University of Nottingham
UK | CHINA | MALAYSIA

Dissertation

Emotion Recognition using Text Analytics Approach

Submitted May 18, 2020.

Masrur SOBIROV

16519829

zy19829@nottingham.edu.cn

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: [illegible]

Supervised by Boon Giin Lee

boon-giin.lee@nottingham.edu.cn

School of Computer Science

University of Nottingham

Abstract

Today a lot of Internet users share their opinions and thoughts on social media such as Twitter. Understanding and recognizing the emotions of this information can help to advance various technological areas such as Human-Computer Interaction or improve the marketing of products. This paper analyzes how emotion recognition can be accomplished using machine learning. Particularly, the paper demonstrates how emotion recognition can be done using the deep learning's Long Short-Term Memory (LSTM) model. The LSTM classifier was used to classify the datasets of Twitter posts into a different group of classes: positive and negative; neutral, worry, happiness, sadness, love, and fun. The tweets are not perfectly organized and are cluttered with useless information, therefore different preprocessing techniques were compared and used to clean them. The results of classifications showed that LSTM is capable of emotion recognition, albeit the classification accuracy was worse than the state-of-the-art models.

Contents

1. Introduction

Today, a large amount of information and data is transferred in text form, namely in news articles, social media posts, blogs, books, etc. Text messaging and emailing are the most popular ways of communication in our connected society. In fact, 97% of all smartphone owners in America send or receive text messages at least once every week [1]. Additionally, text messaging is also the most frequently used type of communication for smartphone owners. Twitter messages and product reviews are also in text form and contain a lot of useful information. In 2014, Twitter reported that people send more than 500 million tweets per day [2]. These findings suggest that a substantial share of information is transferred via text which if analyzed provides invaluable benefits.

This project intended to research and experiment on how such an analysis can be done. Specifically, how emotions expressed in a text can be extracted using deep learning techniques. Every writer of a comment, product review, or tweet expresses their emotion on a topic using words. The six main emotions which are focused on are sadness, joy, fear, surprise, hate, and anger. Emotions are a very important part of a person and are something that is not understood completely. Thus, it can be very valuable to be able to recognize emotions from text and improve technological areas.

There are several ways how such recognition can be done and the machine learning approach is one of them. Moreover, the machine learning approach includes many different algorithms, techniques, and concepts. This project generally focuses on one such technique that is deep learning. Including machine learning, other methods are going to be explained as well namely, knowledge-based and hybrid methods. Emotion classification is also considered as the Natural Language Processing (NLP) problem which focuses on designing algorithms to analyze and process a substantial amount of natural language data.

1.1 Motivation

The main architecture that is used to recognize emotions in this project is called Long Short-Term Memory (LSTM). LSTM belongs to the deep learning field which is particularly useful in NLP because it can remember textual information. Much research has been carried out on emotion recognition using LSTM variants. Peng Zhou et al. (2016) researched and proposed their solution for LSTM classification and compared their results with other similar studies [3]. However, the comparisons show that the classification results with five and two classes of sentiments (using different models including LSTMs) are roughly 50% and 88%, respectively. The results suggest that there is room for improvement and it is worthwhile to explore how the classification accuracy can be improved.

Analyzing and recognizing emotions in text information with great accuracy can provide advancements in various products and services. First, as mentioned earlier, a lot of customers write and read reviews on the different types of topics such as books, games, movies, etc. In fact, a report done in 2017 states that 93% of consumers say that the reviews affect their purchase and 77% would leave a review if asked [4]. Understanding the emotions of these reviews could help companies to improve their products or marketing by understanding what the public wants and dislikes (discussed later).

Next, recognizing key emotions could help in identifying emotional people on the Internet. On many different websites, a chatbot is incorporated that could help with customer questions. If such chatbot could recognize a customer's mood, it could reply accordingly or route the discussion to a human agent when there is a serious matter such as a very angry customer. Additionally, discussions can be analyzed to see which messages emotionally trigger people. For instance, a customer could get angry or annoyed when some phrase was used, or the conversation's tone could improve when exclamation marks or emojis are used. Finally, Human-Computer Interaction could improve which is discussed more in detail later including the mentioned benefits above.

Emotion recognition using a text analysis approach is an active field of research in recognizing a subject's opinion or emotion via text using different techniques such as Support Vector Machine (SVM) [5], Maximum Entropy [6] and Naïve Bayes [7]. Textual emotion recognition can be more reliable compared to speech or facial recognition due to a couple of reasons. First, collecting good quality texts is much easier since a substantial amount of information is in text form which should result in achieving higher accuracy of classification. Next, when recognizing facial or speech emotion, some emotions can be confused with each other, namely neutral with sadness [8, 9]. Finally, facial and speech emotion recognition are dependent on the quality and availability of the video and audio. For instance, when recognizing facial emotion, without marking important points of the face, recognition can be inaccurate and/or challenging.

The rest of this paper is structured as follows. The next chapter talks about the background and work done previously on the topic. Chapter 3 describes the design choices of the project's classifier and application. Chapters 4 and 5 explain the project's implementation and achieved results, respectively. Chapters 6 will describe the project management of the project and encountered challenges. Finally, Chapter 7 summarises the paper by discussing the achieved outcome and reflecting upon it.

2. Background and Related Work

2.1 Human-Computer Interaction

One important field that would greatly improve from emotion recognition using text is Human-Computer Interaction (HCI). HCI is a field of study that focuses on researching how computers and programs can be designed to have a convenient way of interaction between people and computers. The term was defined in the 1980s in the book *The Psychology of Human-Computer Interaction* by Stuart K. Card, Thomas P. Morana, and Allen Newell [10]. Today, many desktop applications and internet browsers are designed in such a way to make their interfaces feel intuitive and friendly for the users. Most applications today make use of graphical user interface (GUI) where graphical and audio elements are integrated for the interaction.

Emotion recognition is another new path to improving HCI and affective computers. To make computers fully "understand" the emotions of humans, a wide range of expressions needs to be processed such as speech [11], facial [12], and movement [13]. As explained earlier, a huge amount of emotional information is also expressed in text form. In the future, combining all these different ways of emotion recognition may achieve human-like recognition, and as a result interaction between computers and humans will be more natural. Additionally, emotion recognition using text alone can be used to implement text-to-speech programs that could be very helpful for people with communication disorders to express themselves more naturally. More specifically, such individuals could write their opinion in text form and the text-to-speech program would express it with emotions.

2.2 Marketing

Companies could greatly benefit from sentiment analysis of social media comments and posts. Today, opinions on products or services are mostly shared on the Internet in text form. Analyzing these opinions and extracting the sentiment of the public on products and services can help companies and organizations to understand what the public likes and dislikes. Additionally, using sentiment analysis companies could measure their brand image on Internet [14]. Different strategies are always used in the advertisements to maximize the positive feedback because different strategies can determine the emotions expressed on a product [15]. Overall, analyzing the satisfaction with products and services of users can help companies in achieving better design and delivery of their products and services.

Emotion recognition is a task that has become very popular recently and strong research has been carried out in this field. The popularity of social media in our everyday lives and the amount of data transferred there are the main reasons. The following focuses on previous work done on emotion recognition and its results. As listed earlier, this project and most of the other related work focuses on six basic human emotions: surprise, sadness, hate, happiness, anger, and fear. These six emotions are understood and expressed similarly by almost every culture. To implement an algorithm that would recognize these emotions a few main methods exist, namely, knowledge-based method, statistical method, and hybrid [16].

2.3 Knowledge-based method

Knowledge-based textual emotion recognition makes classifications based on the presence of obvious sentimental keywords in the text. These words are usually manually categorized as some particular emotion in a lexicon which is an available resource such as WordNet-Affect [17] and SenticNet [18]. This method has a benefit due to being simple and straightforward to use but has a major weakness. Its disadvantage includes inflexibility and inaccuracy in intricate sentences that cannot be classified correctly just by the presence of keywords categorized in the lexicon [19].

Many models are already developed or being developed applying the approach mentioned above. For instance, one such knowledge-based model for sentiment analysis called VADER (Valence Aware Dictionary for sEntiment Reasoning), was implemented to analyze social media [20]. Particularly, this model was tested to analyze tweets, and results showed that it outperforms human raters in classifying the tweets into three groups of emotions: positive, negative, and neutral. To have such performance, developers of VADER implemented a gold-standard sentiment lexicon combined with grammatical and syntactical conventions [20]. The developers reported that their knowledge-based algorithm, considering its results, has major advantages: it is computationally fast without sacrificing accuracy, the accessible engine can easily be customized and performs well in diverse situations without requiring a large amount of training data.

2.4 Statistical method

In a statistical method of recognizing the emotion from a text, machine learning is used where a model is trained on a corpus of training samples labeled with emotions. The trained model would learn the patterns on the training samples and then predict the emotions of other samples that don't have labels. The machine learning method includes supervised and unsupervised learning models. In supervised learning, a model is trained on a dataset with input and corresponding labels to correctly output the labels of new unfamiliar samples. Whereas, for models in unsupervised learning the training is done without labels to recognize the patterns in the data space. Support Vector Machines (SVM), Naïve Bayes classifier, and deep learning are considered successful machine learning algorithms for textual emotion recognition. One of the main drawbacks of this method includes the requirement of a sufficiently large dataset to achieve adequate results.

2.4.1 Support Vector Machines

The SVM algorithm was developed by Vladimir Vapnik in the 1990s and since then is successfully used in text categorization [21, 22]. SVMs are algorithms that are considered as supervised learning models. A linear classifier is the most common SVM known which classifies inputs into two possible categories. Robert Gove and Jorge Faytong (2012) stated the clear definition of SVM [23]:

> ”Support Vector Machine builds a hyperplane or set of hyperplanes to classify all inputs in a high-dimensional or even infinite space. The closest values to the classification margin are known as support vectors. The SVM’s goal is to maximize the margin between the hyperplane and the support vectors.”

