University of Sheffield

Contextual Emotion Detection in Text
Final Dissertation Report

Andrianos Michail

Supervisor: Dr. Nikolaos Aletras

A report submitted in partial fulfilment of the requirements
for the degree of Artificial Intelligence and Computer Science by Andrianos Michail

in the

Department of Computer Science

January 1, 2021

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.
Name: Andrianos Michail

Signature:

Date: 13/05/2020

Abstract

Detecting emotion in textual conversations is a Natural Language Processing task, which involves creating a system that breaks down a text into its core elements to identify what emotion it conveys. This is done by utilising inference based on labelled samples or by following preset rules. In this dissertation, we will be taking a look at the 3 emotional classes Happy, Angry and Sad as well as one conveying the rest named as Others. The most effective techniques being utilised in NLP so far have been rule-based systems (lexi-con approaches) and most successfully Machine Learning. The task of emotion detection through textual conversations is a complex task because human communication lacks structure, is subject to many nuances and varies from person to person. In this dissertation, we will initially introduce and analyse various pre-processing techniques as well as simple classification models implemented in Python. We will then demonstrate our results in reference to their effectiveness in modelling the task. The information conveyed in this dissertation can be of great use to future researchers in the creation of new robust systems for the detection of emotion in text. Further advances in the field of NLP and emotion detection could contribute in establishing new ways of understanding and studying human emotions.

COVID-19 Impact Statement

The lockdown imposed because of COVID-19 caused additional challenges for the completion of this project. In the second semester of the project, the university switched to online delivery of all teaching, and university buildings were closed. All project meetings were shifted to email correspondence and video meetings.

Acknowledgements

This Dissertation project was created as a part of my degree entitled ”Artificial Intelligence and Computer Science” at the Department of Computer Science, University of Sheffield.

Throughout this degree, I had the chance to obtain an excellent grasp on algorithms and their underlying mathematical derivations, design robust and high quality software and most importantly, the ability to properly plan and conduct experiments in a meaningful way that yield meaningful data. This project puts the aforementioned skills to use to produce a high quality report which contributes to the ability of future researchers to pave new ground in the field of NLP.

Quite importantly, I would like to thank my friends and family for the support and encouragement through these tough times.On a last note, I would like to express my gratitude to my supervisor Dr. Nikolaos Aletras for the opportunity to take part in such an interesting and innovate project and their support throughout this year.

Contents

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Chapters Overview 2
  - 1.2.1 Background 2
  - 1.2.2 Data and feature extraction 2
  - 1.2.3 Models 2
  - 1.2.4 Experiments 2
  - 1.2.5 Results and Discussion 2
  - 1.2.6 Conclusions 2

2 Background **3**
- 2.1 Emotions 3
- 2.2 Modelling Emotions 3
- 2.3 Emotion Detection in Textual Format 4
- 2.4 Previous work on emotion detection 5
- 2.5 Data in the Shared Task 5
- 2.6 Summary 6

3 Data and feature extraction **7**
- 3.1 Task 7
- 3.2 Data shaping for different architectures 7
- 3.3 Preprocessing 8
  - 3.3.1 Separating Emojis and Punctuation 8
  - 3.3.2 Stop-word removal 9
  - 3.3.3 Lowercasing 9
  - 3.3.4 Tokenisation 9
  - 3.3.5 Limiting data samples length 10
  - 3.3.6 Preprocessing Procedure 10
- 3.4 Model Features 11
  - 3.4.1 Bag of Words 11
  - 3.4.2 TF-IDF 12
  - 3.4.3 word2vec 13
- 3.5 Data exploration 13
  - 3.5.1 Data samples and tokens 13
  - 3.5.2 Dictionary analysis 13

3.5.3 Features analysis 15

4 Models **16**

4.1 Model Architectures 16

4.1.1 Flat Classification Architecture 16

4.1.2 Hierarchical Classification Architecture 16

4.2 Classification Models 17

4.2.1 Logistic Regression 17

4.2.2 FeedForward Neural Network 17

4.2.3 Activation Functions 18

4.2.4 Neural Network Optimisation 20

4.2.5 Weights Initialisation 20

4.2.6 Bias Initialisation 21

4.2.7 Underfitting 21

4.2.8 Overfitting 21

4.2.9 Validation 21

4.3 Implementation 22

5 Experiments **23**

5.1 Hyperparameter search space 23

5.2 Evaluation Metrics 24

5.2.1 Shared Task's Official evaluation 24

5.2.2 Evaluation Metrics in this Project 24

6 Results and Discussion **26**

6.1 Baseline Results 26

6.1.1 Baseline Results on the shared task 27

6.2 Best Results 27

6.3 Hierarchical Classification & Flat Classification 28

6.4 Relu & Sigmoid FFNN 28

6.5 Bag of Words & TF IDF 28

6.5.1 The BoW IDF 29

6.6 word2vec 29

6.7 Developers set results 29

6.8 Individual Class analysis 30

7 Conclusions **31**

8 Appendix A - Train Classification Models Code Snippet **35**

9 Appendix B - Hierarchical Architecture Classification Code Snippet **39**

10 Appendix C - Experiments and Results Hyperparameters **41**

10.1 Best Results 41

10.2 Hierarchical Classification & Flat Classification 42

10.2.1 Computationally Light 42

10.2.2 Computationally Heavy 42

10.3 Relu & Sigmoid FFNN 42
10.3.1 Computationally Light Experiments 42
10.3.2 Computationally Heavy Experiments 42
10.4 Bag of Words & TF IDF 43
10.4.1 Computationally Light 43
10.4.2 Computationally Heavy 43

List of Figures

3.1 An example conversation from the data set 7
3.2 A problematic data point 8
3.3 Problematic data point after restructure 9
3.4 Data points unaffected at cutoff 10
3.5 Pre-processing procedure diagram 11
3.6 A data point with a BoW value 22 11
3.7 Observing the Bag of Words value 12
3.8 Amount of infrequent words 14

4.1 Flat classification diagram 16
4.2 Hierarchical classification diagram 16
4.3 FeedForward Neural Network visualised example 18
4.4 Visualisation of Relu 19
4.5 Visualisation of Sigmoid 19

List of Tables

2.1 Official's task data set 5

3.1 The class cardinality of the test data set 7
3.2 Flat classification system data 8
3.3 Data of first layer Hierarchical classification 8
3.4 Data of second layer Hierarchical classification 8
3.5 The four most frequent words in the data set 14
3.6 The data set samples 14

5.1 Hyperparameter search space 23

6.1 Baseline results from Logistic Regression 26
6.2 Baseline results from shared task 27
6.3 Best results across experiments 27
6.4 Class distribution of models on dev set 29
6.5 Best results per class 30

Chapter 1

Introduction

1.1 Motivation

When we read a text message saying “Why should I care” or “Why are you not here yet!?”, we can observe that the sender conveys emotions through those messages. These emotions can be perceived differently from person to person but due to the complexity of conceptualising emotions through text, more often than not, people will perceive this text as an expression of anger and yet again whilst others can perceive emotion in the same text as frustration, and disappointment.

A computer system that can classify someone’s emotion based on a text they wrote, can be of great value when utilised properly. Picture this, imagine a world where your phone can detect if you are upset and responds by various such as of which could be recommending relaxing music. This is only a fragment of what a consistent automatic emotion detection system can bring to this world as the possibilities are in fact, endless.

However, there are challenges in the creation of a system that understands human emotion. Human emotions are strongly context-dependent, ambiguous and somewhat arbitrary.

Algorithms that detect emotion, are grouped into one of the three sub-categories: Rule-Based [6], simple Machine Learning [24] and Deep Learning approaches [31]. A comprehensive review of the approaches is available at [28].

