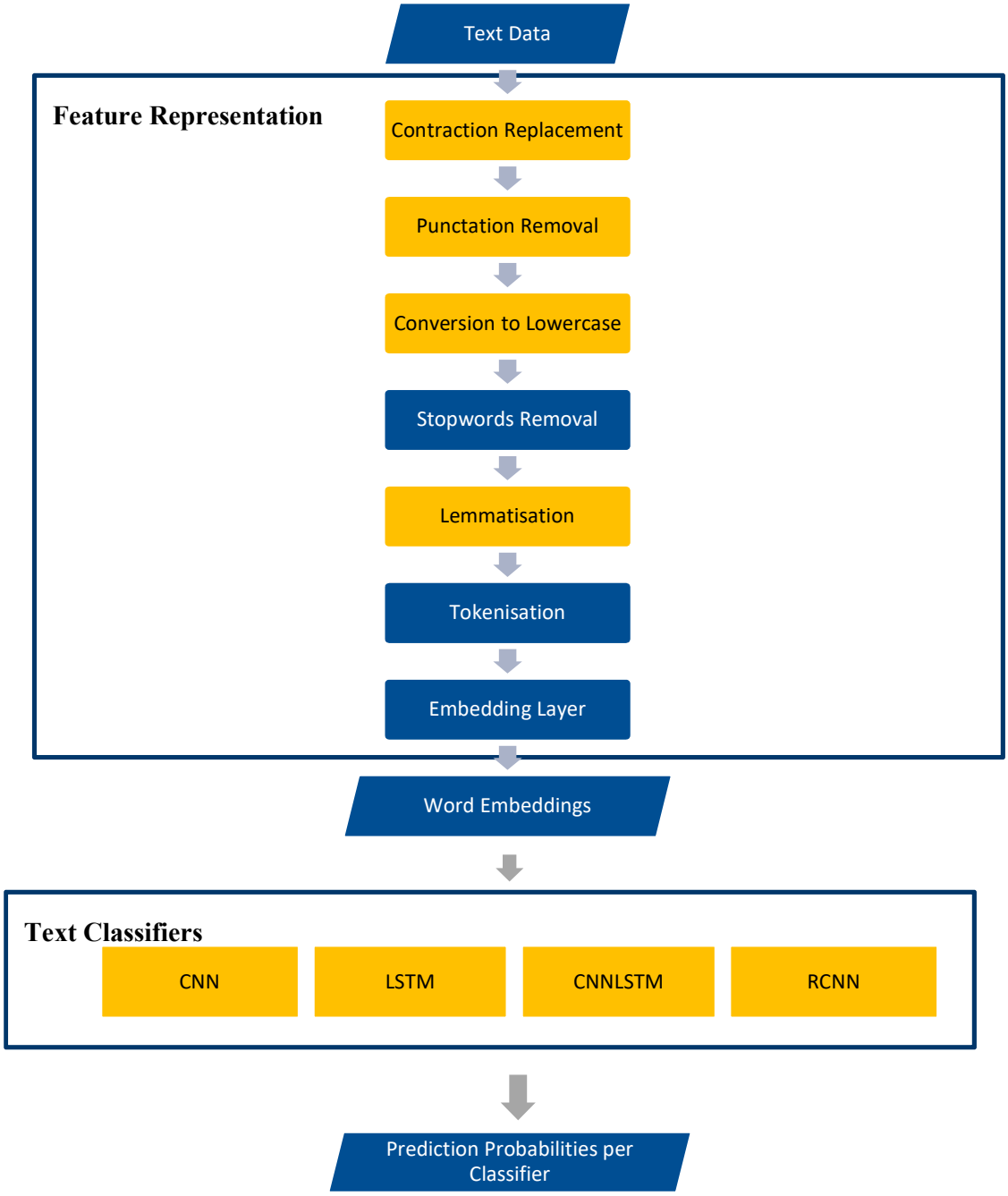

Figure 5-4 WE + CNN, LSTM, CNNLSTM, and RCNN

Four text classification models were then designed using CNNs, LSTMs, and combinations of both using the word embeddings as input. These models were adapted and modified from existing research. In addition, a number of regularisation techniques and modifications to the hyperparameters were applied in order to improve upon the results of the baseline approach. The following subsections further describe each of the selected models.

WE+CNN (Convolutional Neural Network)

Figure 5-5 describes the CNN architecture adapted from the model described by Kim [21]. The tokenized text was passed into an embedding layer that was initialized with the weights

from the pre-trained GloVe embeddings, in contrast to the pre-trained Word2Vec embeddings introduced by Kim. Secondly, a dropout layer was added after the embedding layer and before the final activation layer to help prevent overfitting. The model consists of three 1-dimensional convolution layers, each followed by a max pooling layer. The final layer is a fully-connected layer that takes the output features and applies the softmax function to generate prediction probabilities for each of the classes.

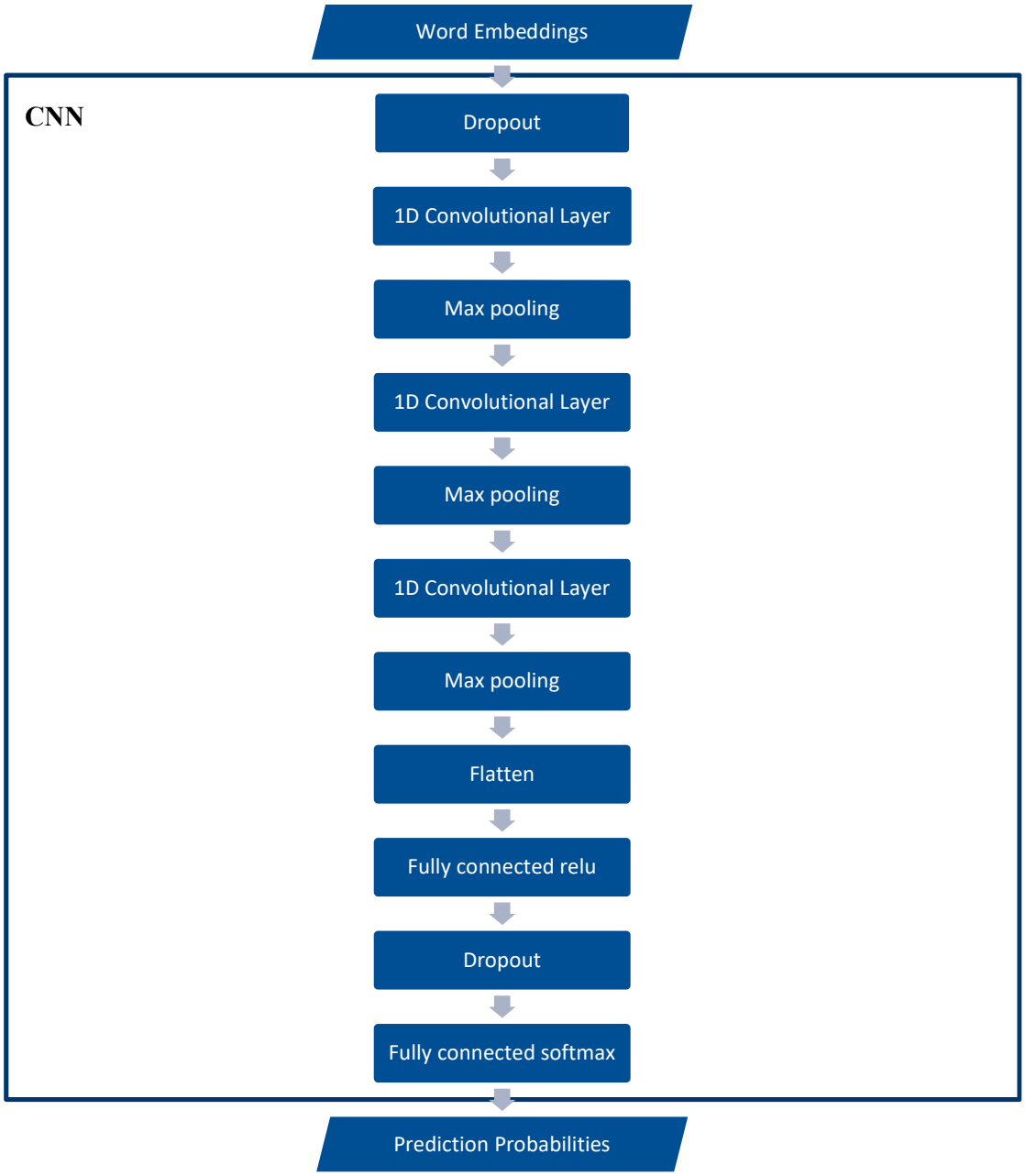

Figure 5-5 WE+CNN Architecture

WE+LSTM (Long Short-term Memory)

Recurrent neural networks are often used for sequence-related tasks where the order in which the data appears is important. However, one of their limitations is that they are biased towards the most recent data they see, so for example, when analysing a long text, they heavily favour the last few words that they received. LSTM networks are a variation of recurrent neural networks that were developed to ameliorate this issue.

For this thesis, a standard LSTM model architecture similar to the one described by Nowak et. al. [22] was implemented, as shown below in Figure 5-6. The tokenized text was passed into the same embedding layer used for the CNN described in Figure 5-5, and then into a dropout layer to prevent overfitting. The input values were then fed into a single LSTM layer. Finally, the softmax function was applied in order to obtain the prediction probabilities.

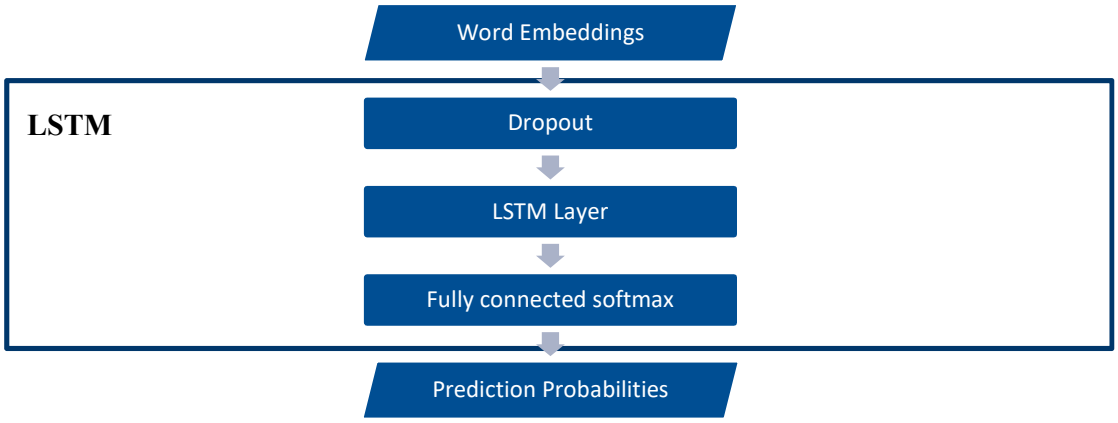

Figure 5-6 WE+LSTM Network Architecture

WE+CNN-LSTM

The model architecture in Figure 5-7 CNNLSTM Network Architecture describes how a CNN and an LSTM model are combined in order to see if more complex models are able to perform better than simple ones. The architecture described in the figure below was an adaptation of the first model described by Zhou et. al. [23] where they connected a convolutional layer to an LSTM layer without any max pooling in between. They also stated that they apply dropout either before the convolutional layer or after the LSTM layer, but they do not say which configuration they ended up using. In addition, they state that they add a L2 regularisation layer after the final softmax layer, which we omit in our design.

As was done in the separate CNN and LSTM models, pre-trained GloVe embeddings were used in the embedding layer. In contrast to the original model, the max pooling layer between the convolutional and LSTM layer was retained. Furthermore, a dropout layer was added after the embedding layer, and the L2 regularisation layer was omitted.

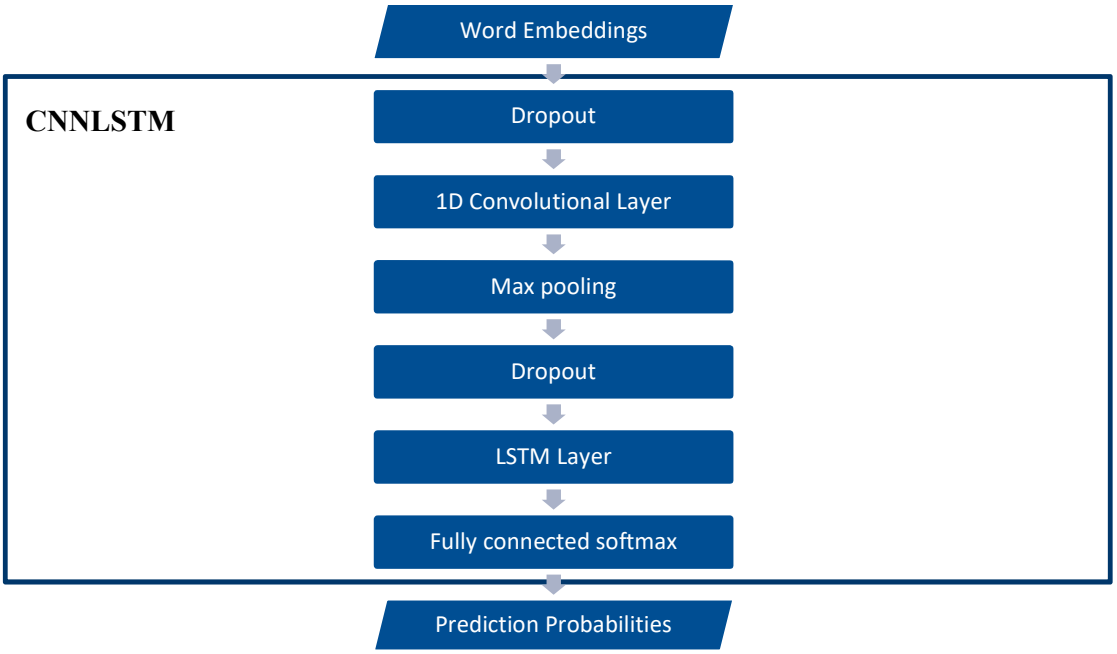

Figure 5-7 CNNLSTM Network Architecture

RCNN (Recurrent Convolutional Neural Network)

The final model used was based on the RCNN described by Lai et. al. [24]. This model accepts three input vectors, based on the original word, and the words on either side of it called the left context and the right context respectively. This produces a window effect similar to how a CNN works, where it looks at each text in small defined batches. All input vectors were then passed to an embedding layer that used GloVe pre-trained embeddings instead of the Word2Vec the authors used in their approach. The left and right context were then each fed into two LSTM layers and their output was concatenated with the original vector. Following this, the output of the concatenated layer was fed into a fully connected layer with a ReLU activation function and another fully connected layer with a softmax to get the probabilities for each class. The network architecture is shown below in Figure 5-8 Recurrent Convolution Neural Network Architecture.