Research is done in 2004 by ZJ Chuang and CH Wu, who utilized SVM to propose a system that combines textual and speech data to recognize emotions [24]. The speech and textual recognition modules classified the samples individually and the outputs were then combined to have multi-modal input. The textual information was collected from broadcast dramas and manually labeled. The sentences were classified into six main emotions listed earlier. The textual emotion recognition module alone classified correctly an average of 65% of inputs. On the other hand, the speech recognition module achieved an average of 76% accuracy. The results of this research suggest that SVM probably is better in speech than textual emotion recognition and integrated modules have better accuracy (81%) than individual ones.

2.4.2 Naïve Bayes classifiers

Naïve Bayes (NB) classifiers are one of the oldest algorithms but still are very successful compared with modern more sophisticated classifiers [25]. It is also one of the most popular methods for text categorization like SVMs despite being said to be simple. The classifier is seen as naïve due to the assumption that features are independent of each other. NB classifiers are considered as supervised learning models that apply the Bayes’ rule:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

Which can be simply written as follows:

$$posterior = \frac{prior \times likelihood}{evidence}$$

Recently in September (2019), a study was carried out on sentiment analysis of Twitter posts (tweets) using NB classifier [26]. The developers of this research tried to make classifications in two different ways. Particularly, for sentiment analysis, there were three classes in text classifications: positive, neutral, and negative. And for recognizing emotions six main emotions were used as classes. The results showed that as the number of classes increased in emotions the accuracy of classifications decreased. Nevertheless, the accuracy of textual classifications using the NB classifier in this research was not reliable enough (66% for sentiments, 47% for emotions). The paper mentions that the reason for such accuracy could be due to the unreliable and unstructured nature of tweets. These tweets most of the time are grammatically incorrect and contain a lot of emoticons and abbreviations where possibly all emotions are expressed at. Thus, to achieve better accuracy for this task the data must be preprocessed more cleverly, especially for Twitter posts.

The paper above also compared NB classifier, SVM, and Random Forest algorithms together on the same dataset of tweets. The outcome of the comparison showed that the NB classifier performed the best while the Random Forest algorithm was around 10% less accurate, and SVM was least accurate for the given dataset.

2.4.3 Deep Learning

The deep learning method is the last method that is going to be discussed in a statistical approach for textual emotion recognition. In very simple words, deep learning tries to imitate how the human brain works and learns to do different tasks using a network of neurons. Deep learning can have a supervised or unsupervised learning model, and similar to SVMs and NB classifiers is a subset of machine learning. In deep learning, the idea of artificial neural networks (ANN) is applied where several connected layers of neurons are communicating with each other to learn a pattern of some task they want to accomplish. Therefore, due to many layers of neurons in deep learning, such models are called "Deep" Neural Networks (DNNs).

Generally, the input layer of a DNN accepts data in a processed vector form and outputs the result from the output layer. In supervised learning, the output would be compared with the expected output and the "weights" of neurons will be modified according to the difference in the outputs. The calculations of the model's error (difference) in predictions are done by the loss functions. Furthermore, activation functions compute and determine whether a neuron should be activated or not and also do the normalization of the outputs between 0 and 1 (or -1 and 1). The weights of neurons define the importance of a neuron in the decision. Between the input and output layer, many "hidden" layers exist which also have neurons and the number of these layers can determine the accuracy of results.

Various very powerful variants of DNNs exist such as Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). DNNs are successfully applied in several fields namely in computer vision [27], speech recognition [28], image recognition [29] and NLP tasks. Particularly, RNNs are more suitable for NLP tasks while CNNs for video recognition tasks. In recent research, a deep convolutional neural network is applied to analyze Twitter messages and movie reviews to recognize their sentiment [30]. This research found out that CNN can be used to improve feature extraction from tweets as they can be very chaotic and as a result achieved state-of-the-art accuracy (85.7% for binary positive/negative and 48.3% for five classes).

Recurrent Neural Network

RNNs were developed in the 1980-1990s by Jeffrey L. Elman (1990) [31] and Paul J.Werbos (1988) [32] to propose a model that could remember past information. Unlike the classic neural networks which learn during training, the RNNs also learn from their own output it generated a moment ago due to the loops inside them. This makes RNNs to be dynamic and capable to "remember" what they have processed previously in time. Therefore, RNNs are helpful in textual emotion recognition where important emotional words in a sentence need to be remembered. Unfortunately, RNN has a "long-term dependency" problem that occurs when the gap between relevant past information and the point it is needed is too big, resulting in an inability to connect data.

Long Short-Term Memory

In 1997, Schmidhuber and Hochreiter proposed Long Short-Term Memory (LSTM) networks [33] that prevent long-term dependency problems of RNN. LSTM is a type of RNN design which is particularly effective in speech recognition and textual emotion recognition. LSTMs have the ability to remember information for a long period by removing or adding information to the memory. This type of network became widely used and is very successful. For instance, one of the biggest companies, Google, reported that they drastically improved their speech recognition for Google voice search by utilizing LSTM networks [34]. Another study by Google (2014) achieved great results in machine translation using LSTM networks [35]. They reported that LSTM did surprisingly well in translating long sentences and it outperformed other approaches.

Furthermore, several modifications of LSTMs exist to improve its outcome for various datasets. A large scale study was done by Klaus Greff et al. (2017) to compare "vanilla" LSTM with eight LSTM variants and find which performs better [36]. The experiments showed that vanilla LSTM performs reasonably well across various datasets and that the modified LSTMs do not improve performance significantly. The LSTM that was used in the project was vanilla.

The common variant of LSTM unit has a complex structure with three gates: **input gate**, **output gate**, and **forget gate**. Essentially, these gates control what is added or removed from the memory. As can be seen in Figure 2.1, information flows through the LSTM network which has multiple units in a chain with their own gates. Further excellent detailed explanations of the LSTM unit and how its gates work is described in a blog by Christopher Olah (2015) [37].

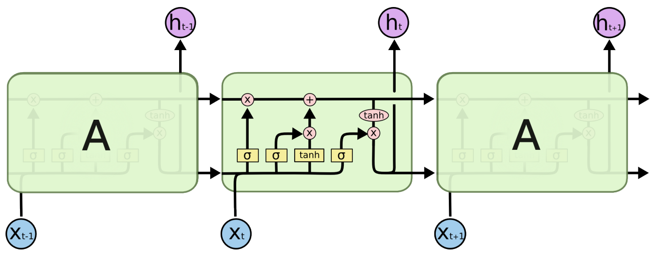


Figure 2.1: LSTM chain of units [37]

2.4.4 Overfitting and Underfitting

One of the most troublesome problems in machine learning which negatively affects the results is an overfitting or underfitting model. To describe a machine learning model's goal shortly, it tries to generalize the problem well enough by training on the data and then attempts to predict the future problems it hasn't seen before. A problem occurs when the model learns the specific noise and patterns of the training data and resulting in false predictions of new never seen data. On the other hand, an underfitting model cannot learn the patterns of both training and new data. Thus, the goal when training a model is always to find a sweet spot between overfitting and underfitting to achieve the best results.

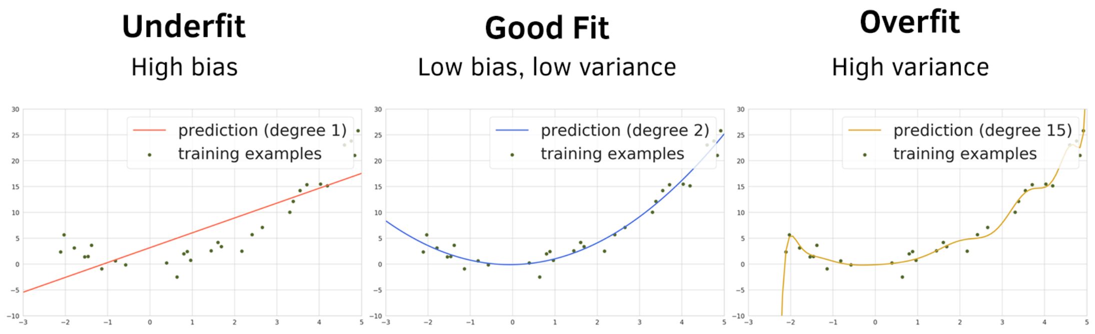


Figure 2.2: Type of Model fit [38]

There are many possible ways to fix an overfitting or underfitting model. Usually addressing an underfit model is easier and can be fixed by training longer, getting more informative features of the data, and complexifying the model [39]. Whereas, overfitting can be avoided by adding more data, simplifying the model, and regularization [40]. One effective regularization approach is called Dropout. Dropout forces neural networks to drop out or ignore some number of neurons in the layers at random with set probability as shown in Figure 2.3. As a result, the model with a new different set of neurons corrects the mistakes from previous layers and reduces the co-dependency of the neurons. Nitish Srivastava et al. (2014) proved that Dropout reduces overfitting and is a better approach compared to other regularization approaches [41]. Additionally, this research showed that it improves the performances of neural networks on various tasks such as vision, speech recognition, document classification, etc.

During the development of the LSTM model of the project, there was strong overfitting in the model. To combat it Dropout was used with different probability ranging

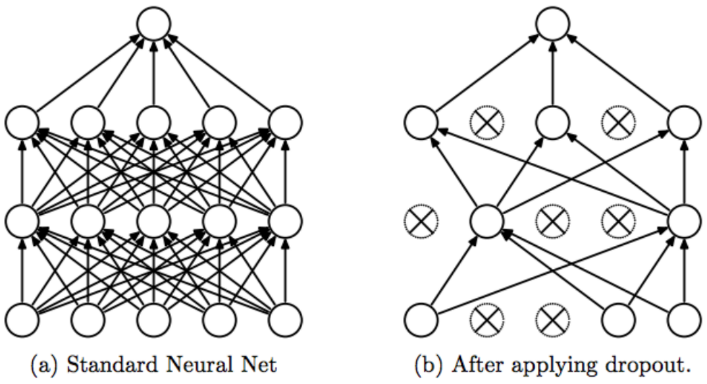


Figure 2.3: Dropout [41]

from 0.1 to 0.6. The best results were found to be around 0.4. The results with details are going to be discussed later.