Textual conversation whether it is direct messaging on social media, e-mail or even Short Message Service(SMS), is the most common form of human communication, as seen in a report written by [23]. Not only, there is a large volume of textual conversation available for training purposes but they are also reliable, easy to access and easy to process when compared to spoken communication. This makes textual conversations an excellent way to study human emotions and their links to communication.

Within this dissertation, research is conducted to a specific type of textual conversations, social media messaging. Inspired by SemEval2019 Task 3: “EmoContext: Contextual Emotion Detection in text”, through this emotion classification in text is explored. In this task, participants are expected to attempt classification of textual dialogues i.e. a user utterance along with two turns of context into the 4 following emotional categories; Happy, Sad, Angry or Others [3].

In order to achieve this, numerous experiments are conducted with multiple types of supervised classification systems, using Machine Learning and Deep Learning approaches.

As a training source for the classifier the labelled training data set of SemEval2019 Task 3: "EmoContext: Contextual Emotion Detection in text" which contains 30 158 textual dialogues taken from social media is utilised. For the evaluation of the system both micro and macro-f1 scores are considered.

1.2 Chapters Overview

In this section, a brief overview of the chapters will be provided.

1.2.1 Background

The Background chapter of this report is split into multiple sections, covering emotions and how to model them, as well as how an emotion detection system works and what previous researchers have done to face the multiple difficulties of the task.

1.2.2 Data and feature extraction

The Data and Feature Extraction chapter initially demonstrates the data formed from split into fractions to assist in the correct assessment of the system. Furthermore, it introduces different techniques designed to shape our data into a form that can be transformed into neural network features. Following up, different types of features are introduced and explained how they were altered to better utilise the data set. Finally, Data exploration is presented for the processed data, dictionary and neural network features.

1.2.3 Models

Within this chapter, the different models and architectures will be presented and explained as well as the underlying mathematical equations.

1.2.4 Experiments

In the experiments chapter, the search space for the experiments is initially presented. Afterwards, the evaluation metrics that are considered in this report, are introduced and explained. This chapter assists in the understanding of the following chapter demonstrating the results.

1.2.5 Results and Discussion

As per the Final result chapter, the best results are presented. Additionally, some hypotheses will be introduced using a wider variety of experiments. Following, the idea of conducting experiments on the developers set is explored. Lastly, an attempt to further analyse the best results so that they present valuable information regarding each emotional class.

1.2.6 Conclusions

In this chapter, the the summary of the project is discussed and ideas for future work are presented.

Chapter 2

Background

This chapter presents background and previous work on emotion detection. Initially, a brief discussion emerges on human emotions and theories behind that. Afterwards, a brief overview of how emotion detection in textual format has been perfomed is enclosed. Lastly, success of previous researchers and their techniques used in mentioned.

2.1 Emotions

Humans express their emotions with all of their actions but certain characteristics of their actions convey the majority of information regarding their emotions. Tone of voice, body language, body posture, spoken or written language and most importantly facial expressions are the main cues to understand emotion as seen at [1].

A recent study has argued for the existence of 27 distinct categories of emotions bridged by continuous gradients, an idea that shows how diversely human emotions can be interpreted as. More information about this study available at [5].

2.2 Modelling Emotions

Paul Ekman and others have developed theories on how some emotions are considered more basic than others [8]. These emotions are said to have ties to universal facial expressions and physiological processes such as increased heart rate and perspiration. However, not everybody agrees on which set of emotions are the most basic, as most big psychologists proposed different sets of basic emotions [8] [25]. Even more controversially, the very theory of basic emotions has been challenged in recent works[5]. Nonetheless, much of the efforts in automatic detection of emotions in text has focused on the handful of proposed basic emotions. More detailed information on modelling emotions and their use into Natural Language Processing available at [18].

Since most of the previous researchers on similar tasks used the Ekman's Six basic emotions [8], and the shared task allows classification into 3 emotions and one emotion category entitling Others, it would make most sense to follow this idea in this research.

2.3 Emotion Detection in Textual Format

Before we dive into emotion detection in text, a brief overview of the necessary knowledge is presented.

Lexicon-based Sentiment Analysis: The Lexicon-based Sentiment Analysis approach involves calculating intention/orientation for a text from the semantic orientation of words or phrases in the text based on the semantic orientation of word or phrases in the labelled training data set provided.

Machine Learning (ML): The study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns emerging from data and inference instead.

Supervised Learning: The Machine Learning task of learning a function that maps an input to an output based on the example input-output pairs provided beforehand. This input-output pairs is commonly refereed to as training data. It infers a function from labelled training data mapping inputs to outputs.

Deep Learning (DL): Deep Learning is a class of Machine Learning that utilises single or multiple layers of artificial neural networks to progressively increase performance, also commonly referred to as Deep neural networks.

Statistical Natural Language Processing (NLP): The Natural Language Processing research that has relied heavily on Machine Learning and Deep Learning techniques. The Machine Learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large labelled data sets with either human or computer annotations of typical real-world examples.

Reccurent Neural Networks(RNN): Recurrent Neural Networks, a subset of Deep Learning, are dynamical systems with temporal state representations. They are computationally powerful, and can be used in many temporal processing models and applications. Recurrent Neural Networks are modeled by systems of ordinary differential equations, they are also suitable for digital implementation using standard software for integration of ordinary differential equations. More information can be found at [7].

Nowadays, the majority of human communication is via text, therefore detecting emotion in textual format is the first and most important step in automatic detection of human emotions.

The majority of social media messages are not professionally typed or structured sentences. Due to the nature of human communication, textual conversations contain sarcasm, ambiguity, slang language and quite importantly, grammar and spelling mistakes.

These characteristics make any form of rule based emotion classifier perform poorly, as one of the biggest and most important requirements of their good performance is the consistent spelling of the same words, which is definitely not the case in this data set. Therefore, the examination of any rule based emotion classifier was avoided, no matter how appealing and interesting the idea may seem.

This however does not imply that any Machine or Deep Learning approach is going to perform well. It is expected, that with the correct preprocessing and training, these statistical NLP methods are likely to be able to recognise a majority of the textual conversation's nuances and recognise the patterns that emerge.

2.4 Previous work on emotion detection

Many researchers have previously worked on this specific task, along with many more working on similar tasks. To be more precise, observing the paper published on the specific task by Microsoft India at [3]. This paper initially introduces the task and then informs on what the data set provided is. Additionally, it contains a performance comparison table of the top 15 teams (according to F1-Score) along with a quick description of what model each team used. From the system description papers of the top 15 teams, it was observed that BiLSTMs/LSTMs were the most frequently used neural models [3]. Since this paper summarised the results and the techniques of the top performers, it is only logical to keep this information in mind when conducting investigation in this data set.

Additionally, SemEval2018's Task 1 : Predicting Affective Content in Tweets is another recent task, which contained a subtask with somewhat similar data and objective. Focus will be given only into the content in English from this task [19]. There here is an official paper that introduces the task and analyses the data set provided and contains multiple tables demonstrating the results of the top performing teams, and summarises the models used [19].

In the similar subtask of this task, there were a wide range of successful techniques. According to [19], among the successful ones were SVM/SVR, Logistic Regression and Recurrent Neural Networks.

Visible at [19], in this task there was a wider variety of successful techniques used for the detection of emotion in text, including various simple Machine Learning techniques. Taking this as proof that conventional Machine Learning techniques can perform well on emotion detection, it motivates the idea to start the investigation from simple Machine Learning techniques, rather than dive straight into Deep Learning models.