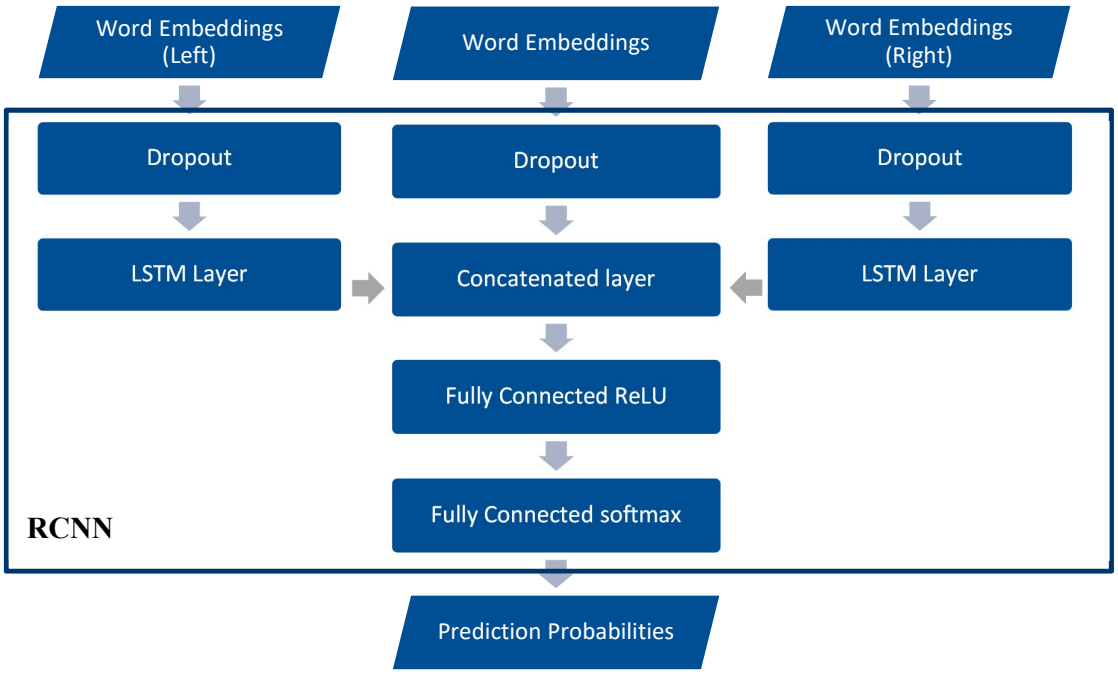

Figure 5-8 Recurrent Convolution Neural Network Architecture

5.4 Ensemble Prediction (Voting Classifier)

As was done in the baseline approach, we used the Voting Classifier algorithm to obtain the final predictions based on the predictions from each of the individual classifiers. For this study, we obtained the final predictions using both "hard" and "soft" voting. Hard voting takes the predicted label from each classifier and selects the mode or the most frequently voted class. For soft voting, we took the mean probability for each class and then calculated the maximum mean probability value in order to get the final prediction label.

6 Experiment Design

In this section, we describe the parameters of the experiment and how it was conducted. In the following subsections, we define the different technologies used in the experiment, the selected dataset, the control variables in the experiment task, and the response variables we will use to evaluate the classifier's performance.

6.1 Tools and Libraries

The section below describes the languages and libraries used in the experiment, including the environment used to run the classifier training and testing.

Languages and Libraries

All the classifier scripts were written in version 3.6.0 of the Python programming language. For linguistic processing, we used the *Natural Language Toolkit*[9] which provides a collection of stopwords in various languages in addition to text-processing libraries for stemming and lemmatisation. The main libraries used to create the feature vectors and baseline classifiers are *scikit-learn*[10], a popular library for machine learning algorithms, *numpy*[11], a library for handling matrices and arrays, and *pandas*[12], a library for data structures and data manipulation. In addition, we used *keras*[13], a high-level Application Programming Interface (API) for implementing the models of the neural networks. This was run on top of *tensorflow*[14], an open source framework for machine learning.

To create the visualizations in this document, we used *matplotlib*[15] and *seaborn*[16].

Environment

The MLP model was executed on an Intel® Core™ i5-6200U CPU @ 2.30Ghz 2.40 Ghz with 16.0 GB of RAM. All other models were executed on NVIDIA Tesla P100 GPU Accelerators for PCIe-based servers with 16GB of VRAM.

Callbacks

Keras provides a series of functions called *callbacks*[17] that can be applied to the model while it is training. The first callback function we used was *Tensorboard*, which collects defined metrics during training, saves them as event log files and provides a graphical interface to display these metrics in a web browser. The second callback was *ModelCheckpoint*, which monitors a defined metric during training and saves the model either after every epoch or only if the metric has improved. We selected the second option and set it to save the model only if validation accuracy has improved in order to obtain the best model, and prevent overtraining of the model. The last callback was *ReduceLROnPlateau* which reduces the learning rate of the model when the defined metric starts improving. In this case, we assigned it to monitor the validation loss and reduce the learning rate if the validation loss stops decreasing.

[9] https://www.nltk.org/
[10] http://scikit-learn.org/stable/
[11] http://www.numpy.org/
[12] https://pandas.pydata.org/
[13] https://keras.io/
[14] https://www.tensorflow.org/
[15] https://matplotlib.org/
[16] https://seaborn.pydata.org/
[17] https://keras.io/callbacks/

Source code references

A number of tutorials and other open source projects were used as references during the implementation of the selected text classifiers. A tutorial created by the Keras Team on how to use GloVe word embeddings with Keras CNNs Networks [24] was used as a reference for creating the embedding layers and how to use convolutional layers using the Keras API. In addition, the RCNN model implementation was based partially on an implementation done by Alcorn [25]. All materials referenced were released under the MIT License.

6.2 Selected Dataset

The dataset used by Siav in their baseline approach is the *20NewsGroups*[18] dataset. It consists of a collection of 18000 texts from newsgroups divided into 20 different topics or classes. Some of the topics are quite similar to each other, which makes the texts difficult for classifier systems to categorize. In addition, the texts contain spelling and grammatical mistakes. This attribute makes it an appropriate and popular choice for gauging the performance of text classifiers.

The original text had two sections: the header and the actual text. The header contained meta about the text such as the sender's email address, the date sent and which newsgroup it belonged to. A sample text from the original dataset can be found in Appendix I.

Additional modifications were applied to the original dataset by Siav. Firstly, the headers were removed as they contained the newsgroup category of the text, and retaining them would make it too easy for the classifiers. An example of the remaining text can be found in Figure 6-1.

> I am looking for an inexpensive motorcycle, nothing fancy, have to be able to do all maintinence my self. looking in the <$400 range.
>
> if you can help me out, GREAT!, please reply by e-mail.

Figure 6-1 Sample of modified 20Newsgroups text

In addition, three classes were omitted in order to obtain a balanced dataset: *alt.atheism*, *comp.graphics* and *comp.sys.ibm.pc.hardware*, leaving 17 classes and a total of 17390 samples.

Table 6-1 Distribution of text samples for each fold

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Total
comp.os.ms-windows.misc	201	200	200	200	200	1001
comp.sys.ibm.pc.hardware	203	203	203	203	203	1015
comp.sys.mac.hardware	201	201	200	200	200	1002
comp.windows.x	201	201	200	200	200	1002
misc.forsale	202	202	202	202	202	1010
rec.autos	201	201	201	201	200	1004

[18] http://qwone.com/~jason/20Newsgroups/

rec.motorcycles	200	200	200	200	200	1000
rec.sport.baseball	200	200	200	200	200	1000
rec.sport.hockey	200	200	200	200	200	1000
sci.crypt	200	200	200	200	200	1000
sci.electronics	202	202	202	201	201	1008
sci.space	201	200	200	200	200	1001
soc.religion.christian	200	200	199	199	199	997
talk.politics.guns	211	211	210	210	210	1052
talk.politics.mideast	201	201	201	200	200	1003
talk.politics.misc	235	234	234	234	234	1171
talk.religion.misc	225	225	225	225	224	1124
Total	3484	3481	3477	3475	3473	**17390**

The dataset was also originally divided into two subsets, one meant for training the classifier, and one for testing. However, Siav decided to use k-fold cross-validation, so both subsets were merged into one and then split into five folds. Figure 6-2 shows the distribution of samples used as test data for each fold. For each fold, one-fifth of the data is reserved as the test data while the remaining samples are used for training.

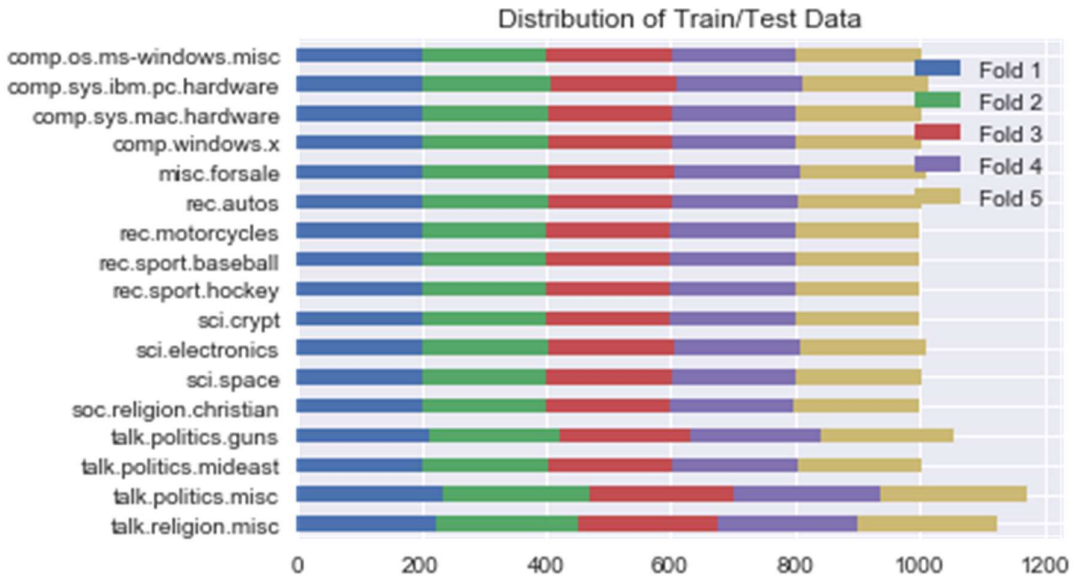

Figure 6-2 Distribution of train/test samples

The same dataset was used in both the baseline and the experimental setup in order compare the results between the two.

6.3 Experiment Settings

This section describes the technical description of the different neural network classifiers, including a description of the input and output data, and the hyperparameters for each neural network layer.

Linguistic Processing

The table below lists the common contractions that were replaced sequentially during the pre-processing stage. After these are performed, all non-alphabetic characters were replaced with spaces. Following this step, stopwords were removed from the text, and then each remaining word was lemmatized and changed to its lowercase form.

Table 6-2 Contraction replacements

Contraction	Replacement
“‘s”	“ s”
“’ve”	“ have”
“won’t”	“will not”
“can’t”	“can not”
“n’t”	“ not”
“’re”	“are”
“’d”	“ had”
“’ll”	“will”

Input/ Output

Input for Baseline+LP approach and MLP

Following additional linguistic processing, each text in the corpus was stemmed using the SnowballStemmer[19] library provided by the Natural Language Group. The dataset was then transformed into feature vectors using the TfIdfVectorizer[20] library provided by scikit-learn.

In addition, we configured the Chi-test feature selection algorithm to return the top 20000 features from the original 5000 in the baseline approach. These parameters are described in the table below.

Table 6-3 Input vector parameters for Baseline+LP approach and BOW+MLP

Baseline+LP	Feature vector length: 20000 Chi-test selection parameter: 5000
BOW+MLP	Feature vector length: 20000 Chi-test selection parameter: 20000

[19] https://www.nltk.org/_modules/nltk/stem/snowball.html
[20] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Input for CNN, LSTM, CNNLSTM, and RCNN

Each text in the dataset was first tokenized into a list of indices that represent a unique word in the corpus. The tokenizer only used the 20000 most frequent words and converted the rest into zeros. After tokenization, each list was then transformed into a uniform length of 1000 tokens.

This input was then passed into an Embedding layer that assigned weights to each index, initially using the weights from the pre-trained GloVe embeddings. These weights were updated automatically during training.

The output of the classifier was a list of prediction probabilities for each of the 17 classes, detailing the likelihood of the text belonging to a particular class. The final prediction was the result of applying the argmax function to the probabilities.

Hyperparameters

The table below lists the specific hyperparameters for the different layers in each neural network model. These hyperparameters were assigned a random value at first, and then these were replaced with varying values to see if there was an improvement in the results.