2.5 Hybrid method

Finally, in the hybrid method, both statistical and knowledge-based methods' concepts are exploited to achieve better performance in emotion recognition. A combination of rich lexicon and classifiers benefits the accuracy of classifications as was found out by a group of researchers [42]. Particularly, they have used the SVM classifier with a keyword detection approach on the classification of blog posts. In another similar research, a hybrid approach was used for emotion detection in texts and achieved 80% precision [43]. Overall, studies suggest that a hybrid approach is also a viable option for emotion recognition and has its advantages.

The chart below summarizes and displays the general map of possible approaches and their models for emotion recognition.

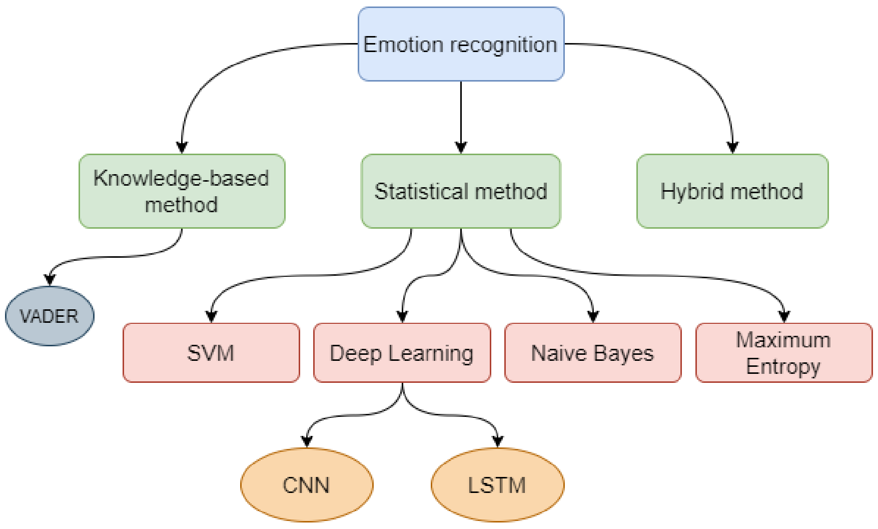


Figure 2.4: Approaches for emotion recognition

2.6 Text preprocessing

Text preprocessing part of emotion recognition is as important as the decision of the classification method. As mentioned earlier, tweets can be chaotic, grammatically incorrect, and contain unnecessary information which makes them unreliable to classify. Thus, it is crucial to clean and process these texts before the classification part. To accomplish this, several techniques exist such as:

- Stop-word removal: removing irrelevant words from the sentences such as ”the”, ”a”, ”an”, ”me”, ”in”, etc.
- Tokenization: breaking up a document into tokens like words or sentences.
- Lower case conversion: identical words can have different cases which then are considered as different words in recognition.
- Stemming: chopping off prefixes of words to have only common root form of the word. However, after stemming the actual meaning of the words might be lost.
- Lemmatization: similar to stemming but it transforms words into their dictionary root form.

Additionally, most of the tweets used in training, contain URLs (usually link to an image), mentions (a reply or tagging of some username) and symbols such as ”.”, ”-”, ”?”, etc. This information does not underline the emotions stored in the tweets, therefore, it was better to remove them.

Besides, these techniques can reduce the dimensionality of the dataset and increase the efficiency of the system. However, not all techniques can positively improve the accuracy of recognition, namely, stemming and lemming can have negative effects on text categorization in some cases [44]. A study analyzed all the preprocessing techniques and found out that there are no unique combinations that would work perfectly fine for various datasets [45]. Therefore, it is suggested to experiment and analyze which techniques work best for a specific task.

2.7 Feature extraction

Features are the unique properties of a text, word, or sentence in the dataset. These features must be extracted from the dataset in a numeric vector form because most of the time algorithms' optimization only works on numeric data. The most common and effective feature extraction methods for text classification are models such as Bag of Words and TD-IDF, and Word Embedding.

2.7.1 Bag of Words

The Bag of Words (BoW) model is the most convenient and simple method for feature extraction. This model converts a document into a vector of numbers that represents the frequency of all unique words that occurred in the document. Thus, the weight of each word in the vector is identical to the frequency of the word in a document. To distinguish which words are present in a document, each word has its special index/position assigned in a vector. Therefore, the length of each vector is equal to the length of vocabulary for a given dataset. An example will be illustrated further to help visualize the model. Let's assume there are these three documents in our corpus:

The weather is sunny today.
The weather is rainy tomorrow.
The weather was rainy yesterday.

There are nine unique words in this corpus: *The, weather, is, was, sunny, rainy, today, tomorrow, yesterday.* So, the length of vectors for each document will be also nine. Let's also assume the index for each unique is assigned in order as they are written above. So, *the* is assigned to first index in the vector, *weather* second, ..., *yesterday* ninth. Hence, the vector representations of all documents will look as such:

The	weather	is	was	sunny	rainy	today	tomorrow	yesterday
1	1	1	0	1	0	1	0	0
1	1	1	0	0	1	0	1	0
1	1	0	1	0	1	0	0	1

The drawback of the BoW model is that the small weight of some words may imply that these words are not as important as the words that have occurred a lot. However, this is not true because less frequent words may express stronger emotions and can be overlooked in the BoW model. The additional major issue this model has is related to vectors' size and sparsity. In the example above the length of vocabulary and vectors is only nine and it may not be a problem at this point. However, with larger datasets, the length of vocabulary increases, producing very big vectors with a length of $> 50,000$. Moreover, since the length of documents usually is much smaller than $50,000$, the vectors become sparse and will contain a lot of zeros in between the ones and twos which are needed. Finally, the model does not take context or semantics into account and loses the meaning of the word in the document. This happens because the words are assumed to be independent of each other by ignoring the important co-occurrence of the words.

2.7.2 TF-IDF

The TF-IDF model, which stands for Term Frequency-Inverse Document Frequency, is another feature extraction method that tries to solve the BoW's problem by increasing the weight of less frequent (important) words and decreasing the more frequent ones. First, to find the weight of a word, the Term Frequency of a word is calculated. This is done by counting the frequency of that word in a document then dividing by the length of the document as a way to normalize the weights, since a word may occur more in a longer document compared to a shorter one:

$$\textbf{TF} = \frac{\textbf{occurrence of a word in a document}}{\textbf{number of words in a document}}$$

Next, the Inverse Document Frequency determines how important or rare a word is across the dataset. To calculate it the total number of documents is divided by the number of documents a given word occurred in and the result is logarithmically scaled:

$$\textbf{IDF} = \log \frac{\textbf{total number of documents}}{\textbf{number of documents with the word in it}}$$

Finally, the obtained TF and IDF are multiplied together to produce the weight of a given word in the document:

$$\textbf{TF} \times \textbf{IDF}$$

The vector representation using this model is similar to the BoW model but considers the importance of the words. Thus, the TF-IDF also has the same size and semantics problem as described in the BoW model.

Due to the TF-IDF being quite effective in finding a relevant document for a given query, its variants are often used in recommender systems and search engines. In fact, more than 80% of recommender systems in automated libraries use TF-IDF [46].

2.7.3 Word Embedding

Finally, Word Embedding is probably the best way to extract the features today. Similar to TF-IDF and BoW models, Word Embedding tries to represent the words using a vector of numbers but has a semantic advantage over them. As explained earlier, the other two models assume that the words are independent of each other causing the semantics of the words to be ignored. Whereas, in Word Embedding, similar words are grouped or have similar representations. Additionally, the real-valued vectors that are assigned to each word are considerably denser than the sparse representation of the other two models.

Today, neural networks are mostly used in Word Embedding techniques which improve the training time of the vectors. One such Word Embedding technique is created by Tomas Mikolov et al. (2013), called **word2vec** [47, 48]. They've reported that the model trains on data several times bigger (corpora with billion of words) and is more accurate than the older models. An interesting discovery they made was how the vector representation of different words could be used to calculate the next semantically closest word. For instance, they've shown how the word *smallest* can be found using the following operations: $vector("biggest") - vector("big") + vector("small")$. Additionally, the subtle word relationships between cities and countries are also taken into account. For instance, the closest word for $vector("Germany") + vector("capital")$ is $vector("Berlin")$. The word2vec toolkit can use two different learning models, Continuous Bag of Words (CBOW) and Continuous Skip-Gram. According to the developers, CBOW is faster but skip-gram performs better with uncommon words. The CBOW model predicts the current word by its context while skip-gram predicts the surrounding words by the given word. Apart from word2vec there are also other Word Embedding techniques developed such as **GloVe** [49], **Gensim** [50], **ELMo** [51] and others.

However, Word Embedding also has its limitation, particularly, it does not distinguish the homonyms (words with multiple meanings). For instance, in a sentence "*She will $\underline{park}$ the car so we can walk in the $\underline{park}$*", the word *park* has two meanings, but when the word is embedded it will have a single representation. Thus, multi-sense embeddings exist that try to represent multiple senses of the words. For instance, Multi-Sense Skip-Gram assumes there are a specific number of meanings a word has and computes embeddings for each meaning [52]. Most Suitable Sense Annotation combines a lexical database such as WordNet and ConceptNet, disambiguates, and labels the senses of the words [53].