Outside of SemEval tasks, there have been great advancements in the field of NLP recently. A large amount of effort in the recent years is spent on experimenting with Deep Learning techniques such as Long Short-Term Memory, Gated Recurrent Units and Convolutions Neural Networks. One particularly important reason for the success of these experiments is the increasing usage and utilisation of robust word embedding techniques which have proved to be more accurate representation of data compared to the more traditional techniques such as Bag of Words and TF IDF. More information regarding the recent advancements and techniques used in NLP are available at [31]

2.5 Data in the Shared Task

The data set consists of real English textual dialogues i.e. a user utterance along with two turns of context which was filtered for offensive language, personal identity information. This results in a real, useful training data set, but as mentioned before, since it consists of human texts, unstructured and very likely to contain sarcasm, ambiguity slang and typos.

	Happy	**Angry**	**Sad**	**Others**
Training set	4243 - 14%	5463 - 18%	5506 - 18%	14948 - 50%
Dev and Eval set	4% Total			96%

Table 2.1: *Official's task data set*

As mentioned in the task and visible at table 2.1 there is a strong class imbalance, reasons behind being, that the developers and evaluation set follows a real-life distribution of data, while the training set is more tailored to allow us to train the classification system.

In the system, the data is shaped slightly differently and is discussed in the Data and Feature Extraction Chapter.

2.6 Summary

The research done on automatic emotion detection systems in the past few years has been colossal compared to the past. The need for a robust and consistent emotion detection system, has driven a huge attention into the field of Natural Language Processing (NLP). Even if the field has been around for over 60 years, it has seen a huge leap in performance and consistency with the recent re-introduction of Deep Learning in the research field.

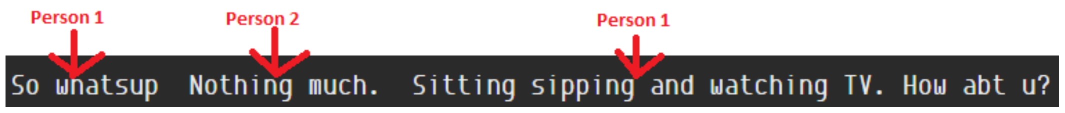
To conclude, I believe this field has a lot of room for exploration and solutions to be discovered. Previous work on the last 2 years SemEval's shared tasks with similar targets, strengthens this belief with the plethora of different successful techniques that performed excellently.

Chapter 3

Data and feature extraction

3.1 Task

The data set consists of real English textual dialogues i.e. a user utterance along with two turns of context, filtered for offensive language, personal identity information. To explain this visually, following is an example data point:



```
So whatsup  Nothing much.  Sitting sipping and watching TV. How abt u?
```

Figure 3.1: *An example conversation from the data set*

As observed in the figure above, there are 2 users in this conversation. The task is to classify the emotion of ”Person 1” and one approach is to only use their messages in the classification. During the experiments it was chosen to use the entire data sample as it was deemed that Person 2’s response is also important.

3.2 Data shaping for different architectures

Within this system 20% of the training set maintaining the original class balance is separated to be used as a testing set. The testing set is not utilised in training or validation of any model and is only considered for the evaluation of performance of models.

	Happy	**Angry**	**Sad**	**Others**
Testing Set - 20%	848 - 14%	1093 - 18%	1101 - 18%	2990 - 50%

Table 3.1: *The class cardinality of the test data set*

An attempt to solve the class imbalance in the provided data set as seen in table 2.1, was by the utilisation of class weights. However, this technique did not perform well so it was chosen to experiment with under-sampling techniques to solve the class balance problem.

After sampling the data to be balanced class cardinality wise to be used for the Flat classification system, the data was shaped to:

	Happy	**Angry**	**Sad**	**Others**
Training Set	2716 - 25%	2716 - 25%	2716 - 25%	2716 - 25%
Validation Set	679 - 25%	679 - 25%	679 - 25%	679 - 25%

Table 3.2: *Flat classification system data*

As seen from table 3.2, the sampling of the data caused the loss of 8000 samples from the Others class, data that would be helpful to utilise.

To avoid this data loss, an alternative hierarchical classification architecture was developed that consists of 2 classifiers. This classification system allows the utilisation of the data set more efficiently. Following is the split of the data for this architecture:

	Others	**{Happy, Sad, Angry}**
Training Set - 80%	9566 - 50%	9566 - 50%
Validation Set - 20%	2392 - 50%	2392 - 50%

Table 3.3: *Data of first layer Hierarchical classification*

	Happy	**Angry**	**Sad**
Training Set - 80%	2716 - 33.3%	2716 - 33.3%	2716 33.3%
Validation Set - 20%	679 - 33.3%	679 - 33.3%	679 33.3%

Table 3.4: *Data of second layer Hierarchical classification*

This model is analysed in more detail at Chapter 4. More information regarding hierarchical classification systems can be found at [29].

3.3 Preprocessing

Preprocessing or text sanitation is bringing the initial data set to a form that is normalized and structured [9]. There are various techniques available to preprocess data, each effective for different data sets and tasks. In this section, the preprocessing techniques used will be introduced.

3.3.1 Separating Emojis and Punctuation

Initially it is observed that emojis and punctuation are very frequent in our data set and have no specific format or pattern. To keep the consistency of the data set, it was decided to treat each emoji as a separate token and process them like any other word. Following is a problematic data point.

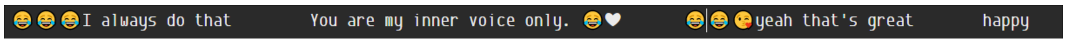
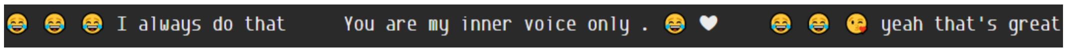

```
😂😂😂I always do that      You are my inner voice only. 😂🤍      😂|😂😘yeah that's great      happy
```

Figure 3.2: *A problematic data point*

There are issues in the data point presented. Different emojis are used one next to another, they are concatenated with other words and are repeated multiple times without any space. The way this problem is tackled is by the separation of all emojis using spaces.

It is equally important to notice that there is a full stop exactly after “only”. Similarly, the punctuation needs to be separated and this is achieved by adding a space before each punctuation point.

This would shape the data point to look like the following figure:



```
😂 😂 😂 I always do that    You are my inner voice only . 😂 🤍    😂 😂 😘 yeah that's great
```

Figure 3.3: *Problematic data point after restructure*

This procedure introduces a structure to the data set and allows features like Bag Of Words to be a representative feature of the data. This process is also applied to emoticons such as ;D, :) as they tend to have the same effects.

3.3.2 Stop-word removal

A stop-word list removal is removing the set of the most commonly used words as they do not provide any meaningful context, such as “the”, “and”, “of” and “a”. Removing words in the stop-list speeds up the classification process as the data set is significantly reduced and focuses on semantically important information instead of low information words. More information regarding stop-word removal is available at [9].

Sadly this can cause problems, for example removing words such as “the” and “of”, would make the phrase “Have the hang of something” into “Have hang something”, which would result in loss of information and sentiment.

Within the experiments conducted, the majority of the feature types utilised ignore the original order of words therefore the negatives of this effect are minimal.

In this dissertation, the minimalist stop-word list: {“a”, “the”, “is”} is used for removal in order to discard as little data as possible.

3.3.3 Lowercasing

Lowercasing is the replacement of all capital letters in the data with their lowercase equivalent. This minimises the effect of incorrect capitalisation and different capitalisation of the same words in our data.

One issue particularly common when using this technique in emotion detection is that capitalisation of words are often there to hint anger, sadness or excitement. Techniques have been developed to replace words written in all capitals with special symbols to minimise data loss [22].

After initial experimentation it was deemed most effective to apply lowercasing to the entire data set as it performed more consistently.

3.3.4 Tokenisation

Tokenisation is the method of reducing a given text into smaller pieces known as tokens. An example of tokenisation is breaking sentences into words, most commonly done by separation on spaces. More information regarding Tokenisation is available at [30].