Table 6-4 Neural Network Hyperparameters

BOW+MLP	Fully-Connected	units: 128 Activation: ReLU
	Fully-Connected	units: 17 Activation: Softmax
WE+CNN	Dropout	0.5
	Conv1D	No. of filters: 200 Kernel size: 5 Activation: re ReLU
	Max pooling	5
	Fully-Connected	units: 17 Activation: Softmax
WE+LSTM	Dropout	0.5
	LSTM	units: 256 dropout: 0.5 recurrent_dropout: 0.5
	Fully-Connected	units: 17

		Activation: Softmax
WE+CNNLSTM	Dropout	0.5
	Conv1D	No. of filters: 200 Filter size: 5 Activation: ReLU
	Max pooling	5
	LSTM	units: 256 dropout: 0.5 recurrent_dropout: 0.5
	Fully-Connected	units: 17 Activation: Softmax
WE+RCNN	Dropout	0.5
	LSTM	units: 256 dropout: 0.5 recurrent_dropout: 0.5
	Fully-Connected	units: 128 Activation: ReLU
	Fully-Connected	units: 17 Activation: Softmax

Dropout layers help prevent overfitting by randomly setting a percentage of input values to zero while training. A dropout rate of 0.5 (or 50%) was used for all dropout layers. For the CNN and CNNLSTM models, a combination of a 1-dimensional convolutional layer (Conv1D in keras) and a max pooling layer was selected. For the models with LSTM layers, the same configurations of 256 units and dropout rates of 0.5 were used for both the input and the recurrent state (recurrent dropout). The final fully-connected layers for all models use 17 units (equal to the total number of classes) and the softmax activation function.

Compilation and Training parameters

Before starting the training sequence, several learning parameters needed to be configured. As the task is multi-class classification, *categorical cross-entropy* (also called *log loss*) was used for the loss function and *categorical accuracy* was used for the metric. The selected optimiser was *rmsprop* with a learning rate of 0.003. This learning rate was selected by

starting with the default values listed in the Keras API documentation[21], and then gradually increasing them until the performance stopped improving.

For training parameters, a batch size of 128 was selected and each model was trained for 50 epochs. To select the batch size and the number of epochs, the settings were initialized with random values and experiments were conducted to increase the values until the performance started deteriorating. In the case of the number of epochs, we specifically chose a consistent value for all the neural networks to provide an equal environment for comparison between them and applying the ModelCheckpoint callback provided sufficient protection against overfitting.

6.4 Response Variables

We defined several standard metrics used to evaluate classification models: accuracy, loss, precision, recall, and the F1-score. These are computed by taking the final prediction values (predicted labels), the correct values (true labels), and calculating the contingency table values listed in Table 6-5 Contingency Table for Classifier Evaluation.

Table 6-5 Contingency Table for Classifier Evaluation

	Correctly Classified	**Incorrectly Classified**
Positive Samples	True Positives (TP)	False Negatives (FN)
Negative Samples	True Negatives (TN)	False Positives (FP)

Accuracy

Accuracy is measured by getting the ratio of how many samples were correctly classified to the total number of sample,s as shown in Equation 6.1 :

Equation 6.1 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The higher the accuracy, the better – this means that the classifier did not make a lot of mistakes. However, accuracy by itself is not a good measure of the classifier's performance, as a skewed or imbalanced distribution of samples can still result in high accuracy even though the classifier is actually not performing very well.

Loss

Loss in this context refers to the logarithmic loss or cross-entropy loss. It is used in classification models that return probability values rather than just a single prediction. It complements accuracy wherein the aim is to minimize the log loss while maximizing accuracy. The log loss increases when the model assigns high confidence or probability values to incorrect classes, thus providing insight into the uncertainty in the model's prediction. For binary classification, log loss is calculated using the formula in Equation 6.2

[21] https://keras.io/optimizers/#rmsprop

where y refers to a binary value of whether the class is the correct prediction or not, and prefers to the prediction probability returned by the classifier. The mathematical constant *e* is used as its base, making this the natural logarithm *ln*. The negative log is also used in order to return positive values.

Equation 6.2 Log loss for binary classification[22]

$$Log\ loss = -(y \ln p) + (1 - y) \ln(1 - p)$$

In multi-class classification, the log loss is calculated by taking the sum of the log loss for each class (*c*) and for each sample or observation (*o*). The complete formula is described in Equation 6.3.

Equation 6.3 Log loss for multi-class classification[23]

$$Log\ loss = -\sum_{c=1}^{M} y_{o,c} \ln(p_{o,c})$$

Precision, Recall, and F1-score

Precision measures how much of the results for a class are correctly classified into that label compared to actually belonging to another label, while recall measures how many of the positive samples were correctly classified compared the total number of all samples for that class. A model with only a small percentage of correctly classified samples will result in high precision and low recall, while a model that returns a large percentage of results but with a lot of misclassified samples will result in low precision but high recall. Precision is calculated using the formula in Equation 6.4 and Recall is calculated using the formula in Equation 6.5.

Equation 6.4 Precision

$$Precision = \frac{TP}{TP + FP}$$

Equation 6.5 Recall

$$Recall = \frac{TP}{TP + FN}$$

In addition, we compute the F1-score, or the weighted average or the harmonic mean of precision and recall. The equation is detailed in Equation 6.6.

Equation 6.6 F1-score

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

[22] http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy
[23] http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy

In practice, it is generally impossible to obtain a perfect score in any of the measures, thus the aim is usually to obtain a balance between the different metrics depending on business or technical requirements.

7 Results and Evaluation

In this section, we discuss the results obtained from training the different experimental models. In addition, we compare these results to the baseline. All of the classification output is available in Comma-Separated Values (CSV) format in Google Drive[24].

7.1 Analysis of Baseline+LP Performance

Firstly, the results of applying additional linguistic processing to the text before classification were compared to the baseline.

Precision, Recall, and F1-score

Figure 7-1 shows the precision, recall, and F1-score from both approaches. In addition, it also shows the support column, which contains the number of test samples.

Table 7-1 Comparison of Baseline with and without processing

	Baseline			**Baseline+LP**			
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Support
LinearSVC	0.8428	0.8413	0.8417	0.8416	0.8404	0.8407	17390
NBBernoulli	0.8154	0.8070	0.8081	0.8159	0.8057	0.8074	17390
NBMultinomial	0.8449	0.8449	0.8448	0.8452	0.8452	0.8451	17390
Passive Aggressive	0.8443	0.8434	0.8437	0.8434	0.8424	0.8428	17390
Perceptron	0.8392	0.8378	0.8384	0.8395	0.8381	0.8386	17390
Ridge	0.8570	0.8575	0.8570	0.8561	0.8566	0.8561	17390
SGD	0.8211	0.8149	0.8165	0.8182	0.8134	0.8144	17390
SVM	0.8549	0.8542	0.8543	0.8551	0.8545	0.8545	17390
Voting Classifier	0.8676	0.8673	0.8670	0.8664	0.8656	0.8654	17390

As using the default 2 decimal places produced almost the same values, it was decided to increase the number of decimal places to 4 to see the differences more clearly. The additional linguistic processing does not seem to have improved the results of the classification. In fact, overall, the results seem to have deteriorated with the addition, though not by a significant percentage.

The best classifier for both scenarios is still the Voting Classifier with more than 0.86 across all scores, while the worst classifier is the Naïve Bayes Bernoulli with 0.81.

We now select the best classifiers for both Baseline and Baseline+LP and compare their results by class. Table 7-2 shows the classification performance of the Voting Classifier using the baseline approach, while

Table 7-3 shows the performance using the Baseline+LP approach. Each row in the tables contains the precision, recall, and F1-score of how well the classifier was able to categorize texts in a particular class in comparison to the other classes. In addition, each row also shows

[24] https://drive.google.com/drive/folders/16gzQfbNGBtVUR4ngNScM6r0COgk_NwS8

the support or the number of texts for each class. The final row of each table contains the average scores and the total support, which matches the corresponding data in Table 7-1. This data is shown for future comparison with the experimental scenarios.

Table 7-2 Baseline with Voting Classifier Classification Report by Class

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
comp.os.ms-windows.misc	0.8236	0.8302	0.8269	1001
comp.sys.ibm.pc.hardware	0.7998	0.7951	0.7974	1015
comp.sys.mac.hardware	0.8790	0.8703	0.8746	1002
comp.windows.x	0.8942	0.9022	0.8982	1002
misc.forsale	0.7491	0.8515	0.7970	1010
rec.autos	0.8903	0.8974	0.8938	1004
rec.motorcycles	0.9492	0.9350	0.9421	1000
rec.sport.baseball	0.9597	0.9520	0.9558	1000
rec.sport.hockey	0.9797	0.9650	0.9723	1000
sci.crypt	0.9606	0.9270	0.9435	1000
sci.electronics	0.8310	0.8393	0.8351	1008
sci.space	0.9390	0.9381	0.9385	1001
soc.religion.christian	0.8990	0.9378	0.9180	997
talk.politics.guns	0.8357	0.8508	0.8431	1052
talk.politics.mideast	0.8744	0.9023	0.8881	1003
talk.politics.misc	0.7199	0.6849	0.7020	1171
talk.religion.misc	0.8028	0.7171	0.7575	1124
avetotal	0.8676	0.8673	0.8670	17390

Table 7-3 Baseline+LP with Voting Classifier Classification Report by Class

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
comp.os.ms-windows.misc	0.8213	0.8262	0.8237	1001
comp.sys.ibm.pc.hardware	0.7950	0.7833	0.7891	1015
comp.sys.mac.hardware	0.8768	0.8663	0.8715	1002
comp.windows.x	0.8824	0.8912	0.8868	1002
misc.forsale	0.7404	0.8614	0.7963	1010
rec.autos	0.8922	0.8904	0.8913	1004
rec.motorcycles	0.9491	0.9320	0.9405	1000
rec.sport.baseball	0.9625	0.9500	0.9562	1000
rec.sport.hockey	0.9816	0.9620	0.9717	1000
sci.crypt	0.9627	0.9300	0.9461	1000
sci.electronics	0.8288	0.8403	0.8345	1008
sci.space	0.9430	0.9421	0.9425	1001
soc.religion.christian	0.9016	0.9378	0.9194	997
talk.politics.guns	0.8254	0.8451	0.8351	1052
talk.politics.mideast	0.8721	0.9043	0.8879	1003
talk.politics.misc	0.7177	0.6883	0.7027	1171
talk.religion.misc	0.8121	0.7153	0.7606	1124

avetotal	0.8664	0.8656	0.8654	17390

The classes with the best results are the ones with very specific topics such as *rec.sport.baseball* and *rec.sport.hockey*. In contrast, the classes the classifier is having trouble with are the classes that have *misc*. The general use of the word misc is a shortened form of miscellaneous, which means a collection of diverse or unrelated elements, which may explain why it was difficult to categorize items from these classes.

The confusion matrix in Figure 7-1 also suggests that this may be the case, as for example with the class *comp.os.ms-windows.misc*, 63 samples from the test set were misclassified as belonging to *comp.sys.ibm.pc.hardware*. In addition, the class with the lowest scores is talk.politics.misc, where 300 samples were misclassified into *talk.politics.guns*, *talk.politics.mideast* and *talk.religion.misc*.

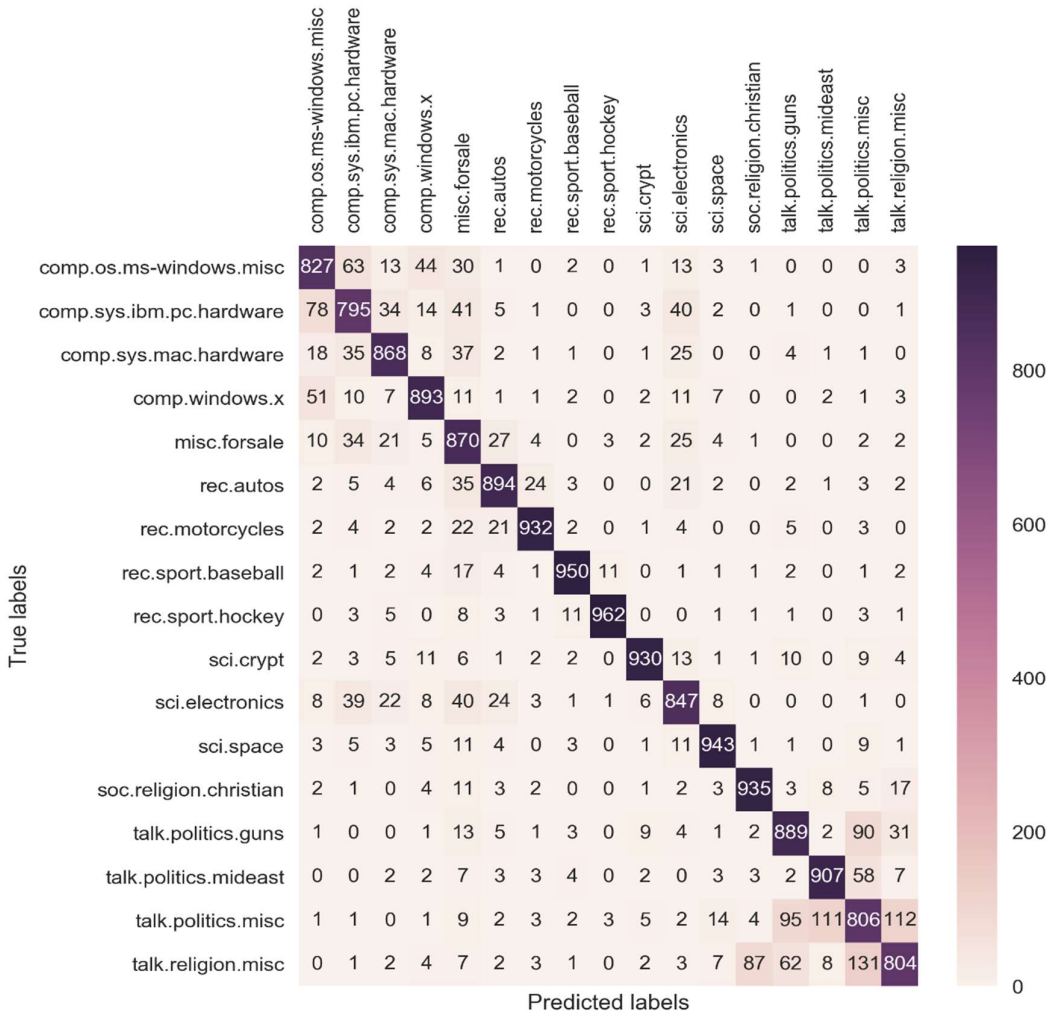

Figure 7-1 Baseline+LP with Voting Classifier Confusion Matrix

Accuracy and Loss

We then compare the accuracy and log loss between the Baseline and the Baseline+LP approaches. Table 7-4 shows the results of both approaches.