Overall, Word Embedding is seemingly the better choice for emotion classification of text compared to the other two feature extraction models. The main reason is that it captures the meaning of the words which is crucial for NLP. While it demands more memory because it captures more information on each word, this information helps to understand the context. Whereas, the other two models' vector representations are sparse that contain a lot of meaningless zeros.

3. Design

3.1 Main decisions

As described in the chapter above, there are several approaches to emotion recognition and each with different techniques. Additionally, the cleaning of texts and feature extraction methods play a major role in this task. Therefore, design decisions were made that determined the outcome of the project. The Figure 3.1 below displays the main steps that must be followed to implement the project's algorithm.

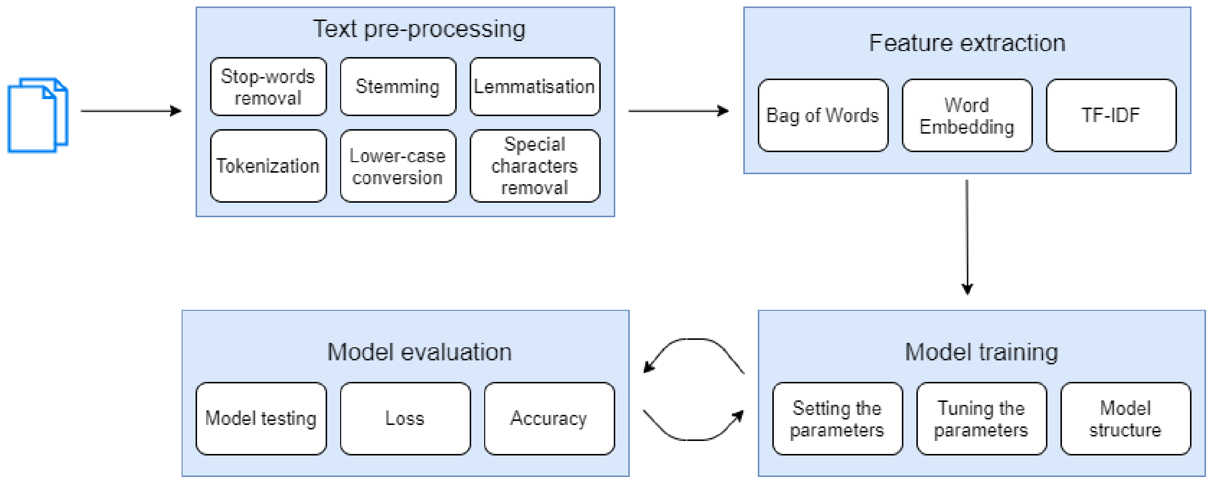


Figure 3.1: Main steps

From the start, this project's objective was to research how textual emotion recognition can be done using a statistical method, specifically, with a deep learning technique. Therefore, during the development of the project, a decision was made to accomplish this task using the LSTM networks. As established earlier, LSTMs are powerful in NLP tasks because they could remember important emotional words and "forget" the irrelevant ones. Thus, it was worthwhile to discover the outcome of using this model.

For the text preprocessing step, all techniques except stemming are used to process and clean the texts. Instead of stemming, lemmatization of words is done due to stemming's flaw as explained earlier. Additionally, special characters such as "$", "@", are removed because they don't have relevant information.

For the feature extraction step, initially, the Bag of Words model was going to be used due to its simplicity and capability. However, due to the BoW model limitations, the TF-IDF model and Word Embedding were used and compared. The difference in results will be discussed later.

The two steps above are going to be performed on two sets of data: training data and testing data. Next, the LSTM model is going to be implemented and trained using the processed dataset. The model's structure consists of an LSTM layer, a regular neural network layer for giving output, and a Word Embedding layer if it is used. Finally, the trained model is going to be evaluated and analyzed how the results change when a different combination of feature extraction and text preprocessing methods are used.

3.2 Datasets and Classes

In general, there were two datasets used in training, and both contained tweets. Analysis of tweets was mostly carried out in the project because they are more accessible which means more samples and a variety of emotions. Additionally, they are a little harder to classify due to the strong noise each tweet usually has (symbols, misspelled words, and shorter sentences), and therefore more interesting to tackle. The first dataset called Sentiment140 [54] has 1.6 million tweets with two classes: positive and negative. This dataset was very helpful for binary classification because this many samples can improve the accuracy. The other dataset had ”only” 40,000 samples/tweets but with 13 classes which are shown in Figure 3.2 below with the amount each has. On the other hand, training on and classifying so many labels hurts the accuracy of the model as the research in the previous chapter showed. Thus, the samples were grouped into three classes: positive, negative, and neutral. This way the model can achieve more meaningful results and it's just easier to start with fewer classification categories.

Classifying the six main emotions was performed using the dataset with 13 classes. However, the dataset doesn't have all of the main emotions and has a small number of samples for some of the emotions. Therefore, such classification was carried out on six classes with the most samples in this dataset except surprise, namely, neutral, worry, happiness, sadness, love, and fun. The surprise was not considered due to the samples being ambiguous and overlapping with other emotions.

3.3 Desktop Application

A desktop application was implemented to demonstrate the classifications of the models. Though the application had the lowest priority in the development of the project and was done in a short period, and because of that, the application has limited functionalities. The application was implemented using **Tkinter** which is the Python's standard toolkit for creating GUIs [55]. The application can be started by running its Python file in the console in standard way. Essentially, the application can take the user's input and predict the class of the input using one of three pre-selected classifiers: binary, ternary, and 6-class.

First, the trained models are saved and then loaded into the app. The user would write a sentence in the text box and this sentence's emotion is predicted by the loaded model and will be shown in the app. However, the text should be preprocessed and integer

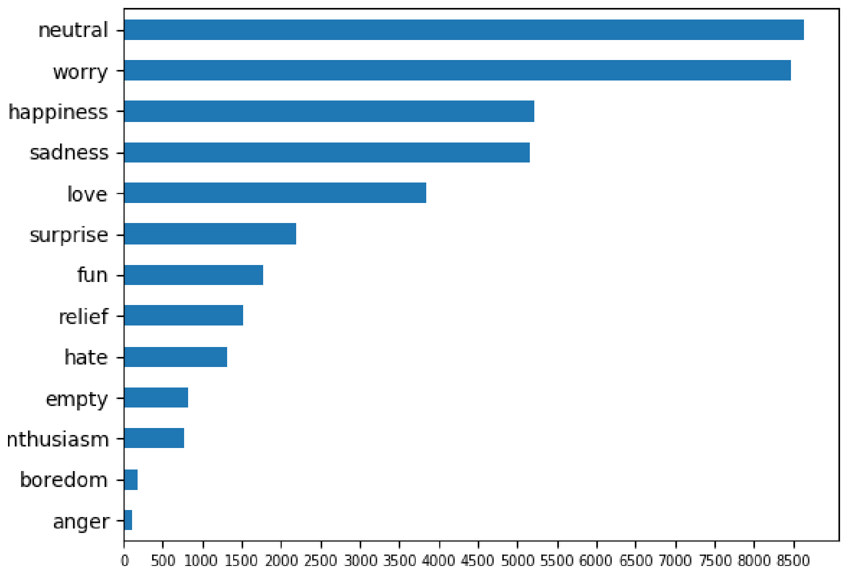
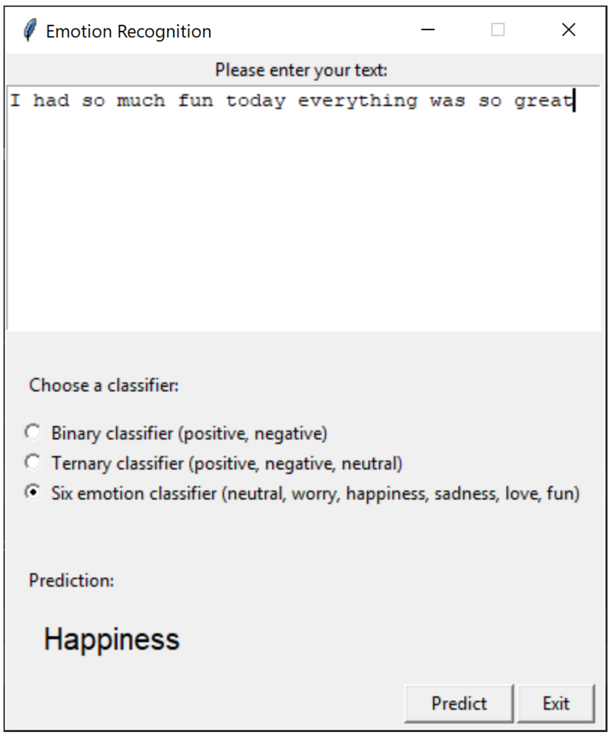


Figure 3.2: 2$^{\text{nd}}$ dataset's labels and size

encoded so that model recognizes it. Thus, during training, the tokenizer which integer encodes the words is saved with its dictionary of encodings and is loaded into the app to encode a new text. Before that, the text is preprocessed using the same techniques that are used during the training of the classifiers. Since each classifier has its saved model and tokenizer, all three pairs had to be loaded and used depending on the corresponding chosen option of classification. The screenshot below shows the only page of the application and its design.



Please enter your text:

I had so much fun today everything was so great

Choose a classifier:

- Binary classifier (positive, negative)
- Ternary classifier (positive, negative, neutral)
- [✓ Six emotion classifier (neutral, worry, happiness, sadness, love, fun)]

Prediction:

Happiness

Figure 3.3: The application

4. Implementation

The implementation done in the project is explained below. It was recommended by the supervisor of the project that the implementation is done on Python programming language since it has immense and very effective machine learning libraries that are going to be used for this system. The three main steps namely, text preprocessing, feature extraction, and model structure were implemented to make classifications of the tweets.