However it is likely that two or more words together that contribute to a common meaning. If those words are tokenized, it results in a loss of information. For example, breaking the name of the band "Sex Pistols" into the corresponding tokens, lead to loss of information and the addition of incorrect data.

3.3.5 Limiting data samples length

A less common technique, that is especially useful when used with word embeddings, is the limitation of the data set to have a maximum word limit per sample. This, ensures a smooth data and minimises the effects of outliers that contain a large amount of words.

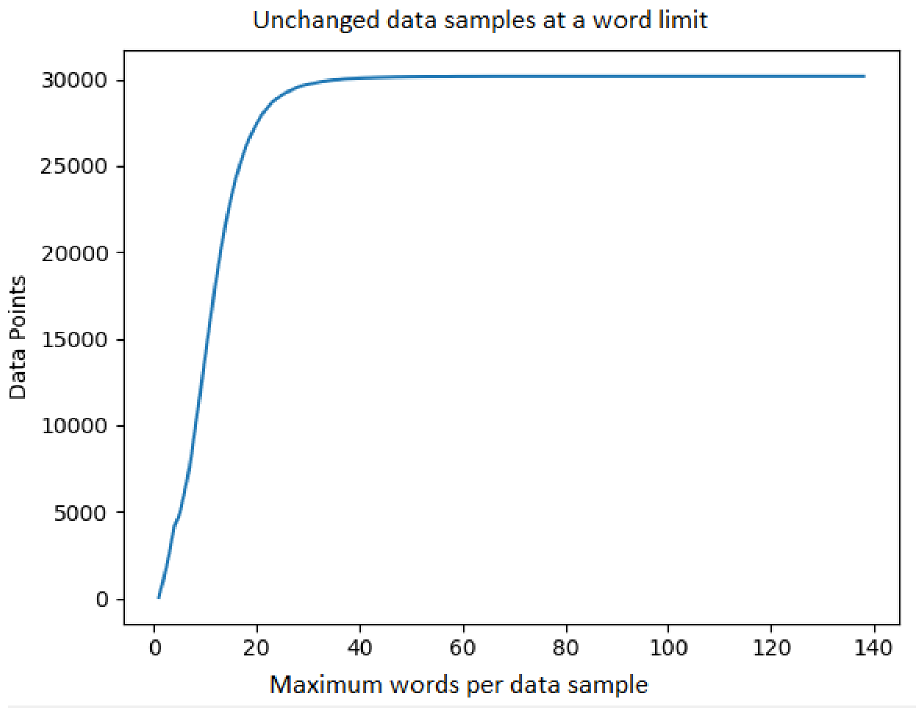


Figure 3.4: *Data points unaffected at cutoff*

As it can be observed from the previous graph, 40 words length is a near optimal data sample limit as it will result in smooth data while causing minimal data loss.

3.3.6 Preprocessing Procedure

Different subset of pre-processing techniques have different results on different features and different models. In order to receive the best results it would be required to conduct a lot of experiments with a variety of techniques. To restrict this set of experiments, it is assumed that the results received from the baseline system of Logistic Regression, are indicative of the results for all of the models. In this manner the above preprocessing techniques were chosen to achieve frugality in our work.

The following diagram demonstrates the pre-processing procedure that is applied to the data set:

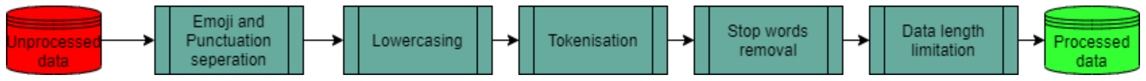


Figure 3.5: *Pre-processing procedure diagram*

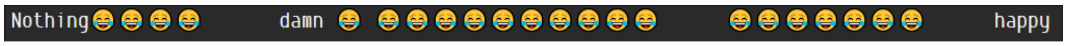
3.4 Model Features

The input features of a neural network are the set of attributes representing each data sample. These features are created through different processes and the models have been designed to accept different features. The accuracy of models directly depend on how well our features represent the data. In this section, an introduction and analysis of the feature types will be presented.

3.4.1 Bag of Words

In this feature model, a text, for example a sentence or a conversation, is represented as the bag (multi set) of its words, ignoring grammar or order while keeping multiplicity.

In the initial observations of the Bag of Words(BoW) feature set, it was noticed that the Bag of Words features contained a few data points with extremely high values. Following is a figure showing one of those data points:



Nothing😂😂😂😂 damn 😂 😂😂😂😂😂😂😂😂😂😂 😂😂😂😂😂😂😂 happy

Figure 3.6: *A data point with a BoW value 22*

As seen from figure 3.6, the data sample contains a lot of repetition and this does not necessarily helps the shape of features, as the classifier would be able to spot the correct emotion, regardless of having a maximum BoW limit or the original 22. Following is a graph that demonstrates how many of the BoW Values will remain unaffected.

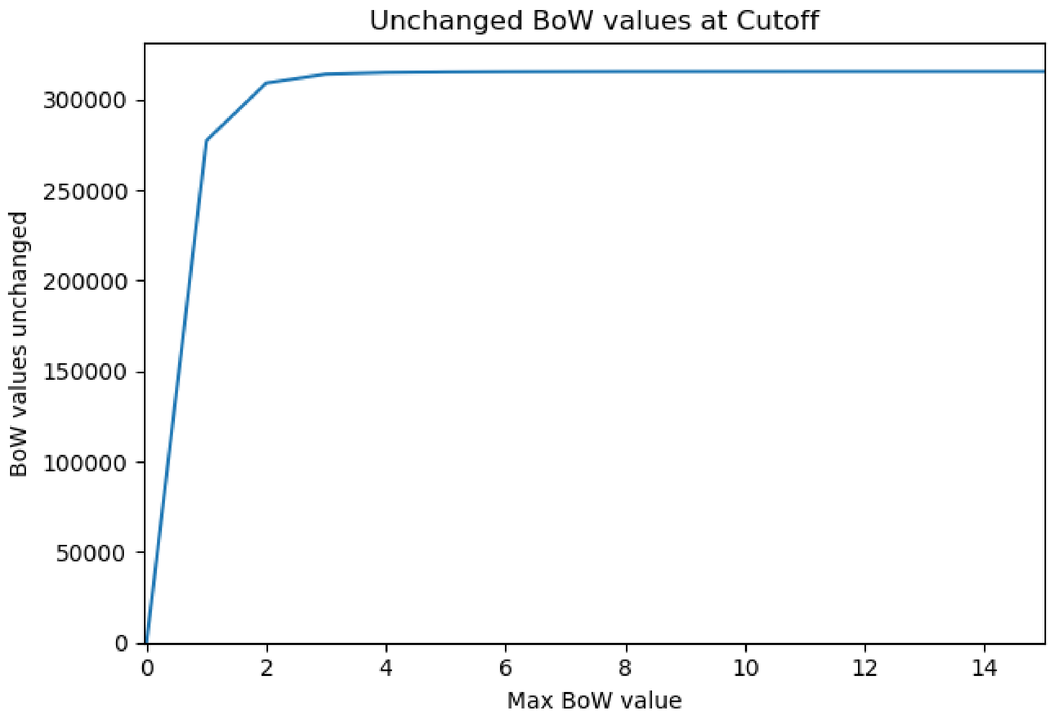


Figure 3.7: *Observing the Bag of Words value*

Visible from the previous figure, a value that will result in minimal data loss is 4. Vast majority of our data remains unaffected and we have minimised the effect of the outliers resulting in a cleaner data set.

3.4.2 TF-IDF

A consistent feature model examined is TermFrequency- InverseDocumentFrequency(TF-IDF).