Table 7-4 Baseline+LP Accuracy and Loss

	Baseline		**Baseline+LP**	
	Accuracy	Loss	Accuracy	Loss
LinearSVC	0.8413	0.6253	0.8404	0.6276
NBBernoulli	0.8070	0.8605	0.8057	0.8678
NBMultinomial	0.8449	0.5883	0.8452	0.5898
Passive Aggressive	0.8434	0.6068	0.8424	0.6065
Perceptron	0.8378	0.6517	0.8381	0.6536
Ridge	0.8575	0.5390	0.8566	0.5410
SGD	0.8149	0.7940	0.8134	0.7958
SVM	0.8542	0.4753	0.8545	0.4764
Voting Classifier	0.8673	0.5550	0.8656	0.5570

We can see from comparing both sets of results that the accuracy and loss more or less stay the same, though Baseline's results were slightly better. This is consistent with the pattern we had seen so far with precision and recall.

The Voting Classifier clearly outperforms the rest of the classifiers with regards to accuracy, followed by SVM and Ridge which have close scores. Naïve Bayes Bernoulli (NBBernoulli) has the worst performance with only 0.8070.

The log loss varies much more significantly in comparison to the accuracy values. One of the aims of a machine learning classifier is to minimize loss, thus the closer to 0 it is, the better the result. While the Voting Classifier has the best accuracy, the Support Vector Machines classifier has the best loss value.

7.2 Analysis of Neural Network Performance

This section reports the results of the different neural network approaches. Table 7-5 Results from Neural Network models shows a summary of the classification scores of each classifier.

Table 7-5 Results from Neural Network models

Neural Networks				
	Precision	Recall	F1-Score	Support
MLP	0.8837	0.8822	0.8828	17390
CNN	0.8416	0.8422	0.8413	17390
LSTM	0.8780	0.8783	0.8780	17390
CNNLSTM	0.8666	0.8658	0.8656	17390
RCNN	0.8658	0.8665	0.8660	17390
Voting Classifier Hard	0.8885	0.8889	0.8884	17346[a]
Voting Classifier Soft	0.8875	0.8880	0.8875	17390
Voting Classifier Soft (Edited)	0.8890	0.8896	0.8891	17346 [a]

Among the individual classifiers, the Multilayer Perceptron (MLP) performed the best, closely followed by the LSTM network. CNNLSTM and RCNN had similar results, while the CNN architecture had the worst performance, with much lower scores than the other networks.

Overall, the best prediction results were the ones obtained using the Voting Classifier, which was similar to our findings in 7.1 Analysis of Baseline+LP . 44 samples were removed from the test data of Voting Classifier Hard because each classifier voted for a different sample, thus there was no majority. Rather than arbitrarily select one of the predictions as a placeholder, it was decided to just remove the samples to remove any bias from the classification.

In light of this, two sets of results are provided for Voting Classifier Soft [a]. The first set of results contained predictions from all 17390 rows in the dataset, while the second set of results (Voting Classifier Edited) had rows removed corresponding to the rows in Voting Classifier Hard.

Voting Classifier Hard had slightly better scores than the first set of results from Voting Classifier Soft but did not have as good scores than the second. When analysing the prediction probability of the rows that were removed, it was observed that 34 out of the 44 of the predictions were incorrect, which may have contributed to the improvement of the results when they were removed. In addition, all the probabilities were less than 0.30, which suggests that the confidence of the prediction was not very high.

The subsections below provide further analysis of the results from each individual classifier.

7.2.1 BOW+MLP

The design of the BOW+MLP architecture was different from the rest of the neural network architectures as it used different feature representation (BoW instead of Word Embeddings). Using the original settings as the Baseline and Baseline+LP approaches resulted in subpar performance (*Precision*: 0.8055, *Recall*: 0.7970, *F1-Score*: 0.7999), so we experimented with increasing the different parameters manually to see if it could be improved. It was observed that increasing the length of the sequence did not affect the performance of the classifier but doubling the feature selection parameter from 5000 to 10000 resulted in an improvement of 7% (*Precision*: 0.8796, *Recall*: 0.8784, *F1-Score*: 0.8789). Using the maximum value of 20000 improved these scores further, as can be seen in the following subsections.

Precision, Recall, and F1-score

The first five rows in Table 7-6 show the resulting precision, recall, and F1-score after each fold.

Table 7-6 BOW+MLP Classification Results by Fold

	Precision	Recall	F1-Score	Support
Fold 1	0.8929	0.8915	0.8919	3484
Fold 2	0.8866	0.8839	0.8849	3481
Fold 3	0.8799	0.8789	0.8792	3477
Fold 4	0.8799	0.8774	0.8782	3475
Fold 5	0.8810	0.8794	0.8800	3473

Standard Deviation σ	*0.0057*	*0.0057*	*0.0057*	-
Merged	0.8837	0.8822	0.8828	17390

We can see that the scores are quite close to each other, which is proven by the value of the calculated standard deviation score. The final row shows the classifier performance using the results from all five folds, which also closely reflects the mean of the individual results from each fold.

Table 7-7 shows the performance of the MLP classifier for each class. As can be expected based on the overall performance of the classifier, the results for each class are much higher than the results from the baseline. Three notable exceptions from this observation are *comp.sys.ibm.pc.hardware*, *comp.sys.mac.hardware* and *rec.sport.hockey*, which did not improve noticeably compared to the other classes, albeit in the case of the *rec.sport.hockey*, the results are already the highest among all the classes.

Table 7-7 BOW+MLP (Merged) Classification Results

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
comp.os.ms-windows.misc	0.8485	0.8452	0.8468	1001
comp.sys.ibm.pc.hardware	0.7946	0.8158	0.8051	1015
comp.sys.mac.hardware	0.8799	0.8703	0.8751	1002
comp.windows.x	0.9350	0.9192	0.9270	1002
misc.forsale	0.8064	0.8703	0.8371	1010
rec.autos	0.9182	0.9163	0.9172	1004
rec.motorcycles	0.9664	0.9480	0.9571	1000
rec.sport.baseball	0.9775	0.9560	0.9666	1000
rec.sport.hockey	0.9738	0.9660	0.9699	1000
sci.crypt	0.9712	0.9450	0.9579	1000
sci.electronics	0.8539	0.8700	0.8619	1008
sci.space	0.9723	0.9461	0.9590	1001
soc.religion.christian	0.9432	0.9328	0.9380	997
talk.politics.guns	0.8548	0.8565	0.8557	1052
talk.politics.mideast	0.8909	0.8873	0.8891	1003
talk.politics.misc	0.7016	0.7267	0.7139	1171
talk.religion.misc	0.7821	0.7696	0.7758	1124
avetotal	0.8837	0.8822	0.8828	17390

Furthermore, the confusion matrix in Figure 7-2 shows that the distribution of misclassified samples is consistently clustered around the same set of classes. This suggests that these samples may belong to more than one class if reassessed.

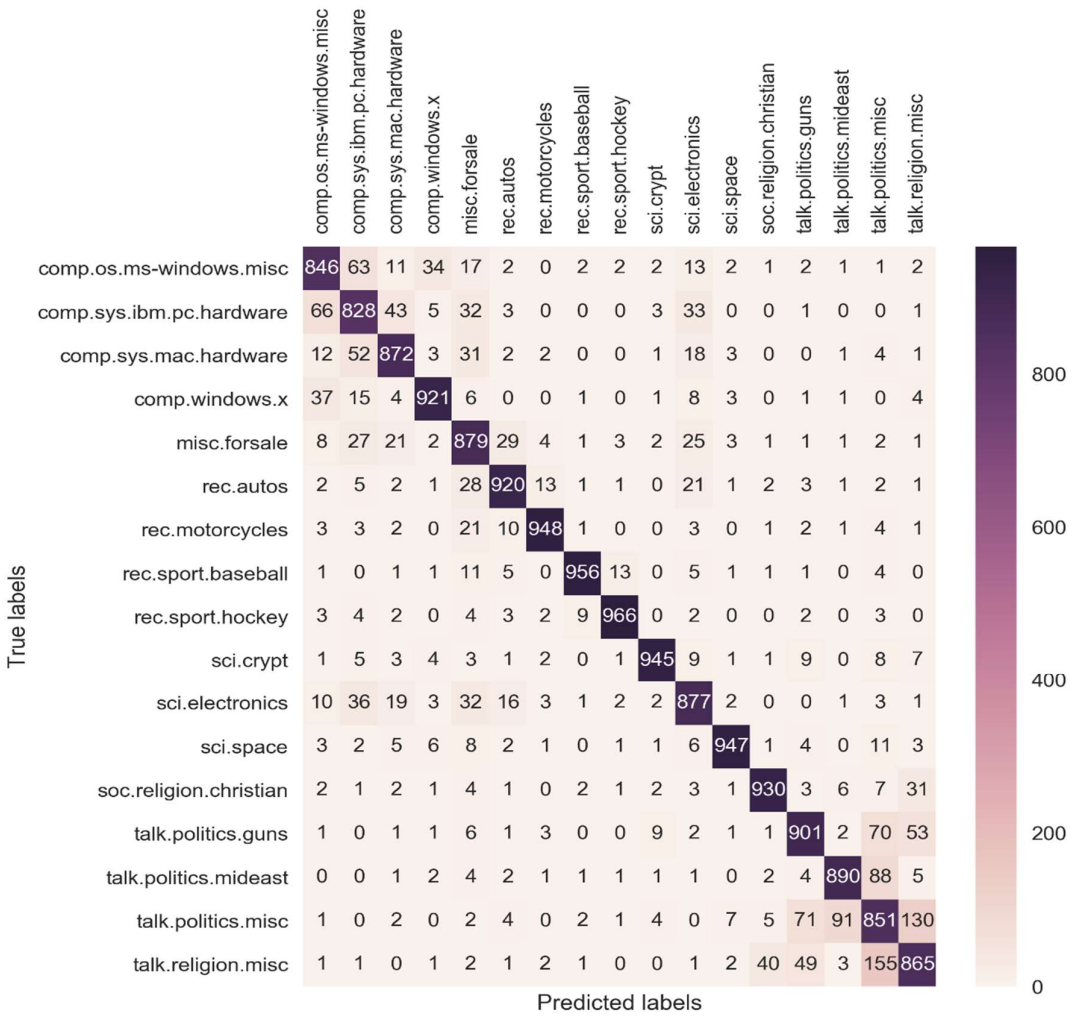

Figure 7-2 BOW+MLP (Merged) Confusion Matrix

Accuracy and Loss

Finally, we analyse the accuracy and loss of the neural network. Figure 7-3 below shows the training and validation accuracy and loss per epoch for each fold.

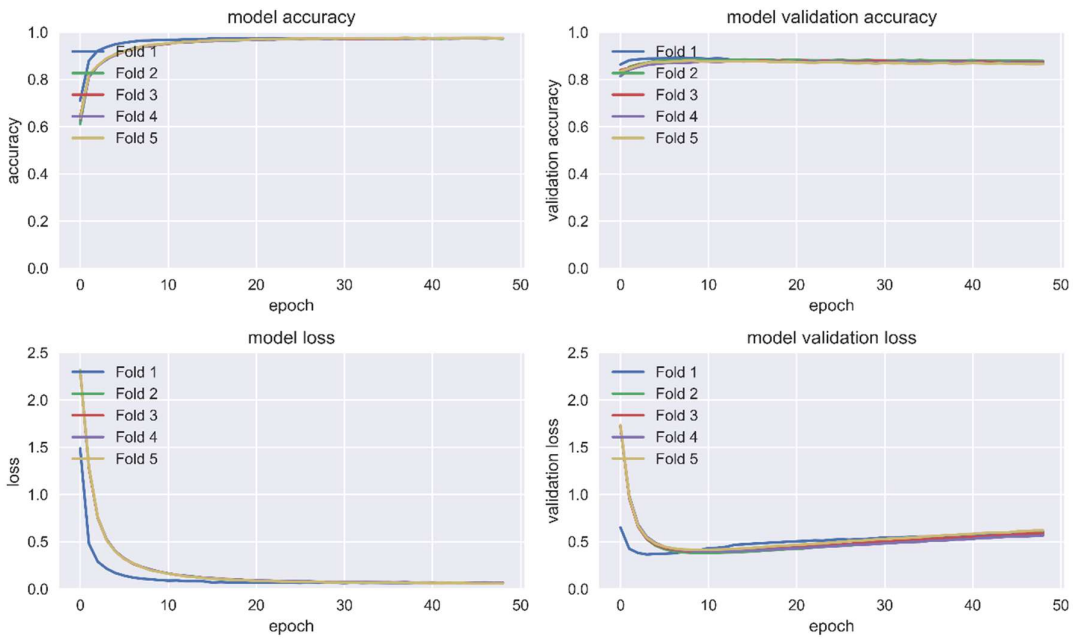

Figure 7-3 BOW+MLP Training History

With the exception of the first fold, we can see that the remaining folds generally follow the same training pattern. The highest values were reached early in the training phase, with only incremental improvements after the first ten epochs. The validation accuracy already starts with a very high score, and only improves incrementally after that.

Despite applying Dropout regularisation between the fully-connected layers, we can still see over-fitting on the training dataset. The difference between the training and validation accuracy, and between the loss and validation loss are consistent after the first ten epochs, although we also observe a slight but consistent increase in the validation loss. As the ModelCheckpoint callback was used, all epochs that did not result in improved validation accuracy were disregarded. Table 7-8 shows the best model saved for each fold using ModelCheckpoint, together with the training and validation accuracy and loss for each epoch.