4.1 Loading the dataset

First, the corpus of tweets is read from a `.csv` (comma separated values) file that contains the tweets and their labels, apart from the unnecessary values such as usernames, tweet IDs, date, etc., that do not help in emotion classification. The dataset is saved into a Dataframe data structure from **Pandas** library [56]. The Dataframe was helpful for convenient `.csv` file reading, value modifications and dataset transformation into the **NumPy** data structure which usually is the preferred data structure for machine learning due to its efficiency [57].

Listing 4.1: Reading the dataset

```
1 df = pd.read_csv('datasets/text_emotion.csv', delimiter=',')
2 df = df[~df["sentiment"].str.contains('surprise')]
3 # label codes for encoding the labels:
4 label_codes = {"sentiment": {"worry": 1, "hate": 1, "sadness": 1, "boredom": 1, "anger": 1, "relief": 0, "happiness": 0, "enthusiasm": 0, "love": 0, "fun": 0, "neutral": 2, "empty": 2}}
5 df.replace(label_codes, inplace=True) # replace labels with codes
6 print(df["sentiment"].value_counts()) # prints the number of samples each label has
7 dataset = df.to_numpy()  # transform into numpy
```

As shown in Figure 3.2, one of the datasets has too many classes and some of them had too little samples. Therefore, they were grouped into three classes, positive, negative, and neutral for one of the classifiers as shown in lines 4 and 5 in Listing 4.1. Line 2 removes the tweets which are labeled as "surprise" because they were found to be confusing and can be put in more than one of the three classes.

4.2 Text preprocessing

The most of preprocessing part were done in the function `preproc()` as shown in Listing 4.2. As explained in the Text preprocessing section, techniques were used to remove irrelevant information such as URLs, usernames, special characters, numbers, and stop-words. However, removing all of the stop-words changes the meaning of the sentences because negating words such as "not", "can't", "wasn't", etc., are also considered as stop-words. For instance, a simple phrase "I am not happy" without stop-words becomes "happy" indicating that the phrase is positive which is not true. Therefore, it was made sure that such stop-words are not removed and as a result, the model's accuracy gained 1-2%. The words were also tokenized, lowercased, and lemmatized.

Listing 4.2: Preprocessing of the tweets

```python
def preproc(data):
    ... ... ...
    for ind in range(len(train_x)):
        # Remove URLs, emojis, and usernames
        train_x[ind] = p.clean(train_x[ind])
        # Removing special characters and numbers
        train_x[ind] = re.sub('[^A-Za-z ]+', '', train_x[ind])
        # Tokenization
        train_x[ind] = word_tokenize(train_x[ind])
        # Normalizing case (lower case)
        train_x[ind] = [word.lower() for word in train_x[ind]]
        # Removing stopwords
        train_x[ind] = [w for w in train_x[ind] if w not in ↩
            stopwords]
        # Stemming
        # train_x[ind] = [stemmer.stem(word) for word in train_x[↩
            ind]]
        # Lemmatization
        train_x[ind] = [lemmatizer.lemmatize(word) for word in ↩
            train_x[ind]]
        # Removing single characters
        train_x[ind] = [i for i in train_x[ind] if len(i) > 1]
        # Add the words to vocabulary
        vocab.extend(train_x[ind])

    ... ... ...
    maxlen = len(max(train_x, key=len))  # max length of sentences
    vocablen = len(vocab)

    return train_x, train_y, vocablen, maxlen
```

It can be seen that the `preproc()` function also returns the length of vocabulary and the longest sentence. This is because they are needed to encode the dataset in the `encode()` function as shown in Listing 4.3. The words are integer encoded (each unique word is assigned to a unique number) because the Word Embedding layer requires the words to be in a unique numeric form. After integer encoding, the vectors are pre-padded with zeros to make each vector's size of one length since some sentences are longer than others. The reason why vectors are pre-padded and not post-padded is that a study showed that LSTM classifiers perform better with pre-padding [58]. Indeed, the model's accuracy increased by roughly 1%. Finally, the labels are one-hot encoded which means that they're represented as a vector sort of similar to BoW's representation. For instance, if there are three classes/labels the vectors will look as so:

[positive, negative, neutral]

[0, 0, 1]

[0, 1, 0]

[1, 0, 0]

Listing 4.3: Encoding the dataset

```python
def encode(x, y, vocablen, maxlen, num_classes):
    ... ... ...
    tk = Tokenizer(num_words=vocablen, lower=False)
    tk.fit_on_texts(x)

    x_encoded = tk.texts_to_sequences(x) # integer encoding the words
    x_padded = pad_sequences(x_encoded, maxlen=maxlen, value=0, padding='pre') # pre-padding the vectors
    y_encoded = to_categorical(y, num_classes=num_classes) # one hot encoding the labels
    ... ... ...
    return x_padded, y_encoded
```

4.3 Model structure and parameters

Finally, LSTM implementation is explained further. The Python library that helped in creating the LSTM classifier was **Keras**, which has useful layers for machine learning such as LSTM, Word Embedding, and regular deeply connected neural network (Dense) [59]. Firstly, line 3 in Listing 4.4, states that the model is a linear stack of layers. Next, a Word Embedding layer is added with set parameters: `input_dim` the length of the vocabulary; `input_length` the length of the inputs (`maxlen`); `output_dim` the size of the vector space.

After the embedded vectors are created, the input is passed to the LSTM layer with 32 units, 0.4 dropout rate, and 0.4 recurrent drop which is for dropping the connections between the units. The last Dense layer is added as an output layer to make predictions. If the binary classification of the bigger dataset is performed the `binary_crossentropy` loss function and `sigmoid` activation function is used, while `categorical_crossentropy` loss function and `softmax` activation function for multi-class classifications. Finally, the optimizer "Adam" is chosen with a set learning rate. The optimizers optimize the weights of the model based on the error given by loss function.

Listing 4.4: The LSTM model

```python
def create_model(input_dim, input_length, num_classes):

    model = Sequential()
    model.add(Embedding(input_dim=input_dim, input_length=input_length, output_dim=32))
    model.add(LSTM(32, dropout=0.4, recurrent_dropout=0.4))
    model.add(Dense(num_classes, activation='sigmoid'))
    adam = keras.optimizers.Adam(lr=0.0002)
    model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

    return model
```

4.4 Application

The application was implemented using widgets from **Tkinter's** toolkit on Windows Operating System. The most of app code is placing and naming those widgets such as buttons, radio buttons, labels, and text boxes. The main functionality (Listing 4.5) is assigned to the "Predict" button by attaching a command to it when pressed. When a user clicks on the button, a sentence written in the textbox is saved and preprocessed (lines 2 and 3), after that the state of radio buttons is saved to see which one is chosen. The `if` block corresponding to the chosen classifier is run where the input is integer encoded (line 10) and predicted (line 12) using the corresponding tokenizer and model, respectively. Finally, the predicted label is shown in the app by changing the "text" of the "output label" to one of the classes (positive, neutral, etc.).

Listing 4.5: The "predict" function

```python
1      def predict(self):
2          sentence = self.textbox.get("1.0", 'end-1c') # get ...
3          inp = preproc(sentence)     # ... preprocess the input
4
5          # the state of radiobuttons (1, 2, or 3):
6          chosenClassifier = self.state.get()
7
8          if chosenClassifier == 1:    # If chosen BINARY CLASSIFIER
9              # encoding the input
10             x_encoded = tokenizer_2.texts_to_sequences([inp])
11             x_padded = pad_sequences(x_encoded, maxlen=22, padding='pre', value=0)  # pre-padding the vectors
12             ynew = model_2.predict_classes(x_padded)
13
14             if ynew == 0:
15                 self.output['text'] = 'Positive'
16             elif ynew == 1:
17                 self.output['text'] = 'Negative'
18         ...  ...  ...
```

5. Evaluation and Results

The following chapter will discuss the achieved results by the LSTM classifier with different combinations of parameters and techniques. For evaluating the model's accuracy, the datasets were split into training and testing sets. While 70% of the dataset was used in training, the rest of the dataset (30%) was to validate the model during training and to evaluate the model after training.

5.1 Parameters and Techniques

First, as explained earlier the bigger dataset with 1.8 million samples was used for binary classification since it only contains two classes namely, positive and negative. Thus, model's loss function is required to be `binary_crossentropy` with the `sigmoid` activation functions. Although, there are other activation functions such as `relu` or `softmax`, `sigmoid` is the most suitable for the output layer of a binary classifier because the range of the `sigmoid` is $[0, 1]$ and we have only two classes encoded as 0 and 1. The `softmax` function can also be used in such a case, but it's more suitable for multi-class classifications and the experiments showed that both produce more or less similar results. This is due to `softmax` being an extension of `sigmoid` for multi-class classifications.

For feature extraction, two methods were tested TF-IDF and Word Embedding. The BoW model was planned to be tested initially but soon realized that TF-IDF is the better model for the classifier's task. As explained in the Feature Extraction section, Word Embedding excels in capturing the context of the words, therefore it should perform better compared to the TF-IDF model. Indeed, the experiments showed that Word Embedding gives better results with 79% accuracy compared to TF-IDF's 77%:

Next, for the preprocessing part, the stop-words removal and padding of the vectors were tested. As explained in the previous chapter, pre-padding helped to increase the accuracy by roughly 1%, while the adjusted stop-words removal increases the accuracy by 1-2%. Additionally, when choosing and testing the lemmatization and stemming techniques, it was found out that lemmatizing the word slightly improved the performance.

Finally, for tuning the parameters of the model manual testing and grid searching were performed. Grid search is a method in which the range of parameters is given and it searches for the combination of parameters that give the best results. Manual testing also helped to see in what direction the accuracy improved or worsened when the parameters are increased or decreased, respectively. Word Embedding layer has one parameter to tune - output dimension. When increasing it to a higher number [80-200] the accuracy would not increase but the training time increased drastically and setting it too low [1-16] hurt the improvement between each epoch. Thus, the best dimension was found to be around 32 and confirmed with a grid search.