Inverse document frequency is a measurement of how important a term is and it is calculated with the following formula:

$$InverseDocumentFrequency[\text{“token”}] = \log_{10} \frac{|\text{Training Samples}|}{|\text{Samples Containing “token”}|}$$

After calculating the inverse document frequency, the Term Frequency is calculated for each value using the following formula:

$$TermFrequency = \frac{\text{Number Of Occurences In Sample}}{\text{Total Tokens In Sample}}$$

$$TF - IDF = TermFrequency * InverseDocumentFrequency \text{ for each token}$$

The purpose of this formula is to give larger values to tokens that are infrequent in the data set and frequent in the data sample. For example, if the word “he” is for example present in 40% of our data set, this means that it is not necessarily indicative of the underlying emotion. In contract, observing the token “furious” we notice that it is present at only a few data samples, which is likely that the token is more indicative and representative of the emotion in the text.

More detailed information regarding the mathematical proof of TF-IDF is available at [12].

3.4.3 word2vec

The last feature model examined is word2vec word embeddings. Word embeddings are dense vector representations of words, designed to capture their semantic and syntactic information [4]. In simpler words, it is a way of representing words into an N dimensional space, so that it can be later used as features for a classification model. More information regarding word embeddings available at [4].

A word2vec model was trained utilising only the training data. Usually, words embeddings are successful when trained with large amounts of training data (most commonly used with pre-trained embeddings) but within the scope of this project it was attempted to achieve good performance without utilizing any source of pre-trained embeddings. According to [17], when training with only a small amount of data, the Skip-Gram training algorithm performs better and therefore conducted all of the experiments by training the word embedder with Skip-gram. More information about word2vec is available at [17].

3.5 Data exploration

In this section, characteristics of the processed data and model features are analysed and presented.

3.5.1 Data samples and tokens

In the preprocessed data set, there are a total of 24126 data samples containing 348911 tokens. These tokens can be words, punctuation symbol or emojis. This means that on average each data point contains 14.5 tokens. This demonstrates that, as expected from social media messaging, each user’s message is on average 5 tokens long.

3.5.2 Dictionary analysis

Looking at the preprocessed data, it is observed that the number of unique words created in the dictionary are 14217 in 24126 data samples. Important information is how many words are regularly used within our data set. The following graph demonstrates the amount of words in the dictionary that occur at most in 15 samples.

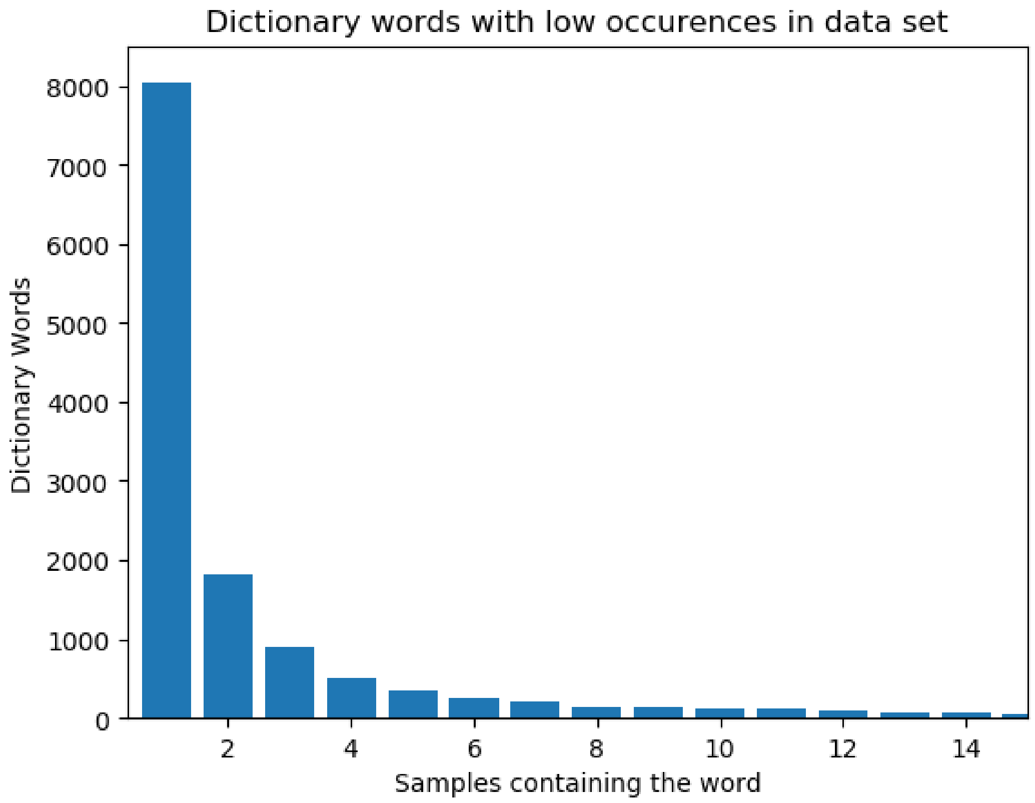


Figure 3.8: *Amount of infrequent words*

As seen from figure 3.8, over 50% of the words in the dictionary occur in exactly 1 data sample while 80% in up to 5 data samples. This demonstrates that the dictionary is unreliable as this is likely caused due to variations and typos of the same word. This is one of the problems that may arise when building a dictionary from real people's informal textual conversations. Following is a table of observations of the 4 most commonly used tokens and their occurrences in different classes:

Word	**Happy**	**Angry**	**Sad**	**Others**	**Total Samples**
"i"	1584 - 13%	2460 - 21%	2570 - 21.5%	5264 - 44.5%	11878
"you"	1454 - 13%	1703 - 15%	2706 - 23%	5600 - 49%	11465
"."	1401 - 15%	1662 - 17.5%	1813 - 19%	4549 - 48%	9425
"?"	782 - 9%	1541 - 19%	1407 - 17%	4494 - 55%	8224

Table 3.5: *The four most frequent words in the data set*

Happy	**Angry**	**Sad**	**Others**	**Total**
4240 - 14%	5465 - 18%	5505 - 18%	14950 - 50%	30158

Table 3.6: *The data set samples*

Looking at tables 3.5 and 3.6 it is observed that the “i” token is more commonly used amongst Angry and Sad data samples and that it is not as often included in the Others class. This implies that when encountering a token “i” in a sample it is more likely to convey a Sad or Angry emotion. As expected, the token “.” is neutral as it approximately follows the class distribution. What is interesting about the token “?”, is that it is rarely used in a Happy data sample and is more commonly used in the “Others” emotional classes.

3.5.3 Features analysis

Equally important is the observation of the features due to the fact that they directly effect the performance of the classification model.

Bag of Words

After conducting further analysis on the Bag of Words feature set it is observed that on average, each data sample contains 10 different tokens in the Bag of Words feature. This demonstrates that on average, 5 out of 15 tokens in each data sample is present more than once in the same data sample. This is positive for the data samples as it demonstrates that the different messages are correlated.

TF IDF

Within the 24000 training set, the maximum IDF value of a token is 4.38. Combining this and the fact that the maximum Bag of Words value allowed is 4, the high values of our feature are capped in low numbers. This implies that the feature set is well controlled and the effect of the outliers is minimised.

word2vec

As mentioned earlier in this section, each data sample has an average of 14.5 tokens. Since the maximum sentence value is set to be 40, we can it can be calculated that on average the last 63.75% of the features are empty. This characteristic makes word2vec features hard to work effectively. A solution that would potentially improve this is to set a lower maximum sentence limit and discard more data, but this solution did not improve performance and was not further considered.