Table 7-8 Best Models per Fold for BOW+MLP

Fold	Model Checkpoint	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	Epoch 07	0.9602	0.8915	0.1194	0.3769
2	Epoch 15	0.9614	0.8839	0.1183	0.3917
3	Epoch 22	0.9684	0.8789	0.0865	0.4477
4	Epoch 17	0.9645	0.8774	0.1058	0.4100
5	Epoch 09	0.9465	0.8794	0.1960	0.4143

We can see that the validation accuracy and the validation loss both remain within a consistent range in each fold, which suggests that we can be reasonably confident with the scores. BOW+MLP has the best results among all the experimental scenarios, as aside from having the highest precision, recall, and F1-score, it also outperformed the rest by having the highest accuracy values and the lowest loss values.

7.2.2 WE+CNN

This subsection analyses the WE+CNN scenario. This classifier used word embeddings as input into a convolution neural network.

Precision, Recall, and F1-score

Table 7-9 shows the results for each fold. The standard deviation is low, meaning that the results remain consistent throughout each fold. In comparison to the other neural network classifiers, this architecture was in the average range with only 0.84 across all scores.

Table 7-9 WE+CNN Classification Results by Fold

	Precision	**Recall**	**F1-Score**	**Support**
Fold 1	0.8332	0.8315	0.8302	3484
Fold 2	0.8469	0.8446	0.8447	3481
Fold 3	0.8419	0.8401	0.8402	3477
Fold 4	0.8523	0.8518	0.8509	3475
Fold 5	0.8416	0.8431	0.8406	3473
Standard Deviation σ	*0.0071*	*0.0074*	*0.0076*	-
Merged	0.8416	0.8422	0.8413	17390

We then investigate the classification results by class. As can be expected from our previous observations from Table 7-9, the scores per class are also much lower compared to the previous scenarios. One exception is its precision score for *comp.sys.ibm.pc.hardware*, but as its recall conversely greatly decreased, we can still say that the merged score for this class is still low.

Table 7-10 WE+CNN (Merged) Classification Results

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
comp.os.ms-windows.misc	0.7717	0.8611	0.8140	1001
comp.sys.ibm.pc.hardware	0.7939	0.7399	0.7659	1015
comp.sys.mac.hardware	0.8354	0.8613	0.8482	1002
comp.windows.x	0.9114	0.9032	0.9073	1002
misc.forsale	0.7874	0.7772	0.7823	1010
rec.autos	0.8508	0.8974	0.8735	1004
rec.motorcycles	0.9304	0.9090	0.9196	1000
rec.sport.baseball	0.9423	0.9480	0.9452	1000
rec.sport.hockey	0.9633	0.9460	0.9546	1000
sci.crypt	0.9327	0.9150	0.9238	1000

sci.electronics	0.8156	0.7986	0.8070	1008
sci.space	0.9177	0.9241	0.9209	1001
soc.religion.christian	0.8447	0.9278	0.8843	997
talk.politics.guns	0.7837	0.8127	0.7979	1052
talk.politics.mideast	0.8633	0.8624	0.8628	1003
talk.politics.misc	0.6652	0.6345	0.6495	1171
talk.religion.misc	0.7437	0.6610	0.7000	1124
avetotal	0.8416	0.8422	0.8413	17390

We further analyse the results from each class using the confusion matrix in Figure 7-4. We can see that aside from the labels starting with *comp-* and *talk-* which were already established in previous results as being generally difficult to classify, the WE+CNN classifier also had trouble classifying the labels starting with *sci-*.

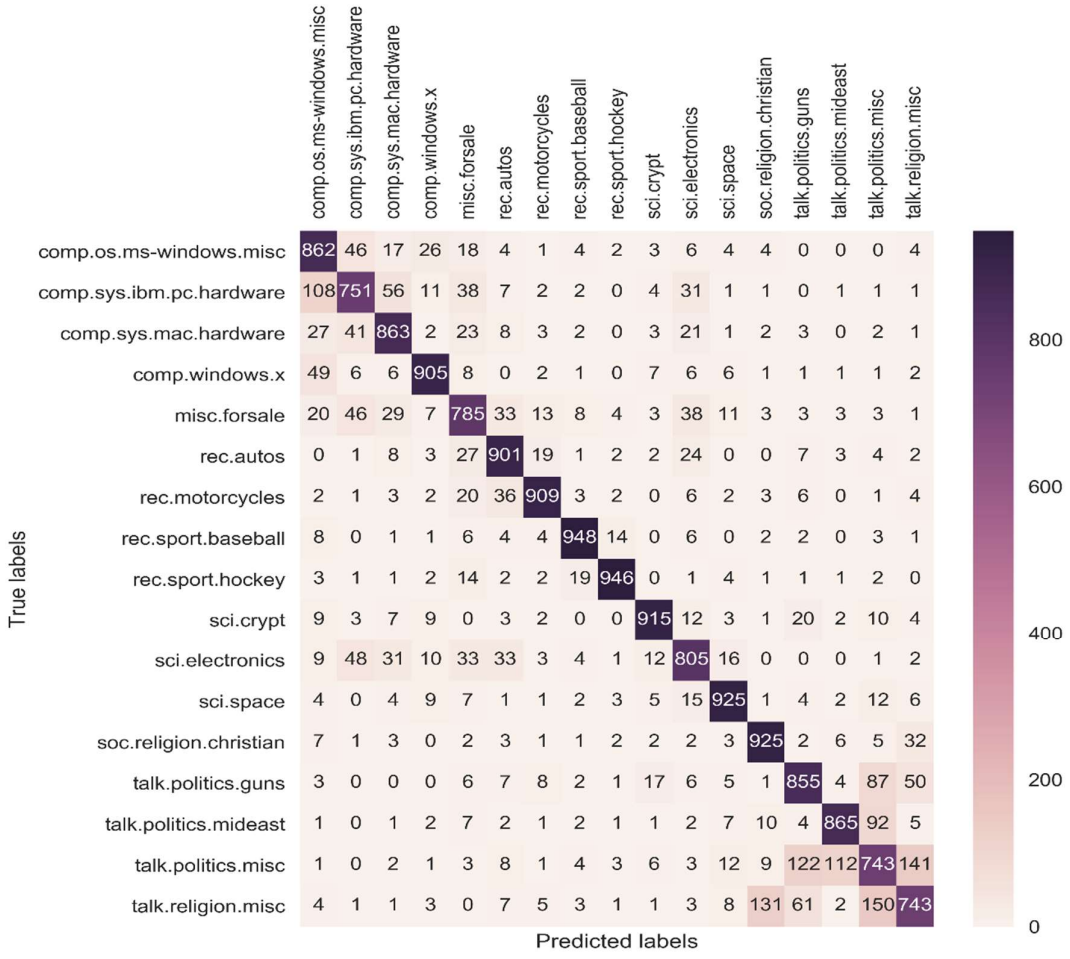

Figure 7-4 WE+CNN (Merged) Confusion Matrix

While in the previous scenarios the misclassifications mostly clustered around similar topics (which could be grouped by the first word in their labels), the mistakes by this classifier seem to be more random.

Accuracy and Loss

Figure 7-5 shows the training history for the WE+CNN architecture. We can immediately observe that the validation loss fluctuated wildly throughout all 50 epochs. Fold 1, in particular stood out in comparison to the other folds with much higher validation loss values. The validation accuracy also had much milder fluctuations, though it stayed consistent around the 0.8 range. Both behaviours are in stark contrast to the training history results from BOW+MLP (Figure 7-3), which remained steady throughout the training phase.

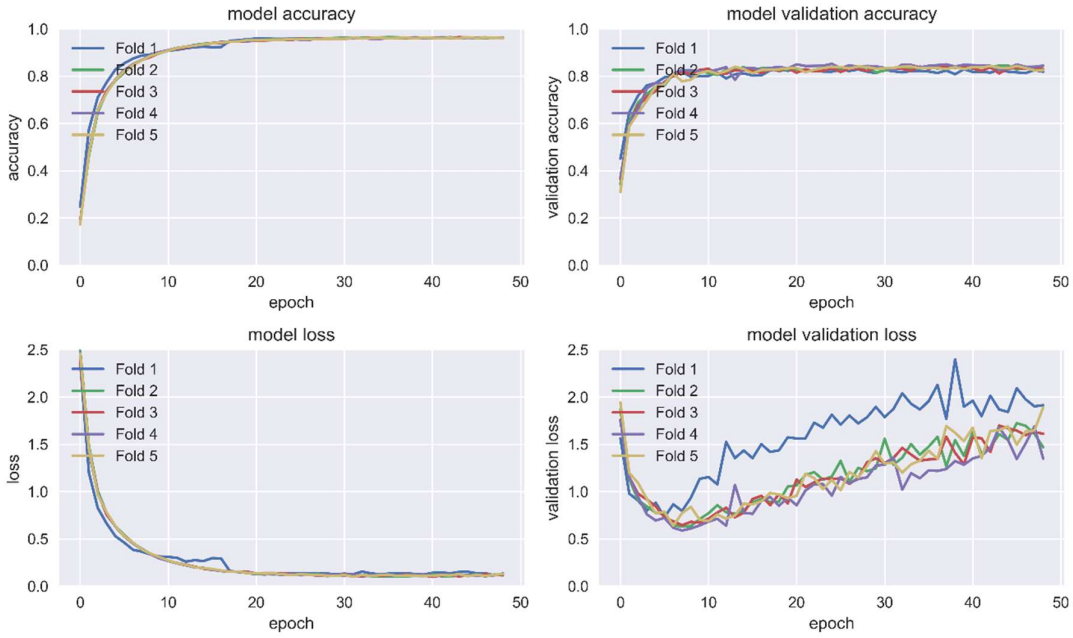

Figure 7-5 WE+CNN Training History

The results in Table 7-11 complement our observations from the confusion matrix in Figure 7-4, which suggested that the classifier not very confident in its predictions. While the validation accuracy values can be considered as average compared to the other classifiers in used in the experimental scenarios, the validation loss values, in contrast, are extremely high, which can be interpreted as low confidence on the quality of the predictions.

Table 7-11 Best Models per Fold for WE+CNN

Fold	Model Checkpoint	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	Epoch 21	0.9584	0.8315	0.1266	1.5615
2	Epoch 45	0.9625	0.8446	0.1134	1.5465
3	Epoch 35	0.9618	0.8401	0.1079	1.3256
4	Epoch 25	0.9563	0.8518	0.1276	0.9564
5	Epoch 46	0.9607	0.8431	0.1187	1.4955

7.2.3 WE+LSTM

The WE+LSTM scenario uses the same word embeddings as the WE+CNN, together with an LSTM network architecture.

Precision, Recall, and F1-score

Table 7-12 shows the average precision, recall, and F1-score for each fold during k-cross validation. We can see that the results for this model are nearly on par with the results from Section 7.2.1, which is the best model among all the individual classifier scenarios. The standard deviation between the values in each fold is also very low, which means that the results can be considered generally stable.

Table 7-12 WE+LSTM Classification Results by Fold

	Precision	**Recall**	**F1-Score**	**Support**
Fold 1	0.8738	0.8728	0.8730	3484
Fold 2	0.8813	0.8808	0.8807	3481
Fold 3	0.8834	0.8824	0.8823	3477
Fold 4	0.8743	0.8748	0.8742	3475
Fold 5	0.8802	0.8808	0.8800	3473
Standard Deviation σ	*0.0043*	*0.0042*	*0.0042*	-
Merged	0.8780	0.8783	0.8780	17390

We then look into the classification results for each class, as shown in Table 7-13. As this classifier has outperformed the baseline on the overall results, it also outperformed the results from each class, with the notable exception of the classes ending in *–misc* where the precision values are generally similar. When comparing the results to the ones from BOW+MLP, we observed that the performance of the results varied from class to class, where in some cases BOW+MLP had better results, and in other cases, WE+LSTM had better results. Thus, while BOW+MLP had a better overall score on a class-to-class level, there does not seem to be a clear winner.

Table 7-13 WE+LSTM (Merged) Classification Results

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
comp.os.ms-windows.misc	0.8478	0.8571	0.8525	1001
comp.sys.ibm.pc.hardware	0.7972	0.8325	0.8145	1015
comp.sys.mac.hardware	0.8811	0.8802	0.8807	1002
comp.windows.x	0.9192	0.9192	0.9192	1002
misc.forsale	0.8322	0.8446	0.8383	1010
rec.autos	0.9034	0.9034	0.9034	1004
rec.motorcycles	0.9531	0.9340	0.9434	1000
rec.sport.baseball	0.9584	0.9680	0.9632	1000
rec.sport.hockey	0.9662	0.9710	0.9686	1000
sci.crypt	0.9440	0.9440	0.9440	1000

sci.electronics	0.8586	0.8492	0.8539	1008
sci.space	0.9546	0.9451	0.9498	1001
soc.religion.christian	0.9211	0.9488	0.9348	997
talk.politics.guns	0.8374	0.8422	0.8398	1052
talk.politics.mideast	0.8793	0.8933	0.8863	1003
talk.politics.misc	0.7177	0.6968	0.7071	1171
talk.religion.misc	0.7957	0.7518	0.7731	1124
avetotal	0.8780	0.8783	0.8780	17390

The confusion matrix shown in Figure 7-6 also points toward this view, as its results have a similar pattern as the results in Figure 7-2. It does not perform as well in classifying the samples with labels starting with *comp*-, and also with labels starting with *talk*-.