Next, the LSTM parameters were tuned namely, the number of units and Dropout rate. The number of units had a similar trend as output dimension of embeddings, with few units [1-10] the model's learning each epoch was worse while with too many units [100-200] training time increased without better performance. Combined with the grid search, it was determined that the best number of LSTM units was 32. The dropout rate was also analyzed and determined that the rate higher or lower than 0.4 hurt the performance. Dropout helped to minimize the overfitting but the model would still overfit after some number of epochs (described further).

The final model parameters were optimizers and their learning rate. **Keras** has multiple optimizers such as Stochastic Gradient Descent (SGD), Adam, Adadelta, Adagrad, etc. Each optimizer was tested with different learning rate and their best results are shown in the table below:

	Adam	**SGD**	**Adadelta**	**Adagrad**	**RMSprop**
Accuracy	79 %	78 %	78 %	78 %	76 %
Loss	0.43	0.45	0.44	0.46	0.47

Although some optimizers provided good accuracy, the training accuracy and loss was worse than Adam. Therefore, Adam was chosen as the best optimizer and its learning rate was grid searched, and the best rate determined to be 0.0002. Finally, the batch size which is the number of samples to be propagated through the network each iteration (in one epoch) was tuned. Experiments showed that smaller batch size increases the training time but the model converges faster and achieves better accuracy. Hence, different sizes were grid searched and the best size was found to be 8.

The table below summarises the parameters and techniques which achieved highest accuracy - 79%:

Padding	"Pre"
Feature extraction	Word Embedding
Word inflection	Lemmatisation
Embedding output dimension	32
LSTM units	32
Dropout	0.4
Recurrent Dropout	0.4
Optimizer	Adam
Learning rate	0.0002
Batch size	8

For the classification of three classes, the same parameters and techniques were used and the highest achieved accuracy and loss was 61% and 0.87, respectively. Whereas, for six emotion classification the model achieved 41% accuracy and 1.48 loss. The table below summarizes the results of all three classifiers:

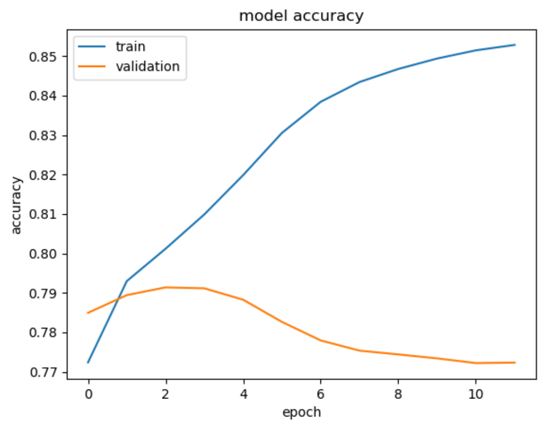
	Binary	**3-class**	**6-class**
Accuracy	79%	61%	41%
Loss	0.44	0.87	1.48

When evaluating the results of six emotion classifications, a discovery was made while analyzing the confusion matrix below. The diagonal elements show the correct classifications while the rest of the elements show the false predictions of labels on the left column with labels on the top row. The matrix shows that the classifier mostly correctly classifies happiness and neutral but also confuses them with each other. However, the classifier also strongly confuses love with happiness and neutral. As can be seen in the matrix, 1731 and 470 love sentences are falsely predicted as happiness and neutral, respectively. The fun and sadness false predictions have a similar trend. Finally, worry is confused with happiness and neutral. The matrix explains how the overall accuracy is 41% and the reason for this might be the ambiguity of the sentences. For instance, love and fun most of the time also mean happiness but the classifier cannot make the distinction. Additionally, the tweets don't provide enough information to make confident and correct classifications.

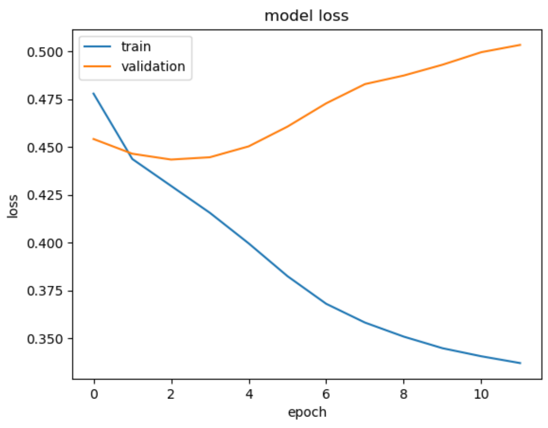
	neutral	**worry**	**happiness**	**sadness**	**love**	**fun**
neutral	**2286**	159	839	2	4	0
worry	814	**191**	503	2	4	0
happiness	963	123	**1934**	1	8	0
sadness	242	73	168	**6**	2	0
love	470	53	1731	2	**59**	0
fun	395	53	490	1	4	**0**

5.2 Overfitting Model

One of the biggest problems in the project was fixing the overfitting of the model. Generally, all classifiers would learn during the first three to five epochs improving the validation accuracy and loss. However, from then on the performance would start to get worse implying that the model is overfitting as shown in Figure 5.1 below. There are few ways to prevent the model from overfitting as described earlier namely, adding more training data, simplifying the model, and adding Dropout. However, the dataset already had more than enough number of samples, the model was already simple and Dropout had an only marginal positive effect. Additionally, an attempt was made to complexify the model by adding more LSTM layers and Dense layers to see if it would have helped, but it only negatively affected the performance. Hence, the final model structure yielded the best results. This is was a serious obstacle in the project because many weeks were spent to understand why the model is overfitting, how to prevent it, and try different solutions (by experimenting).



(a) Accuracy



(b) Loss

Figure 5.1: Training and validation plots

5.3 Conclusion

Overall, these achieved accuracies do not improve the current state-of-the-art results. There are few assumptions for such low performances. First, Twitter data is too noisy, has too many grammatical errors and jargon words. This is proven by checking the number of words generated in the dictionary of the binary classifier which is roughly 410,000 words. The dictionary with this many words contains a word with different grammatical errors multiple times. Such inconsistency will negatively affect the performance of the model. During the development of the project, it was considered to grammatically correct each word using a dictionary to decrease the number of semantically identical words. However, experiments showed that this increases the preprocessing time several times and could not be accomplished due to time constraints.

Next, the length of tweets is very short compared to a dataset of movie or product reviews. The average length of tweets in the datasets was roughly 14 words and even less emotionally useful words. If each sample had more meaningful words it would be mean more useful information for the classifier to learn from.

Finally, the LSTM model developed in the project was not complex enough to make better classifications. The studies shown in this paper have models with different modifications to the feature extraction methods or to the model itself which focuses better on the task and makes better classifications.

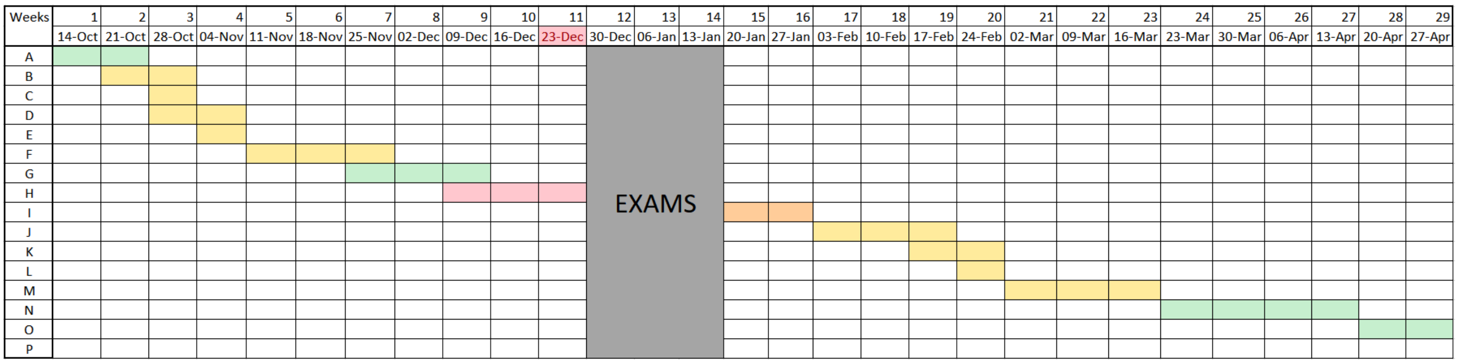
6. Progress

6.1 Project Management

Overall, the research and implementation done in the project is satisfactory but should have been better. The main tasks of the project were accomplished and a few were modified due to the changed priorities namely, the desktop application (lowest priority). The timeline of the tasks was seriously altered due to the unexpected heavy workload from the other modules' courseworks in the first semester and the pandemic in the second semester. As a result, the deadlines of the Interim Report and Dissertation were extended and the actual completion date of the reports was severely behind the planned schedule. Therefore, most of the Dissertation was written in a short period which was not enough for a report of this size. The pandemic forced to readjust the project's tasks and adapt to the difficult and new working environment away from campus.

6.2 Timeline

Initially, the timeline and the remaining tasks of the second half of the project were newly set due to the additional hours of research on the topic and the affected timeline in the first half. However, after pandemic hit unexpectedly the timeline of the second half had to be modified again. The first task that had to be changed in the first semester is task C (Figure 6.1) due to realizing the focus of the project was not on developing software but on the research of emotion recognition. Next, task F had to be pushed (and prototype removed) to the second semester because the unexpected workload from other modules forced to. The research in the first semester also helped to understand the problem better and therefore, the second semester's tasks for developing the classifier were accurately specified (tasks I, J, K, and L, in Figure 6.2). The unforeseen problem occurred during the second half of the project which consumed too much time. The problem was the overfitting models that had to be fixed in the middle of the other tasks (shown in Figure 6.3, task M). Finally, the Final Timeline shows that the Dissertation was written in a shorter time than as planned previously.



Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	14-Oct	21-Oct	28-Oct	04-Nov	11-Nov	18-Nov	25-Nov	02-Dec	09-Dec	16-Dec	23-Dec	30-Dec	06-Jan	13-Jan	20-Jan	27-Jan	03-Feb	10-Feb	17-Feb	24-Feb	02-Mar	09-Mar	16-Mar	23-Mar	30-Mar	06-Apr	13-Apr	20-Apr	27-Apr
A	🟩	🟩										▨	▨	▨															
B		🟨	🟨									▨	▨	▨															
C			🟨									▨	▨	▨															
D			🟨	🟨								▨	▨	▨															
E				🟨								▨	▨	▨															
F					🟨	🟨	🟨					▨	▨	▨															
G							🟩	🟩	🟩			▨	▨	▨															
H									🟥	🟥	🟥	▨	EXAMS	▨															
I												▨	▨	▨	🟧	🟧													
J												▨	▨	▨			🟨	🟨	🟨										
K												▨	▨	▨					🟨	🟨									
L												▨	▨	▨						🟨									
M												▨	▨	▨							🟨	🟨	🟨						
N												▨	▨	▨										🟩	🟩	🟩	🟩		
O												▨	▨	▨														🟩	🟩
P												▨	▨	▨															

Figure 6.1: Original Timeline

🟩 A - Complete Project Proposal.

🟨 B - Research previous work on emotion recognition using text analysis approach.

🟨 C - Define software specifications of the project.

🟨 D - Find, set up and familiarise with the Python workspace

🟨 E - Collect and pre-process the dataset.

🟨 F - Start to design the initial neural network and the visual prototype of the desktop application.

🟩 G - Write the Interim Report (Deadline 16th of December).

🟥 H - Work on other courseworks and prepare for exams.

🟧 I - Holidays.

🟨 J - Finish designing the network and the algorithm.

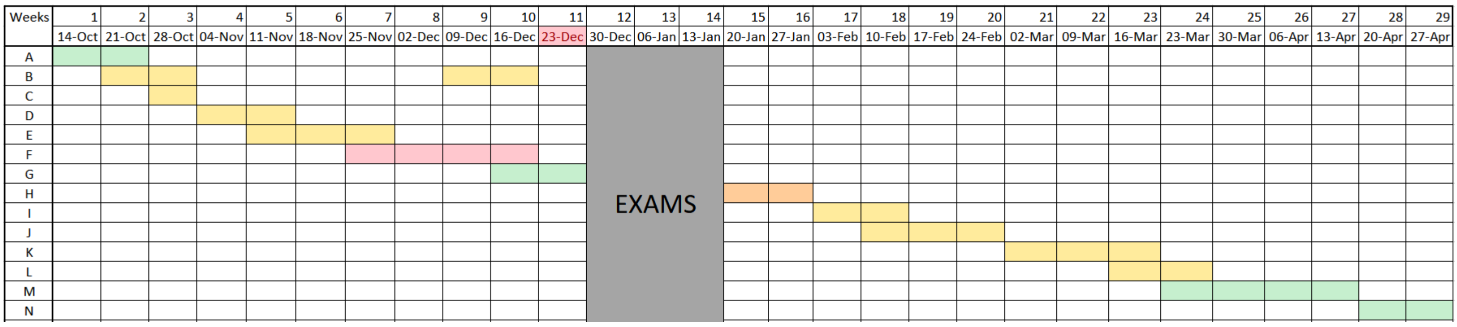
🟨 K - Train the model.

🟨 L - Finalise and evaluate the model and tune the parameters to achieve higher accuracy.

🟨 M - Implement a desktop application and integrate the algorithm.

🟩 N - Write the Final Dissertation (Deadline 20th of April).

🟩 O - Work on the presentation/demonstration.



Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
	14-Oct	21-Oct	28-Oct	04-Nov	11-Nov	18-Nov	25-Nov	02-Dec	09-Dec	16-Dec	23-Dec	30-Dec	06-Jan	13-Jan	20-Jan	27-Jan	03-Feb	10-Feb	17-Feb	24-Feb	02-Mar	09-Mar	16-Mar	23-Mar	30-Mar	06-Apr	13-Apr	20-Apr	27-Apr
A	🟩	🟩										▨	▨	▨															
B		🟨	🟨						🟨	🟨		▨	▨	▨															
C			🟨									▨	▨	▨															
D				🟨	🟨							▨	▨	▨															
E					🟨	🟨	🟨					▨	▨	▨															
F							🟥	🟥	🟥	🟥		▨	▨	▨															
G										🟩	🟩	▨	▨	▨															
H												▨	EXAMS	▨	🟧	🟧													
I												▨	▨	▨			🟨	🟨											
J												▨	▨	▨				🟨	🟨	🟨									
K												▨	▨	▨							🟨	🟨	🟨						
L												▨	▨	▨									🟨	🟨					
M												▨	▨	▨										🟩	🟩	🟩	🟩		
N												▨	▨	▨														🟩	🟩

Figure 6.2: Actual Timeline after first semester

🟩 A - Complete Project Proposal.

B - Research previous work on emotion recognition using text analysis approach.

C - Define the objectives of the project.

D - Find, set up and familiarise with the Python workspace.

E - Collect, pre-process and extract features from the dataset.

F - Work on other courseworks.

G - Write the Interim Report (Deadline extended: 27th of December).

H - Holidays.

I - Start designing the LSTM network.

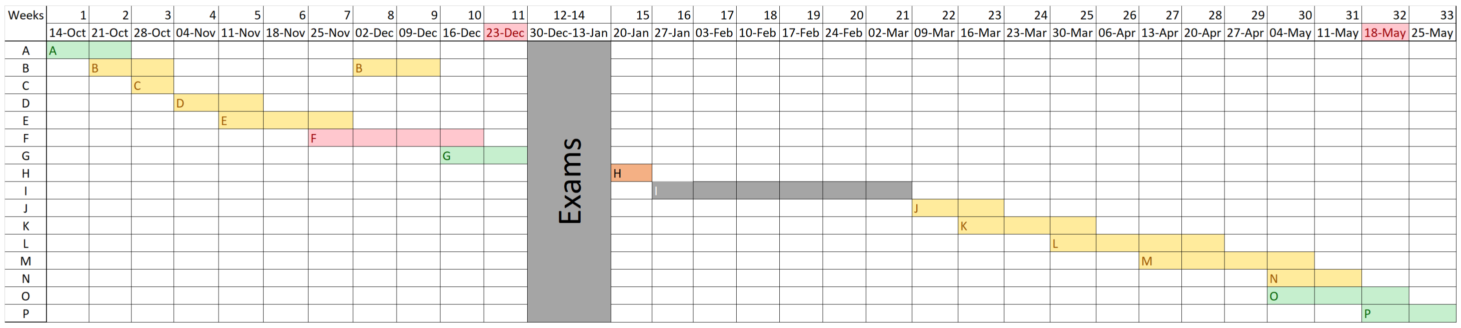
J - Train and evaluate the model.

K - Try different classifiers and combination of preprocessing techniques, and compare the results.

L - Finalise the variant of the model and tune the parameters to achieve higher accuracy.

M - Write the Final Dissertation (Deadline 20th of April).

N - Work on the presentation/demonstration.



Weeks	1	2	3	4	5	6	7	8	9	10	11	12-14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	14-Oct	21-Oct	28-Oct	04-Nov	11-Nov	18-Nov	25-Nov	02-Dec	09-Dec	16-Dec	23-Dec	30-Dec-13-Jan	20-Jan	27-Jan	03-Feb	10-Feb	17-Feb	24-Feb	02-Mar	09-Mar	16-Mar	23-Mar	30-Mar	06-Apr	13-Apr	20-Apr	27-Apr	04-May	11-May	18-May	25-May
A	A	🟩																													
B		B	🟨					B	🟨																						
C			C																												
D				D	🟨																										
E					E	🟨	🟨																								
F							F	🟥	🟥	🟥																					
G										G	🟩																				
H												Exams	H																		
I														I	⬛	⬛	⬛	⬛	⬛												
J																				J	🟨										
K																					K	🟨	🟨								
L																							L	🟨	🟨	🟨					
M																									M	🟨	🟨	🟨			
N																												N	🟨		
O																												O	🟩	🟩	
P																														P	🟩

Figure 6.3: Final Timeline

A - Complete Project Proposal.

B - Research previous work on emotion recognition using text analysis approach.

C - Define the objectives of the project.

D - Find, set up and familiarise with the Python workspace.

E - Collect, pre-process and extract features from the dataset.

F - Work on other courseworks.

G - Write the Interim Report (Deadline extended: 27th of December).

H - Holidays.

I - Adapting to changes

J - Design the LSTM network.

K - Train and evaluate the model.

L - Try and compare different classifiers and combination of preprocessing techniques.

M - Fix the overfit models

N - Finalise the variant of the model and tune the parameters to achieve higher accuracy.

O - Write the Final Dissertation (Deadline extended: 18th of May)

P - Work on the presentation/demonstration.

6.3 Challenges

The Final Timeline also shows a big period that was not used for developing the project (task I). The pandemic introduced a few challenges that forced to adapt to changes and find a way to resume development. First, all of the implementation and research completed in the first semester were saved on a personal local computer on the campus dormitory, and it had to be retrieved to resume the project. Thanks to the project's supervisor the source code was retrieved, albeit the older file which didn't have a part of the implementations. Nevertheless, the code was rewritten and it was much better than starting all over again. Yet, this could be prevented if the source code was stored and updated on a version control system such as GitLab or GitHub.