Chapter 4

Models

4.1 Model Architectures

4.1.1 Flat Classification Architecture

The Flat classification architecture is the simple idea of training one classification model to declare between the 4 classes: Other, Happy, Angry, Sad. Following is a figure demonstrating the flat classification architecture:

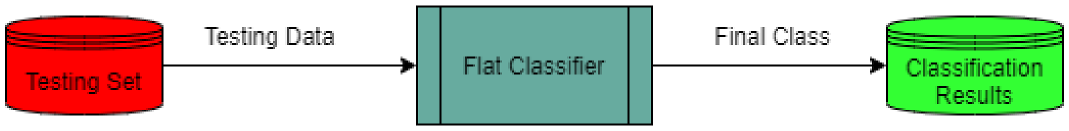


Figure 4.1: *Flat classification diagram*

The data for such a technique require to be adjusted to be equal. A common technique to adjust the data is by oversampling the less represented classes, by adding weights to each class. The technique that performed better within the experiments and is used throughout this dissertation is undersampling the more populated classes. This shaped the data set to be as in table 3.2. More information about learning from imbalanced data available at [11].

4.1.2 Hierarchical Classification Architecture

The Hierarchical classification structure allows for the recognition of more complicated patterns and typically exceeds the performance of flat classification by a small margin.

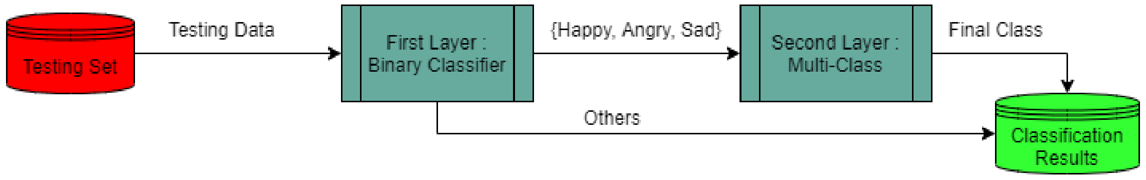


Figure 4.2: *Hierarchical classification diagram*

The first layer classifier checks whether the sample is part of the set S {Happy, Angry, Sad} or Others and if it deems that it belongs into the set S, a second different classification is conducted in the second layer to find its true class.

4.2 Classification Models

4.2.1 Logistic Regression

Logistic Regression(LR): In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail etc.

In the scope of the project a simple classification model is used to obtain baseline results. After some consideration and research, Logistic Regression is a commonly used, successful classification model suitable to provide baseline results for this dissertation.

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. It is a Statistical Learning technique categorized in Supervised Machine Learning methods dedicated to classification.

Logistic regression uses the Sigmoid function for binary classification. When generalised for multi class classification it instead utilising the soft max function to convert the input features into the output of a network as a probability measure.

4.2.2 FeedForward Neural Network

A FeedForward Neural Network(FFNN) also known as multilayer perceptrons(MLPs) is a deep learning model whos goal is to to approximate some function $f*$. In this dissertation, the FFNN trained attempts to correctly map the input (different feature types) into the correct emotion. Following is a figure demonstrating a Feedforward Neural Network with 2 hidden layers:

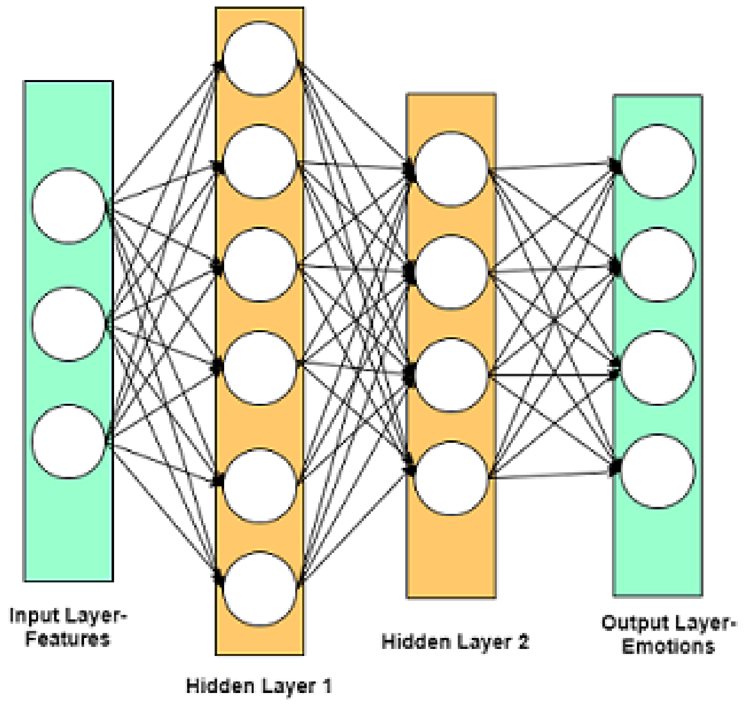


Figure 4.3: *FeedForward Neural Network visualised example*

A FeedForward Neural Network always consists of an input layer, an output layer and an integer arbitary number of “hidden” layers. As seen in figure 4.3, each node in a layer outputs to every node in the next layer.

4.2.3 Activation Functions

Using a nonlinear activation function in a FFNN allows the modelling of complex nonlinear functions [2]. If we were to instead use only linear activation function, the FFNN would end up being a linear function.

Relu

Relu is a non linear function that outputs the input directly if it is positive, or 0 otherwise [10]. This function is calculated with the following formula :

$$Relu(x) = max(0, x)$$

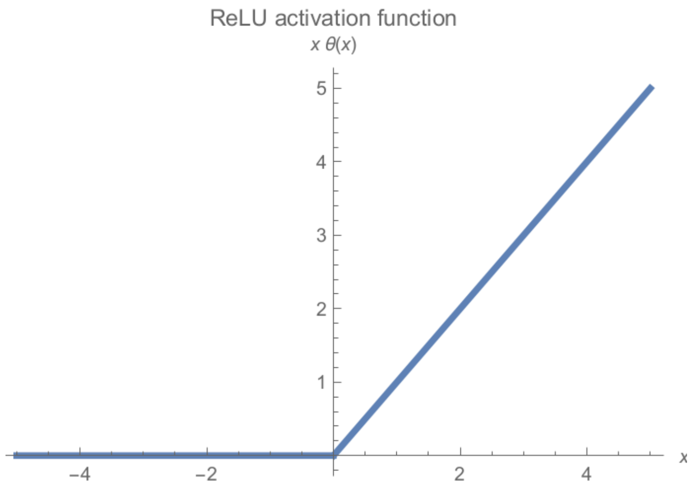


Figure 4.4: *Visualisation of Relu*

Source: https://miro.medium.com/max/970/1*Xu7B5y9gp0iL5ooBj7LtWw.png

One of the biggest advantages of Relu is how computationally light it is. Additionally, recent researchers have had great success using it due it the function activating only on positive numbers. This results in leading to different dead neurons which leads to variation in learning and through multiple runs, great classification systems.

Sigmoid

Logistic Sigmoid is a monotonic non-linear function. It is declared using the following formula:

$$Sigmoid(x) = \frac{1}{1 + e^{-}x}$$

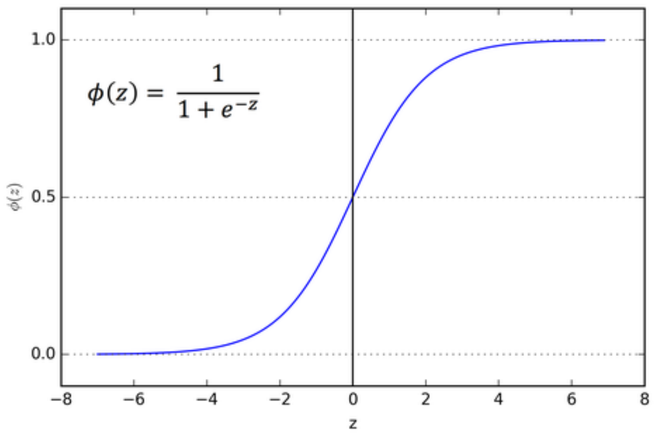


Figure 4.5: *Visualisation of Sigmoid*

Source: https://michielstraat.com/talk/mastertalk/featured.png

Sigmoid has been widely used in neural networks and has received some success, as most data sets tend to be normalised to $[0, 1]$ range where as seen from figure 4.5, this functions thrives.