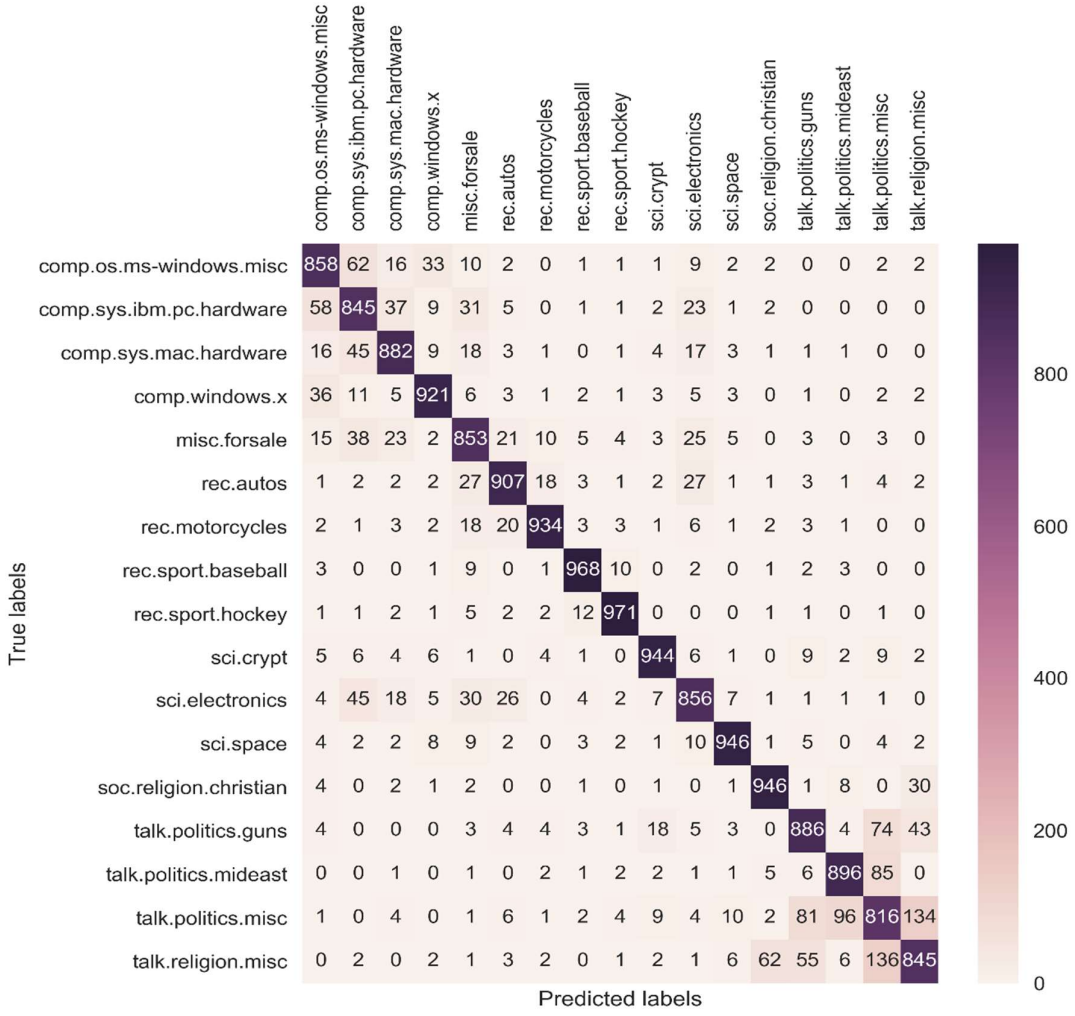

Figure 7-6 WE+LSTM (Merged) Confusion Matrix

Accuracy and Loss

Based on the training history shown in Figure 7-7, we can see that the accuracy and loss scores were much more stable throughout the training phase, especially in comparison to WE+CNN. Compared to the two previous neural network scenarios, the WE+LSTM network took longer to reach its maximum scores, as the previous models had much steeper curves. Furthermore, the training history of Fold 1 stands out compared to the other four folds, especially for validation loss where it consistently remained a little higher than the rest.

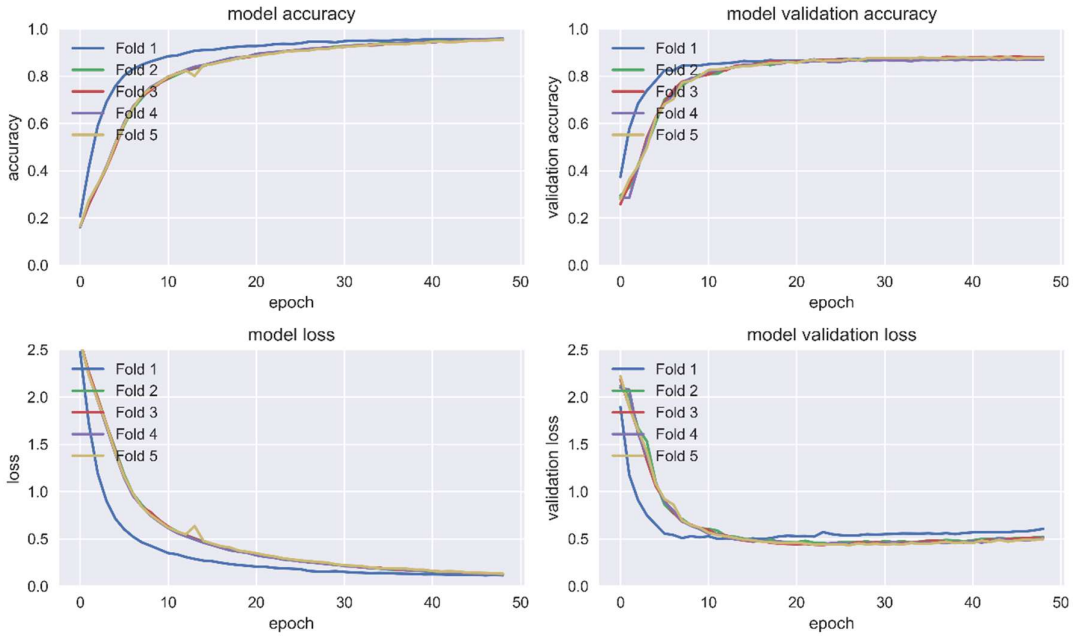

Figure 7-7 WE+LSTM Training History

In addition, unlike BOW+MLP and WE+CNN where the model last saved by ModelCheckpoint was early in the training phase, for WE+LSTM the model continues improving its validation accuracy up until the last 10 epochs. An exception is with Fold 1 which stopped halfway through the training phase, as shown in Table 7-14. This coincides with our earlier observation of Fold 1's training history in Figure 7-7 where it had a much steeper curve compared to the other folds.

Table 7-14 Best Models per Fold for WE+LSTM

Fold	Model Checkpoint	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	Epoch 27	0.9454	0.8728	0.1592	0.5349
2	Epoch 40	0.9463	0.8808	0.1579	0.4745
3	Epoch 46	0.9518	0.8824	0.1304	0.5012
4	Epoch 42	0.9471	0.8748	0.1477	0.4595
5	Epoch 45	0.9491	0.8808	0.1425	0.4694

While the accuracy of WE+LSTM is comparable to the results from BOW+MLP, the validation loss of the latter is still much better.

7.2.4 WE+CNNLSTM

This model uses word embeddings together with a neural network architecture combining convolutional and LSTM layers.

Precision, Recall, and F1-score

The overall classification results shown in Table 7-15 show that the WE+CNNLSTM model produces results comparable to the baseline, though it is not as good as either BOW+MLP or WE+LSTM.

Table 7-15 WE+CNNLSTM Classification Results by Fold

	Precision	Recall	F1-Score	Support
Fold 1	0.8567	0.8556	0.8549	3484
Fold 2	0.8687	0.8681	0.8679	3481
Fold 3	0.8752	0.8712	0.8721	3477
Fold 4	0.8698	0.8673	0.8671	3475
Fold 5	0.8679	0.8670	0.8659	3473
Standard Deviation σ	*0.0068*	*0.0060*	*0.0064*	-
Merged	0.8666	0.8658	0.8656	17390

Looking further into the results for each class, as shown in Table 7-16, we observe that while it performs very well for some clusters, it also performs poorly for others, with worse results than the baseline. For example, while it behaves very well with the classes starting with *rec-* and *sci-*, it has a lot of trouble classifying the classes starting with *comp-* and *talk-*.

Table 7-16 WE+CNNLSTM (Merged) Classification Results

Classes	Precision	Recall	F1-Score	Support
comp.os.ms-windows.misc	0.7798	0.8561	0.8162	1001
comp.sys.ibm.pc.hardware	0.7574	0.8305	0.7923	1015
comp.sys.mac.hardware	0.8863	0.8633	0.8746	1002
comp.windows.x	0.9228	0.9072	0.9149	1002
misc.forsale	0.8028	0.8020	0.8024	1010
rec.autos	0.8767	0.9203	0.8980	1004
rec.motorcycles	0.9536	0.9250	0.9391	1000
rec.sport.baseball	0.9590	0.9590	0.9590	1000
rec.sport.hockey	0.9787	0.9670	0.9728	1000
sci.crypt	0.9533	0.9390	0.9461	1000
sci.electronics	0.8697	0.8214	0.8449	1008
sci.space	0.9496	0.9421	0.9458	1001
soc.religion.christian	0.8775	0.9268	0.9015	997
talk.politics.guns	0.8208	0.8489	0.8346	1052
talk.politics.mideast	0.8756	0.8843	0.8800	1003

talk.politics.misc	0.7004	0.6806	0.6903	1171
talk.religion.misc	0.8080	0.7002	0.7502	1124
avetotal	0.8666	0.8658	0.8656	17390

The confusion matrix in Figure 7-8 further shows where the classifier is having trouble. A lot of texts are misclassified into the *comp-* and *talk-* clusters, which explains why their precision scores were so low.

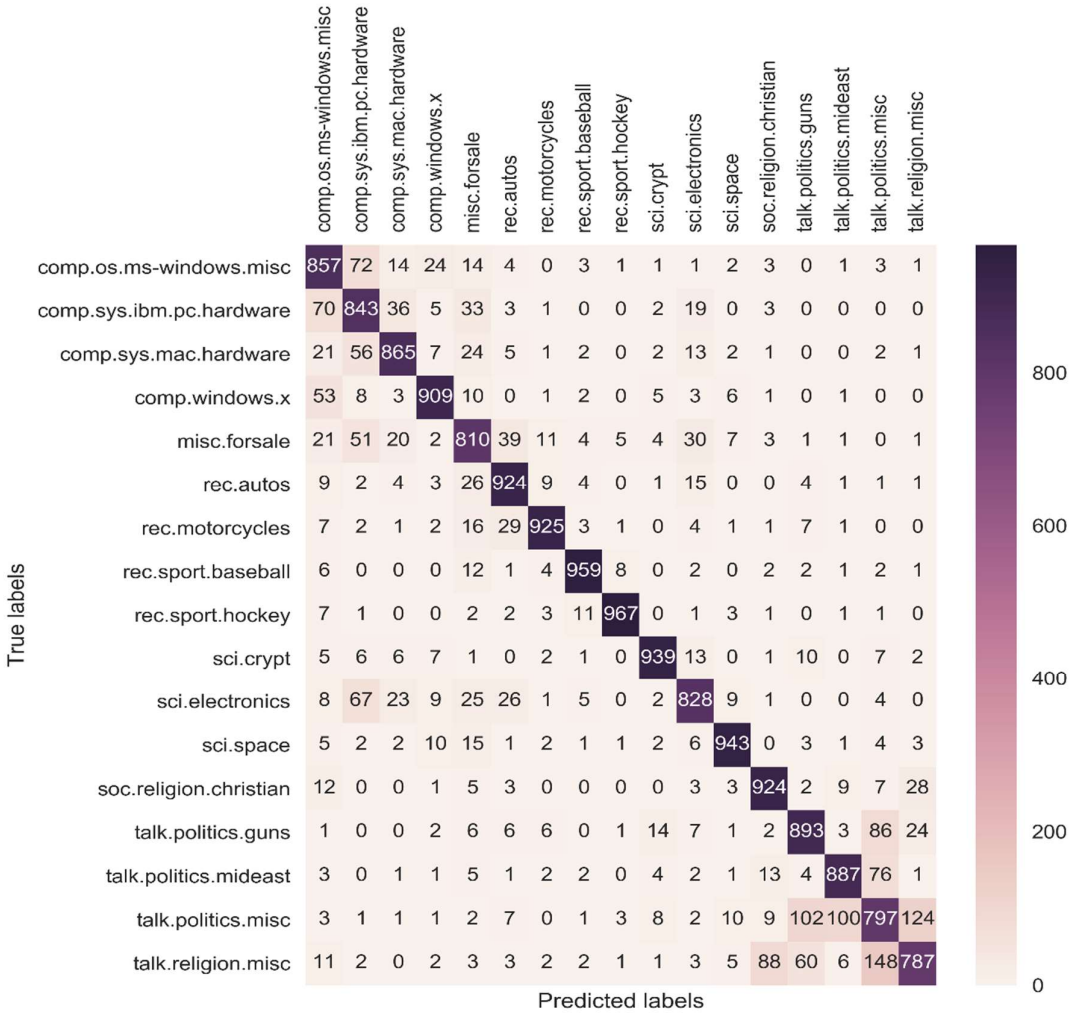

Figure 7-8 WE+CNNLSTM (Merged) Confusion Matrix

Accuracy and Loss

The training history in Figure 7-9 shows that the training results and especially the rate of increase/decrease are generally similar throughout each fold, though the validation loss in Fold 1 remains a little higher after the 10th epoch.