Next, since the project would be ideally done on the local computer on campus, a new device had to be arranged to be able to work. Thankfully, a device was borrowed soon on which all the remaining work of the project was performed on. However, since it was a new device, the software environment was required to be installed to work. What made the matters worse was the slow Internet connection speed in the home city (100-200 kbps). With such speed, the software had to be downloaded and when the jobs for training the models had to be submitted to HPC of the university (which would take 20-30 hours), remote control's connection had a big delay and would get interrupted every 10 minutes. The remote control stability got better after finding a workaround and was crucial for the project since training the models on personal devices was impractical.

Finally, a challenge that was somewhat of personal discovery was the difficulty of working at home. The campus provides a working environment that generally doesn't require from the student other non-learning responsibilities. On the other hand, when studying away from campus, there are responsibilities that need to be taken care of. As a result, often the allocated time for the project was shorter than it was preferred. Later in the development, more time could be used to work due to the placed quarantine in mid-April.

7. Summary

The project's objective was to explore how emotion recognition can be accomplished specifically with statistical method using deep learning technique. Due to the research early on in the project it was found out that LSTMs are effective in such tasks due to the ability to remember information. Since Twitter posts have a lot of meaningless information, various preprocessing techniques were discovered and used to clean the texts. Three classifiers with same architecture were created which can classify texts into multiple classes as discussed. Although, the main objective of the project was achieved, the performance of these classifiers do not improve the state-of-the-art results of previous studies done on the topic. The reasons for this were described earlier which included unreliability of Twitter posts and model's weakness. Compared with the state-of-the-art accuracy, the project's model accuracy is worse by 7-9% in all type of classifications [3].

7.1 Reflection

After an year of researching and implementing the project, an important experience was gained. Although the amount of research and implementation done should be considered as sufficient, more progress could be done to improve the performance of the model and the application. The insufficient implementation was mostly due to the challenges from the other three modules in the first semester and the sudden pandemic. If the project was to be done again from the beginning, an attempt would be made to start all courseworks and project's tasks in the first semester much earlier. Additionally, more focus would be given to model structure and identification and prevention of overfitting. Finally, the combination of text preprocessing techniques used in the project did not produce perfectly clean samples, therefore more techniques would be analyzed and added to the combination.

7.2 Future work

Since the classifier didn't achieve a performance that was hoped for, there are few ways that could help improve it. First, the words in the corpus could be corrected using dictionaries to overcome the problem of duplicated words. As discussed earlier, the generated dictionary of the corpus contained a lot of semantically same words and it would be worthwhile to see how it would improve the performance. Next, during development, project supervisor advised to try "Ensemble learning" to improve the performance. Ensemble learning models is a combination of multiple models that improve the performance and prevent using a bad model. Additionally, ensemble models help in preventing overfitting and it would be useful too see how it would have improved the performance of the LSTM model. Finally, more advanced application with the integrated LSTM model can be implemented to utilize the model in meaningful way. For instance, a healthcare chat bot which could recognize emotions can be helpful by identifying and helping people with mental health problems. These possible solutions and improvements are going to be applied in future since improved project can be helpful in future career.

Bibliography

[1] Aaron Smith. U.s. smartphone use in 2015. Technical report, Pew Research Center, April 2015.

[2] Gabriel Stricker. The 2014 #yearontwitter, December 2014.

[3] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling, 2016.

[4] State of online reviews. Technical report, Podium, January 2017.

[5] DK Kirange and RR Deshmukh. Emotion classification of news headlines using svm. *Asian Journal of Computer Science and Information Technology*, pages 104–106, 2012.

[6] Shiliang Sun, Chen Luo, and Junyu Chen. A review of natural language processing techniques for opinion mining systems. *Information Fusion*, 36, 11 2016.

[7] Vimala Balakrishnan and Wandeep Kaur. String-based multinomial naïve bayes for emotion detection among facebook diabetes community. *Procedia Computer Science*, 159:30 – 37, 2019. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[8] Carlos Busso, Zhigang Deng, Serdar Yildirim, Murtaza Bulut, Chul Min Lee, Abe Kazemzadeh, Sungbok Lee, Ulrich Neumann, and Shrikanth Narayanan. Analysis of emotion recognition using facial expressions, speech and multimodal information. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 205–211. ACM, 2004.

[9] Kevin B Meehan, Chiara De Panfilis, Nicole M Cain, Camilla Antonucci, Antonio Soliani, John F Clarkin, and Fabio Sambataro. Facial emotion recognition and borderline personality pathology. *Psychiatry research*, 255:347–354, 2017.

[10] Stuart K Card. *The psychology of human-computer interaction.* Crc Press, 2018.

[11] Mehmet Berkehan Akçay and Kaya Oğuz. Speech emotion recognition: Emotional models, databases, features, preprocessing methods, supporting modalities, and classifiers. *Speech Communication*, page 2682, 2019.

[12] Sandra Passardi, Peter Peyk, Michael Rufer, Tanja S.H. Wingenbach, and Monique C. Pfaltz. Facial mimicry, facial emotion recognition and alexithymia in post-traumatic stress disorder. *Behaviour Research and Therapy*, 122:103436, 2019.

[13] R. Santhoshkumar and M. Kalaiselvi Geetha. Deep learning approach for emotion recognition from human body movements with feedforward deep convolution neural networks. *Procedia Computer Science*, 152:158 – 165, 2019. International Conference on Pervasive Computing Advances and Applications- PerCAA 2019.

[14] D. Al-Hajjar and A. Z. Syed. Applying sentiment and emotion analysis on brand tweets for digital marketing. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–6, Nov 2015.

[15] Swee Hoon Ang and Sharon YM Low. Exploring the dimensions of ad creativity. *Psychology & Marketing*, 17(10):835–854, 2000.

[16] Erik Cambria. Affective computing and sentiment analysis. *IEEE Intelligent Systems*, 31(2):102–107, 2016.

[17] Carlo Strapparava and Alessandro Valitutti. WordNet affect: an affective extension of WordNet. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA).

[18] Erik Cambria, Daniel Olsher, and Dheeraj Rajagopal. Senticnet 3: a common and common-sense knowledge base for cognition-driven sentiment analysis. In *Twenty-eighth AAAI conference on artificial intelligence*, 2014.

[19] Anaıs Collomb, Crina Costea, Damien Joyeux, Omar Hasan, and Lionel Brunie. A study and comparison of sentiment analysis methods for reputation evaluation. *Rapport de recherche RR-LIRIS-2014-002*, 2014.

[20] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[22] Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

[23] Robert Gove and Jorge Faytong. Machine learning and event-based software testing: Classifiers for identifying infeasible gui event sequences. In *Advances in Computers*, volume 86, pages 109–135. Elsevier, 2012.

[24] Ze-Jing Chuang and Chung-Hsien Wu. Multi-modal emotion recognition from speech and text. In *International Journal of Computational Linguistics & Chinese Language Processing, Volume 9, Number 2, August 2004: Special Issue on New Trends of Speech and Language Processing*, pages 45–62, 2004.

[25] Bart Baesens, Tony Van Gestel, Stijn Viaene, Maria Stepanova, Johan Suykens, and Jan Vanthienen. Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the operational research society*, 54(6):627–635, 2003.

[26] Kashfia Sailunaz and Reda Alhajj. Emotion and sentiment analysis from twitter text. *Journal of Computational Science*, 36:101003, 2019.

[27] Han Luo, Mingzhu Wang, Peter Kok-Yiu Wong, and Jack C.P. Cheng. Full body pose estimation of construction equipment using computer vision and deep learning techniques. *Automation in Construction*, 110:103016, 2020.

[28] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran. Deep convolutional neural networks for lvcsr. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618, May 2013.

[29] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS Research*, 2019.

[30] Cícero dos Santos and Maíra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[31] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[32] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339 – 356, 1988.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[34] Haşim Sak, Andrew Senior, Kanishka Rao, Françoise Beaufays, and Johan Schalkwyk. Google voice search: faster and more accurate. *Google AI Blog*, Sep 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. *arXiv e-prints*, page arXiv:1409.3215, Sep 2014.

[36] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct 2017.

[37] Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[38] Venelin Valkov. Hacker's guide to fixing underfitting and overfitting models. *https://www.curiousily.com/posts/hackers-guide-to-fixing-underfitting-and-overfitting-models [Accessed 9 May 2020]*, October 2019.

[39] Eric Cai. Machine learning lesson of the day-overfitting and underfitting, 2014.

[40] Douglas Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44:1–12, 05 2004.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[42] H. Binali, C. Wu, and V. Potdar. Computational approaches for emotion detection in text. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 172–177, April 2010.

[43] Sudhanshu Tiwari, M. Raju, Gurbakash Phonsa, and Deepak Deepu. A novel approach for detecting emotion in text. *Indian Journal of Science and Technology*, 9, 08 2016.

[44] Michal Toman, Roman Tesar, and Karel Jezek. Influence of word normalization on text classification. *Proceedings of InSciT*, 4:354–358, 2006.

[45] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. *Information Processing and Management*, 50(1):104 – 112, 2014.

[46] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems : a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

[47] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[48] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[49] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[50] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[51] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[52] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. Efficient non-parametric estimation of multiple embeddings per word in vector space, 2015.

[53] Terry Ruas, William Grosky, and Akiko Aizawa. Multi-sense embeddings through a word sense disambiguation process. *Expert Systems with Applications*, 136:288 – 303, 2019.

[54] Richa Bhayani Alec Go and Lei Huang. Sentiment140-twitter sentiment analysis tool. *Sentiment140,[Online]. Available: http://help. sentiment140. com/home.[Accessed 8 May 2020]*, 2016.

[55] Fredrik Lundh. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[56] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[57] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, 2011.

[58] Mahidhar Dwarampudi and N V Subba Reddy. Effects of padding on lstms and cnns, 2019.

[59] François Chollet et al. Keras. https://keras.io, 2015.