Softmax

Softmax function is a function commonly used to transform the input from the final layer to the output layer into a probability distribution. This is defined with the following formula :

$$softmax(x) = \frac{exp(x_m^{out})}{\sum_\iota exp(x_m^{out}))}$$

where $x_{mk}^{out} = \sum j w_{kj}^{(n)} x_j^{(n+1)}$ are the values of the output layer of the network. Softmax gives the differentiable approximation of a non-differentiable function max.

4.2.4 Neural Network Optimisation

The weights of the neural network are optimised using the Backpropagation algorithm. The Backpropagation algorithm updates the weights by calculating the error using the correct labels[14]. This is done by utilising the derivatives of the activation function and propagation of the error back to the previous layers [14].

Cross Entropy Loss Function

Cross-entropy Loss is a measure of the difference between two probability distribution for a given variable [20]. KP Murphy [20] defines cross-entropy loss as “the average number of bits needed to encode data coming from a source with distribution p when we use the model q” . To define this well, it is defined through an example.

Imagine classifying some data using a FFNN, with N samples in the data set and N_o number of classes. The Cross-entropy Loss is defined as:

$$CrossEntropyLoss \equiv -\frac{1}{N} \sum_{m=1}^{N} \sum_{k=1}^{N_0} label_{mk} log P_{mk}$$

For classification problems, Cross-Entropy Loss is the preferred and most efficient loss function and throughout this dissertation it will be used as the error function [21].

4.2.5 Weights Initialisation

Standard Distribution: Standard distribution in statistics, is a continuous probability distribution which is asymptotic and symmetrically bell-shaped about the mean. The two parameters of the distribution are the mean and variance.

To achieve good performance, the weights of the neural network are initialised with:

$$\text{Xavier Initialisation} = \mathcal{N}(\mu = 0, \sigma = 1) \cdot \frac{1}{|\text{Neurons of Previous Layer}|}$$

$$\text{He Initialisation} = \mathcal{N}(\mu = 0, \sigma = 1) \cdot \frac{2}{|\text{Neurons of Previous Layer}|}$$

Within the experiments, Xavier Initialisation is performed to weights of layers that use Sigmoid or Softmax as an activation function. He Initialisation is performed to weights of layers that are activated using the Relu function.

More information about the benefits of using Xavier and He Initialisation and how they are derived available at [16].

4.2.6 Bias Initialisation

To maximise performance, we initialise the weights for our biases. It is common practice for the bias weights to be initialised with zeros. After some initial experiments, it was deemed that the following initialisation to work better for the Relu activated layers:

$$\mathcal{N}(\mu = 0.01, \sigma = 0.01)$$

This bias initialisation method achieves the generation of a majority 84.13% of small positive constant numbers, with the rest being small negative numbers. It is ensured that the majority of our neurons will be activated but also allows for variation of the trained weights from the initialisation.

4.2.7 Underfitting

Underfitting is when a model does not manage to capture patterns differentiating classes in data. This is usually caused by a model that is too simple or by data that is not representative enough. This is usually resolved by additional preprocessing, editing of the features or addition of more layers.

4.2.8 Overfitting

Overfitting is when a model learns the exact form of the training data and results in adequate performance on unseen data. In the next section, the techniques utilised to prevent overfitting will be presented and explained.

4.2.9 Validation

Overfitting is a common issue when training neural networks. This can usually be prevented by using Cross validation or by implementing techniques of early Stopping using the validation set. More information regarding validation techniques is available at [26].

Within this dissertation a simple yet effective early stopping technique is implemented. A check for overfitting is done every N (5 or 3) epochs. Every N epochs, classification of the validation set is performed with the current weights of the training. If it achieved a worse cross entropy loss than than the last run, it stops training and returns the weights of N amount of epochs ago. This validation technique, sometimes causes the model to underfit. This effect is minimised by conducting multiple runs of both $N = 5$ and $N = 3$. More information regarding early stopping techniques available at [26]

4.3 Implementation

All of the components of the system were built in Python, using the Numpy library. Development of entire preprocessing and classification models from scratch, as interesting as the process is, it sometimes results to inconsistency and unexpected bugs compared to implementations from kits. It was very important and interesting to create the models and the Hierarchical Architecture, which is typically not easy to use and tune through kits.

In order to minimise the factor of programmers error, multiple experiments were conducted on each technique and checked step by step. This ensured that the implementation is correct and the results presented in this dissertation can be approximately reproduced using kits.

Training the neural network and classification using the hierarchical architecture code snippets are available in the appendices.

Chapter 5

Experiments

5.1 Hyperparameter search space

To reach the optimal solutions for the models examined it is required to conduct a very high volume of experiments over various repetitions to ensure consistency throughout various random initialization.

If infinite resources were available, conducting all the experiments would be the optimal experimentation plan. Instead, it was chosen that various hyperparameters would be limited to ranges that deemed to have good potential results while respecting our restricted time frame. The following table demonstrates these ranges:

Hyperparameter	**Values**
Number of Batches	100, 200
Learning Rate	0.02-0.09 , 0.12 - 0.19
Hidden Layer Neurons	0, 50-120
Number of Hidden Layers	0-2
Number of Epochs	10, 20, 50
Early stopping N	3, 5

Table 5.1: *Hyperparameter search space*

5.2 Evaluation Metrics

The F measure (F1 score) is a measure of a test's accuracy and is defined as the weighted harmonic mean of the precision and recall of the test.

$$Precision \equiv \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)}$$

$$Recall \equiv \frac{TruePositives(TP)}{TruePositives(TP) + FalseNegatives(FN)}$$

5.2.1 Shared Task's Official evaluation

Evaluation in the task was carried out using the micro-averaged F1 score (F1) for the three emotion classes Happy, Sad and Angry on the submissions made with predicted class of each sample in the evaluation data set.

The official evaluation on the shared task was done using the following equation:

$$P_\mu \equiv \frac{\Sigma TP_i}{\Sigma(TP_i + FP_i)} \forall i \in \{Happy, Sad, Angry\}$$

$$R_\mu \equiv \frac{\Sigma TP_i}{\Sigma(TP_i + FN_i)} \forall i \in \{Happy, Sad, Angry\}$$

$$micro - F1(ExcOthers)_\mu \equiv 2 \cdot \frac{P_\mu \cdot R_\mu}{P_\mu + R_\mu}$$

Since the official evaluation of the task was carried out using F1 score, it would make sense that the experiments are evaluated in a simiar method.

5.2.2 Evaluation Metrics in this Project

For results evaluation in this dissertation one of the simplest evaluation metrics we mentioned is:

$$Accuracy : \frac{NumberOfCorrectPredictions}{TotalNumberOfsamples}$$

In multi class classification, accuracy is not a very suitable metric as it doesn't show how well the classification system performed across each class. Instead more attention will be given to the following two metrics, Micro-F1 and Macro-F1 score.