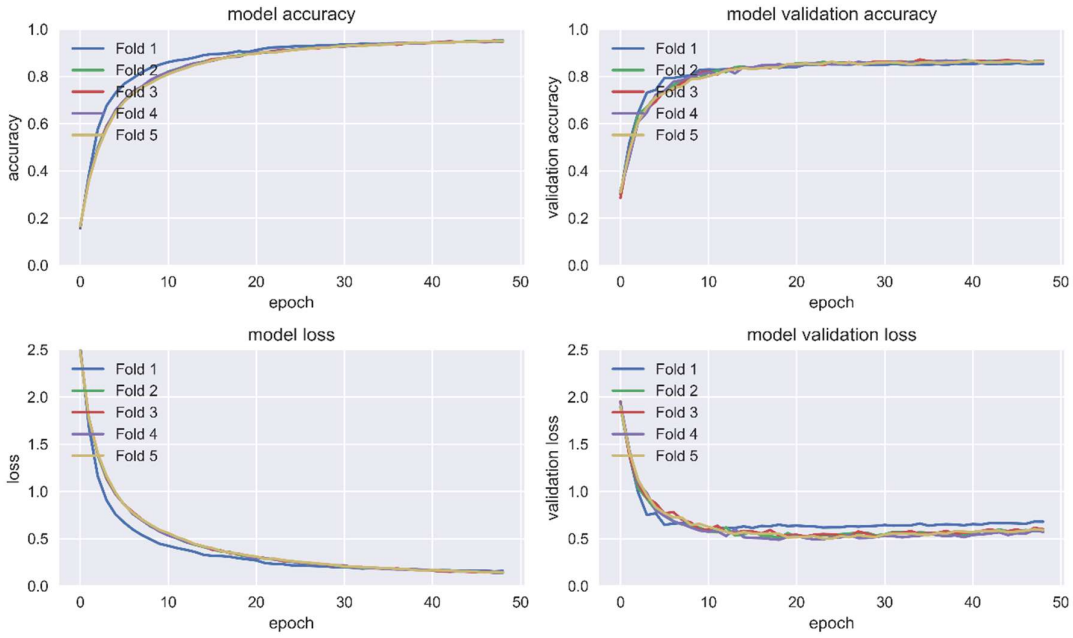

Figure 7-9 WE+CNNLSTM Training History

In addition, we analyse the best accuracy and loss results for each fold. Aside from the validation loss from Fold 1, which is much higher than the other folds, the rest of the validation loss values are similar to the best model from the baseline (Voting Classifier). Unlike the previous scenarios, the last epochs saved by ModelCheckpoint are all towards the end of the training phase, between 35-42 epochs.

Table 7-17 Best Models per Fold for WE+CNNLSTM

Fold	Model Checkpoint	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	Epoch 42	0.9433	0.8556	0.1657	0.6491
2	Epoch 40	0.9405	0.8681	0.1659	0.5417
3	Epoch 35	0.9335	0.8712	0.1871	0.5467
4	Epoch 38	0.9402	0.8673	0.1742	0.5264
5	Epoch 41	0.9435	0.8670	0.1669	0.5700

7.2.5 WE+RCNN

The last of the individual neural network models is the WE+RCNN, which uses word embeddings together with an RCNN model.

Precision, Recall, and F1-score

Similar to the WE+CNNLSTM, the WE+RCNN model generates results comparable to the baseline, though the baseline results are marginally better. The overall results per fold are shown in Table 7-18.

Table 7-18 WE+RCNN Classification Results by Fold

	Precision	Recall	F1-Score	Support
Fold 1	0.8643	0.8622	0.8617	3484
Fold 2	0.8695	0.8702	0.8691	3481
Fold 3	0.8623	0.8617	0.8610	3477
Fold 4	0.8723	0.8717	0.8711	3475
Fold 5	0.8675	0.8670	0.8669	3473
Standard Deviation σ	*0.0040*	*0.0045*	*0.0045*	-
Merged	0.8658	0.8665	0.8660	17390

Looking further into the results by class, as shown in Table 7-19, we can see an interesting comparison to the baseline. Although the overall score of the baseline is better, the scores for each class are better with WE+RCNN, with the exception of *talk.politics.misc* and *talk.religion.misc*. In particular, while several classes in the baseline results perform really badly and other perform really well, the scores among the classes for WE+RCNN seem more balanced, with all but two of the results above 0.79. It performs very well in classifying the labels starting with *comp-*, even better than BOW+LSTM.

Table 7-19 WE+RCNN (Merged) Classification Results

Classes	Precision	Recall	F1-Score	Support
comp.os.ms-windows.misc	0.8512	0.8402	0.8457	1001
comp.sys.ibm.pc.hardware	0.8056	0.8000	0.8028	1015
comp.sys.mac.hardware	0.8673	0.8673	0.8673	1002
comp.windows.x	0.8918	0.9381	0.9144	1002
misc.forsale	0.8214	0.8287	0.8250	1010
rec.autos	0.8869	0.9064	0.8966	1004
rec.motorcycles	0.9446	0.9380	0.9413	1000
rec.sport.baseball	0.9541	0.9560	0.9550	1000
rec.sport.hockey	0.9670	0.9660	0.9665	1000
sci.crypt	0.9462	0.9330	0.9396	1000
sci.electronics	0.8466	0.8433	0.8449	1008
sci.space	0.9371	0.9371	0.9371	1001
soc.religion.christian	0.8937	0.9358	0.9143	997
talk.politics.guns	0.8143	0.8460	0.8298	1052
talk.politics.mideast	0.8773	0.8694	0.8733	1003
talk.politics.misc	0.6815	0.6670	0.6741	1171
talk.religion.misc	0.7789	0.7144	0.7452	1124
avetotal	0.8658	0.8665	0.8660	17390

We further look at the confusion matrix in Figure 7-10 to see where the misclassifications cluster around. A lot of the texts under the *talk-* labels are misclassified into other *talk-* labels, which explains the low precision score for those classes. In addition, less texts are misclassified in general among the *comp-* labels which explains the improvement of the scores for that cluster.

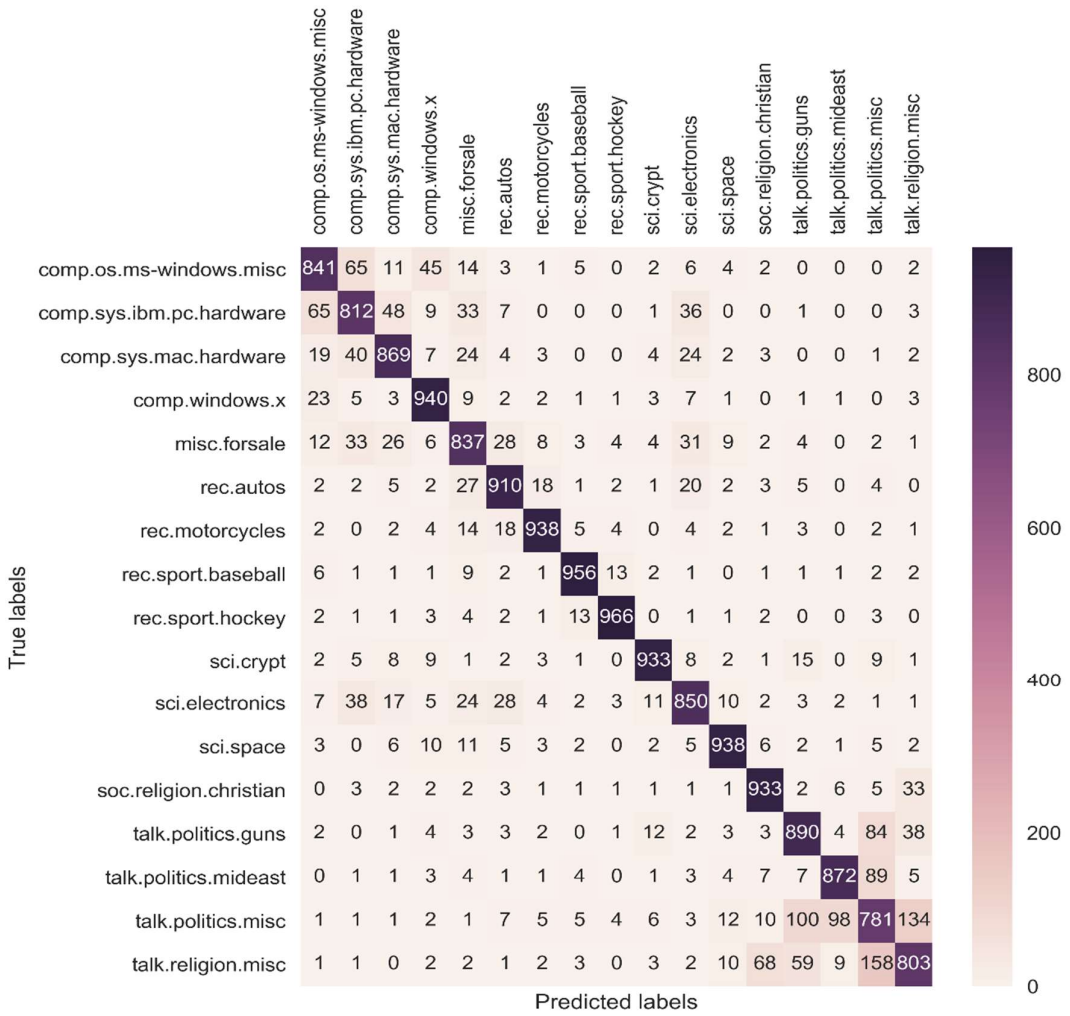

Figure 7-10 WE+RCNN (Merged) Confusion Matrix

Accuracy and Loss

The training history in Figure 7-11 shows the trend of the model's improvement throughout the training phase. In comparison to WE+CNN, WE+LSTM, and WE+CNNLSTM, which started with low accuracy and high loss values before gradually decreasing, WE+RCNN starts at a moderately high validation accuracy and lower validation loss. However, the validation loss increases in proportion to the number of epochs, which suggests overtraining.

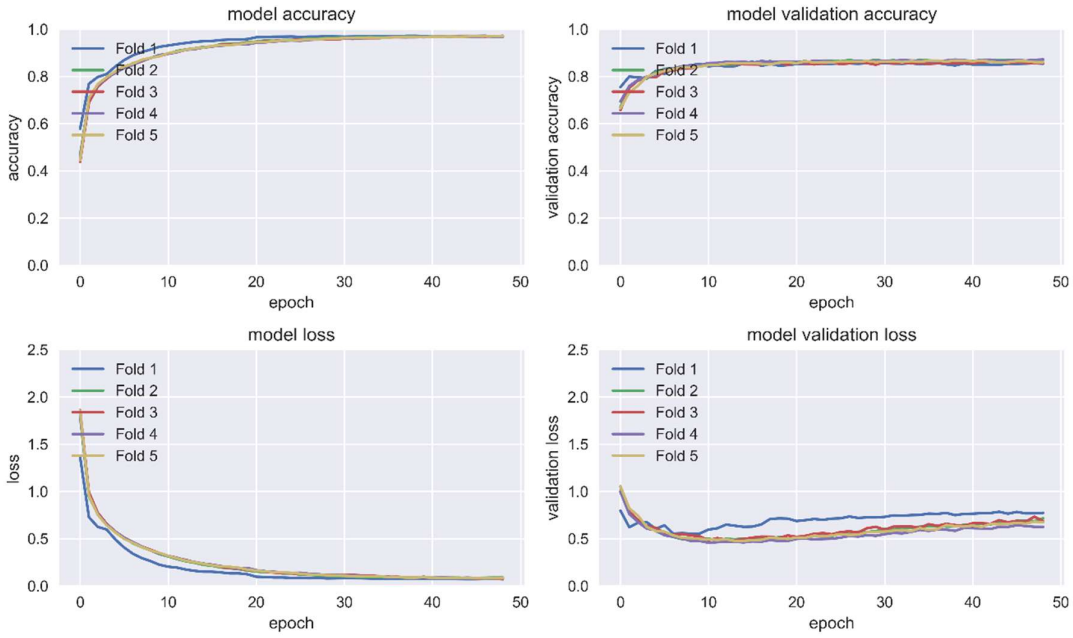

Figure 7-11 WE+RCNN Training History

Finally, we look into the best models per fold for WE+RCNN. Unlike the previous model (WE+CNNLSTM), the last epoch saved per fold is inconsistent, with Epoch 16 for Fold 1 and Epoch 49 for Fold 4. In addition, even though the validation accuracy values are quite good, the validation loss values are subpar, although still superior to the results of WE+CNN.

Table 7-20 Best Models per Fold for WE+RCNN

Fold	Model Checkpoint	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	Epoch 16	0.9499	0.8622	0.1494	0.6271
2	Epoch 35	0.9706	0.8510	0.0753	0.7508
3	Epoch 43	0.9682	0.8617	0.0865	0.6564
4	Epoch 49	0.9663	0.8717	0.0842	0.6241
5	Epoch 37	0.9657	0.8670	0.0925	0.6006

7.2.6 Voting Classifier

Finally, we implemented a Voting Classifier using the prediction probabilities from the five neural network classifiers (BOW+MLP, WE+CNN, WE+LSTM, WE+CNNLSTM, WE+RCNN). We used both 'hard' voting, which used the majority vote from the classifiers and 'soft' voting which used the mean of the probabilities to produce the final prediction. During the hard voting process, there were 44 rows where each classifier voted for a

different result so there was no majority, thus they needed to be removed. In order to keep the same number of texts for comparison, we decided to also remove the same rows from the soft voting.

Precision, Recall, and F1-score

Table 7-21 shows the results from hard voting and Table 7-22 shows the results from soft voting.

Table 7-21 Voting Classifier Hard Classification Results

Classes	Precision	Recall	F1-Score	Support
comp.os.ms-windows.misc	0.8570	0.8819	0.8693	999
comp.sys.ibm.pc.hardware	0.8291	0.8439	0.8364	1012
comp.sys.mac.hardware	0.9067	0.8931	0.8998	1001
comp.windows.x	0.9390	0.9381	0.9385	1001
misc.forsale	0.8295	0.8484	0.8388	1009
rec.autos	0.9071	0.9271	0.9170	1001
rec.motorcycles	0.9680	0.9437	0.9557	994
rec.sport.baseball	0.9681	0.9759	0.9720	996
rec.sport.hockey	0.9808	0.9739	0.9774	997
sci.crypt	0.9626	0.9539	0.9582	997
sci.electronics	0.8761	0.8726	0.8744	1005
sci.space	0.9598	0.9559	0.9578	998
soc.religion.christian	0.9068	0.9607	0.9329	992
talk.politics.guns	0.8390	0.8669	0.8527	1052
talk.politics.mideast	0.8903	0.8983	0.8943	1003
talk.politics.misc	0.7202	0.7029	0.7114	1168
talk.religion.misc	0.8071	0.7279	0.7655	1121
avetotal	0.8885	0.8889	0.8884	17346

Table 7-22 Voting Classifier Soft Classification Results

Classes	Precision	Recall	F1-Score	Support
comp.os.ms-windows.misc	0.8481	0.8829	0.8651	999
comp.sys.ibm.pc.hardware	0.8328	0.8370	0.8349	1012
comp.sys.mac.hardware	0.9093	0.8911	0.9001	1001
comp.windows.x	0.9344	0.9391	0.9367	1001
misc.forsale	0.8358	0.8523	0.8440	1009
rec.autos	0.9049	0.9321	0.9183	1001
rec.motorcycles	0.9692	0.9507	0.9599	994
rec.sport.baseball	0.9662	0.9749	0.9705	996
rec.sport.hockey	0.9818	0.9729	0.9773	997
sci.crypt	0.9646	0.9569	0.9607	997
sci.electronics	0.8878	0.8736	0.8806	1005
sci.space	0.9589	0.9579	0.9584	998
soc.religion.christian	0.9105	0.9637	0.9363	992

talk.politics.guns	0.8404	0.8660	0.8530	1052
talk.politics.mideast	0.8842	0.8983	0.8912	1003
talk.politics.misc	0.7199	0.6952	0.7073	1168
talk.religion.misc	0.8084	0.7342	0.7695	1121
avetotal	0.8890	0.8896	0.8891	17346

Combining the results for both cases resulted in improved scores on all measures. Overall, the soft voting had slightly better results than hard voting, though when we look at the results for each class, we can see that there were some classes where hard voting scored better. The class with the best scores was *rec.sport.hockey* which has an F1-score of 0.98, and the class with the worst scores was *talk.politics.misc* which had an F1-score of 0.71.

Accuracy and Loss

We then evaluate the accuracy and loss values from both voting classifiers, shown in Table 7-23. Since hard voting uses final predictions and not probabilities, it does not have a loss value. We can see that using soft voting to combine the prediction probabilities has resulted in more confident predictions from the classifiers. The loss value is 0.3517, which is the highest among the different scenarios.

Table 7-23 Accuracy and Loss for Voting Classifiers

	Accuracy	**Loss**
Voting Classifier Hard	0.8889	-
Voting Classifier Soft	0.8895	0.3517

7.3 Evaluation of Classifier Training Efficiency

The last section in our analysis takes a look at the efficiency of the different classifiers in our experimental scenarios.

Table 7-24 shows the recorded training time of the baseline scenario and the baseline with linguistic processing. Comparing the times from both sets shows that the additional linguistic processing has not made a noticeable impact on the training time.

Table 7-24 Comparison of Training Time (in seconds) Baseline and Baseline+LP

	Baseline	**Baseline+LP**
LinearSVC	64	67
NBBernoulli	19	18
NBMultinomial	19	19
passiveaggressive	32	31
Perceptron	35	36
Ridge	61	43
SGD	70	71
SVM	1364	1385
Voting Classifier	1542	1567

However, when we compare the baseline results to the neural networks approaches (shown in Table 7-25), we can see that the neural networks approaches take much more time to train.

Table 7-25 Comparison of Training Time (in seconds) for Neural Networks

Neural Networks	
BOW+MLP[25]	2435 *(41 min)*
WE+CNN	3096 *(52 min)*
WE+LSTM	49551 *(13.8 hours)*
WE+CNNLSTM	11765 *(3.27 hours)*
WE+RCNN	93280 *(25.9 hours)*

BOW+MLP had the fastest training time at 41 minutes, followed by WE+CNN, then WE+CNNLSTM, WE+LSTM then finally by WE+RCNN. WE+LSTM and WE+RCNN had extremely long training times compared to the other scenarios, with WE+LSTM taking three times longer to train than WE+CNNLSTM, and WE+RCNN taking twice as long as WE+LSTM. Furthermore, given that the last saved model by ModelCheckpoint for BOW+MLP was at 22 epochs, cutting the number of epochs by half would have reduced the training time even further.

[25] Trained on CPU. See 6.1 Tools and Libraries (Environment)

8 Conclusions and Suggestions for Future Work

In this thesis, we aimed to improve the current text classifier algorithm of Siav through a combination of linguistic processing techniques and neural network-based classification algorithms

Our first approach was to apply only linguistic processing techniques on the text, in order to determine whether or not these techniques by themselves could improve the performance or efficiency of the classifier. The results from this approach showed that instead of an improved model, its performance deteriorated, and there was no real impact on the training time. Using this approach on larger or more complicated datasets may provide further insight into the real benefits of this approach.

The next set of approaches used several neural-network architectures with two different feature representation models. The first neural-network approach used a bag-of-words (BOW) model with an Multilayer Perceptron (MLP), while the remaining four approaches used pre-trained GloVe word embeddings (WE) with a Convolutional Neural Network (CNN), Long Short Term Memory (LSTM) network, combination of a CNN and LSTM (CNNLSTM) and Recurrent Convolutional Neural Network (RCNN). Comparing the results showed that even though the BOW+MLP was the simplest model, it had the best performance out of all the approaches. However, in order to do so, we had to greatly increase the feature dimension, which may impact the efficiency of the model on datasets with larger corpora. Among the other neural network-based approaches, we consider WE+CNN unsuitable for use given the subpar classification results and extremely high log loss values. In contrast, WE+LSTM would be a good candidate for future projects as it is the next best model after BOW+MLP, though the long training time could hamper its usability on larger datasets. The last two models, CNNLSTM and RCNN, while producing good results by themselves, do not outperform the baseline, and take much longer to train.

Finally, we used all of the predictions from the five neural network based approaches and generated final predictions using a Voting Classifier algorithm. As ensemble methods usually result in improved results, we were not surprised to have the same outcome on our implementation. We found that using soft voting was a better approach, as firstly, we would not have the problem where the classifiers could not agree on a single class, and secondly, soft voting would be able to take the confidence of the predictions into consideration.

Overall, we have found the simplest models have provided the best results, which suggests that increasing the complexity of a model may cause it to lose important information which affects its performance as a classifier. We have provided several text classification models that may be considered as improvements based on the results, though which one to select in particular would depend on other factors, such as memory requirements and complexity of data.

Future Work

The work done for this study has shown us that there are several other things that we can still try in order to improve the performance of the text classifier. One technique that we would like to use in the future is to implement hyperparameter optimization using more efficient techniques, such as Grid Search. In addition, other combinations of neural network architectures may also be considered.

Finally, linguistic processing techniques vary from language to language, so what would work for English texts may not necessarily be so for other languages. Furthermore, the pre-trained word embeddings used had been trained on English corpora, so for other languages,

these embeddings would have to be replaced with more appropriate ones. Thus, another direction of study would be to research on approaches for non-English texts.

9 References

[1] S. Dumais, J. Platt, D. Heckerman ja M. Sahami, „Inductive Learning Algorithms and Representations for Text Categorization," *CIKM '98 Proceedings of the Seventh International Conference on Information and Knowledge Management,* pp. 148-155, 1998.

[2] F. Sebastiani, „Machine Learning in Automated Text Categorization," *ACM Computing Surveys (CSUR),* kd. 34, nr 1, pp. 1-47, 2002.

[3] D. D. Lewis, „Feature selection and feature extraction for text categorization," *HLT '91 Proceedings of the workshop on Speech and Natural Language,* pp. 212-217, 1992.

[4] J. O. P. Yiming Yang, „A Comparative Study on Feature Selection in Text Categorization," %1 *International Conference on Machine Learning*, Nashville, Tennessee, 1997.

[5] V. Pekar, „Linguistic Preprocessing for Distributional Classification of Words," %1 *Workshop on Enhancing and Using Electronic Dictionaries*, Geneva, 2004.

[6] T. Z. Rie Johnson, „Effective Use of Word Order for Text Categorization with Convolutional Neural Networks," %1 *North American Chapter of the Association for Computational Linguistics Human Language Technologies*, 2015.

[7] M. Rogati ja Y. Yang, „High-Performing Feature Selection for text classification," *CIKM '02 Proceedings of the eleventh international conference on Information and knowledge management,* pp. 659-661, 2002.

[8] R. D. P. V. C. J. Yoshua Bengio, „A Neural Probabilistic Language Model," *Journal of Machine Learning Research,* pp. 1137-1155, 2003.

[9] K. L. Yunpeng Li, „Word representation using a deep neural network," *CASCON '16 Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering,* pp. 268-279, 2016.

[10] T. &. C. K. &. C. G. &. D. J. Mikolov, „Efficient Estimation of Word Representations in Vector Space," *ICLR,* 2013.

[11] T. a. S. I. a. C. K. a. C. G. S. a. D. J. Mikolov, „Distributed Representations of Words and Phrases and their Compositionality," %1 *Neural Information Processing Systems*, Lake Tahoe, 2013.

[12] W. a. L. J. a. J. Y. Rui, „Unsupervised Feature Selection for Text Classification via Word Embedding," %1 *IEEE International Conference on Big Data Analysis (ICBDA)*, Hangzhou, 2016.

[13] P. a. G. E. a. J. A. a. M. T. Bojanowski, „Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics,* kd. 5, pp. 135-146, 2017.

[14] A. a. G. E. a. B. P. a. M. T. Joulin, „Bag of Tricks for Efficient Text Classification," %1 *Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, 2017.

[15] Q. a. M. T. Le, „Distributed Representation of Sentences and Documents," %1 *International Conference on Machine Learning*, Beijing, 2013.

[16] Y. a. L. X. Yang, „A re-examination of text categorization methods," %1 *ACM SIGIR conference on Research and development in information retrieval*, Berkeley, 1999.

[17] B. a. G. X. a. Y. Y. Xu, „An Improved Random Forest Classifier for Text,“ *Journal of Computers,* kd. 7, nr 12, pp. 2913-2920, 2012.

[18] P. R. &. H. S. Christopher D. Manning, Introduction to Information Retrieval, Cambridge University Press, 2008.

[19] R. S. a. C. D. M. Jeffrey Pennington, „GloVe: Global Vectors for Word Representation,“ %1 *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[20] J. a. K. I. a. P. J. Clark, „A Neural Network Based Approach to Automated E-mail Classification,“ %1 *IEEE/WIC International Conference on Web Intelligence*, Halifax, 2003.

[21] Y. Kim, „Convolutional Neural Networks for Sentence Classification,“ *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* pp. 1746-1751, 2014.

[22] J. &. T. A. &. S. R. Nowak, „LSTM Recurrent Neural Networks for Short Text and Sentiment Classification,“ %1 *Conference: International Conference on Artificial Intelligence and Soft Computing*, 2017.

[23] C. &. S. C. &. L. Z. &. L. F. Zhou, „A C-LSTM Neural Network for Text Classification,“ %1 *arxiv*, 2015.

[24] L. X. K. L. J. Z. Siwei Lai, „Recurrent Convolutional Neural Networks for Text Classification,“ %1 *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[25] A. Moschitti, „Natural Language Processing and Automated Text Categorization A study on the reciprocal beneficial interactions (Doctoral Thesis),“ University of Rome, 2003.

[26] H. M. Wallach, „Topic Modeling: Beyond Bag of Words,“ *ICML '06 Proceedings of the 23rd international conference on Machine learning,* pp. 977-984, 2006.

[27] R. J. Z.-H. Z. Yin Zhang, „Understanding bag-of-words model: a statistical framework,“ *International Journal of Machine Learning and Cybernetics,* kd. 1, nr 1-4, pp. 43-52, 2010.

[28] C. B. Gerard Salton, „Term Weighting Approaches in Automatic Text Retrieval,“ *Information Processing & Management,* kd. 24, nr 5, p. 513=523, 1988.

[29] J. Z. Y. L. Xiang Zhang, „Character-level Convolutional Networks for Text Classification,“ %1 *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, 2015.

[30] G. Salton ja C. Buckley, „Term-weighting approaches in automatic text retrieval,“ *In Information Processing & Management,* kd. 24, nr 25, pp. 513-523, 1988.

[31] I. S. K. C. G. C. J. D. Tomas Mikolov, „Distributed Representations of Words and Phrases and their Compositionality,“ *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems,* kd. 2, pp. 3111-3119, 2013.

[32] G. D. G. K. Marco Baroni, „Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors,“ *Proceedings of Association for Computational Linguistics (ACL),* 2014.

[33] C. L., Encyclopedia of Database Systems, Boston: Springer, 2009.

[34] A. &. L. D. &. M. D. Genkin, „Large-Scale Bayesian Logistic Regression for Text Classification,“ *Technometrics,* kd. 49, nr 3, pp. 291-304, August 2007.

[35] S. a. H. A. Bloehdorn, „Boosting for Text Classification with Semantic Features,“ %1 *Advances in Web Mining and Web Usage Analysis*, Seattle, 2004.

[36] J. a. S. R. a. M. C. D. Pennington, „GloVe: Global Vectors for Word Representation,“ %1 *Empirical Methods in Natural Language Processing*, Doha, 2014.

Appendix

I. Sample text from 20NewsGroups

Path: cantaloupe.srv.cs.cmu.edu!rochester!udel!gatech!howland.reston.ans.net!usc!cs.utexas.edu!qt.cs.utexas.edu!news.Brown.EDU!noc.near.net!bigboote.WPI.EDU!bigwpi.WPI.EDU!kedz

From: kedz@bigwpi.WPI.EDU (John Kedziora)

Newsgroups: misc.forsale

Subject: Motorcycle wanted.

Date: 22 Feb 1993 14:22:51 GMT

Organization: Worcester Polytechnic Institute

Lines: 11

Expires: 5/1/93

Message-ID: <1manjr$ja0@bigboote.WPI.EDU>

NNTP-Posting-Host: bigwpi.wpi.edu

Sender:

Followup-To:kedz@wpi.wpi.edu

Distribution: ne

Organization: Worcester Polytechnic Institute

Keywords:

I am looking for an inexpensive motorcycle, nothing fancy, have to be able to do all maintinence my self. looking in the <$400 range.

if you can help me out, GREAT!, please reply by e-mail.

Figure 9-1 Sample text from 20NewsGroups

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Geraldine King Granada**,

herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Improving the Automatic Text Classification Algorithm of Siav, A Case Study,

supervised by Fabrizio Maggi, PhD and Daniele Turato,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **21.05.2018**