In binary classification, F1 score is calculated in the following way.

$$Precision = \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)}$$

$$Recall = \frac{TruePositives(TP)}{TruePositives(TP) + FalseNegatives(FN)}$$

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

To use F1 for multi class classification calculation of Micro-F1 is done with the following formula:

$$P_\mu = \frac{\Sigma TP_i}{\sum(TP_i + FP_i)} \forall i \in \{Others, Happy, Sad, Angry\}$$

$$R_\mu = \frac{\Sigma TP_i}{\sum(TP_i + FN_i)} \forall i \in \{Others, Happy, Sad, Angry\}$$

From the equations, it is noticeable that Precision and Recall are actually always equivalent, but in order to keep the clear form, the micro-f1 will be calculated as:

$$micro - F1_\mu = 2 \cdot \frac{P_\mu \cdot R_\mu}{P_\mu + R_\mu}$$

Macro-F1 is calculated by simply adding average precision and recall of all of the classes and then calculating the F1-score.

$$macro - prec_\mu = (\frac{1}{n} \sum \frac{TPi}{TPi + FPi}) \forall i \in \{Others, Happy, Sad, Angry\}$$

$$macro - rec_\mu = (\frac{1}{n} \sum \frac{TPi}{TPi + FNi}) \forall i \in \{Others, Happy, Sad, Angry\}$$

Where n is equal to the the number of classes.
Therefore, macro-f1 is calculated using the harmonic mean of micro-rec and micro-prec.

$$macro - F1_\mu \equiv 2 \cdot \frac{macro - prec_\mu \cdot macro - rec_\mu}{macro - prec_\mu + macro - rec_\mu}$$

Both Micro-F1 and Macro-F1 give slightly different results and it is important to observe both of them when evaluating the classifier's performance.

Due to the fact that all of the classes are considered in the evaluation metrics, it is deemed unimportant to present precision and recall in this chapter. In contrast, at the discussion of results in the next chapter, the performance of each class will be analysed by observing both precision and recall.

Chapter 6

Results and Discussion

Within this chapter, the initial baseline results are observed. Additionally, models and architectures are compared on both "light" and "expensive" computational runs. Last but not least, the best results for the models are presented and analysed.

6.1 Baseline Results

An important first step in the experiments, is to obtain baseline results for the different features. Baseline results are used as a comparison point for the other classification models. This will assist in the optimisation of the models and demonstrate how significant improvements were achieved compared to the baseline system. More information on choosing an appropriate baseline systems available at [15].

Logistic Regression with minimum pre-processing was used as a baseline system in this dissertation. It was deemed as optimal to apply 2 pre-processing techniques, stop word removal and separation of emojis and punctuation as this would be a realistic basic system. Following is a table showing best results over multiple runs with different hyperparemeters:

Baseline Model	Feature Type	Macro F1	Micro F1
Logistic Regression	word2vec	0.60	0.60
Logistic Regression	Bag of Words	0.78	0.79
Logistic Regression	TF-IDF	0.73	0.75

Table 6.1: *Baseline results from Logistic Regression*

First thing to note is that the word2vec word embeddings performance was mediocre, only managing to achieve a 0.65 Micro and Macro F1 score. This is not an unexpected result, as word embeddings typically require a larger training set to create accurate representations of words[17].

In contrast, it is observed that the system performs exceptionally well using Bag of Words achieving a Macro F1 score of 0.78 and Micro F1 score of 0.79. This is a very optimistic sign for our classification system and demonstrates that the preprocessing techniques chosen have shaped the data quite well. This baseline results are hard to beat with our more advanced models but the majority achieved slight improvements.

6.1.1 Baseline Results on the shared task

In the paper published regarding the shared task [3], there are results publicly available that can be used as a baseline system. However, these baseline results are tested on a different test set which has a different class distribution and evaluation model but are nevertheless an important quality measure. Here is a brief demonstration of the results:

Baseline Model	**Average Precision**	**Average Recall**	**Micro F1 (-Others)**
Competition Baseline	0.50	0.71	0.60
Top Performers	0.79	0.79	0.8

Table 6.2: *Baseline results from shared task*

These results demonstrate that the baseline results of table 6.1 are of high quality regardless of the difference in class distributions.

6.2 Best Results

Following is a table demonstrating the best results for all of the features for each architecture. Exact hyperparameters available in the appendix.

Feature: word2vec					
Arch.	**Producing Model**	**Macro Prec**	**Macro Rec**	**Macro F1**	**Micro F1**
Flat	FFNN 2HL Relu	0.664	0.626	0.644	0.659
Hier	FFNN 2HL Sigmoid	0.592	0.618	0.604	0.654
Feature: Bag of Words					
Arch.	**Producing Model**	**Macro Prec**	**Macro Rec**	**Micro F1**	**Macro F1**
Flat	FFNN 2HL Relu	0.823	0.787	0.804	0.8
Hier	FFNN 2HL Relu	0.787	0.801	0.794	0.81
Feature: TF_IDF					
Arch.	**Producing Model**	**Macro Prec**	**Macro Rec**	**Micro F1**	**Macro F1**
Flat	FFNN 2HL Relu	0.718	0.781	0.759	0.748
Flat	FFNN 2HL Relu	0.703	0.769	0.735	0.759

Table 6.3: *Best results across experiments*

To complement these results, some further observations will be presented comparing different characteristics and their performance. These hypothesis will be based on a number of experiments, however the data is no way sufficient to support them with certainty.

- Hierarchical Classification & Flat Classification
- Relu NN & Sigmoid NN
- Bag of Words & TF IDF
- word2vec Discussion

6.3 Hierarchical Classification & Flat Classification

Through the experiments (available in the appendix) of different models there was a clear pattern in the head to head performance of those 2 architectures with identical hyperparameters.

The Flat classification system, exceeded the performance of the Hierarchical classification performance with all of the features throughout most of the computationally light experiments had an increased 0.01-0.02 both on Micro and Macro F1 score.

In contrast, the Hierarchical system, exceeded the performance of the Flat classification in the more computationally demanding runs of 2 Hidden Layers by a similar margin of 0.01-0.02 both on Micro and Macro F1. The biggest improvement on features, was achieved on word2vec, which resulted in an average increase of 0.04-0.06 in both Micro and Macro F1.

It is expected that the better overall results of the hierarchical system are due to the utilisation of a larger data set and is a technique that is worth exploring in more complex classification models[13].

6.4 Relu & Sigmoid FFNN

It was demonstrated through the experiments(available in appendix) that FFNN using Relu has performed well and consistently for the majority of the experiments. The results support why it has been so popular in NLP and Deep Learning.

FFNN using Sigmoid in general hasn't provided consistent results. Different runs for the same hyperparameters vary by significant factors. Some of these runs end up being stuck in the Local Minima (all in one class) for Bag of Words and TF IDF runs. Even in the successful Sigmoid runs, the results are typically outclassed by many Relu runs.

The hypothesis that has been around the Deep Learning society that Relu is generally better than Sigmoid[10], is supported by the results of this research as Relu has outperformed Sigmoid across both computationally light and heavy experiments.

6.5 Bag of Words & TF IDF

Bag of Words has performed exceptionally well throughout the experiments. Results vary a lot depending on initialisation. Bag of Words failed to produce results with Sigmoid but worked exceptionally well with Relu.

TF IDF, performed worse than the baseline system. The best TF IDF models were significantly worse by a margin of 0.03 - 0.04 Micro and Macro F1.

This is unexpected because TF IDF works well when used with documents of large data samples [27], but it seems like this is not the case.

To conclude, Bag of Words on the testing set performed exceptionally well whilst the more completed TF IDF did not perform at the same level. However, at a later section, runs on the developers set are observed and the results seem to differ in the opposite direction.

6.5.1 The BoW IDF

During the comparison Bag of Words and TF IDF, there was an idea that came to mind. Would combining BoW with Inverse Document Frequency produce better results?

There are only a small amount of experiments conducted, but the results are indeed promising. This is calculated with the following formula:

$$InverseDocumentFrequency[\text{“token”}] = \log_{10} \frac{|\text{Training Samples}|}{|\text{Samples Containing “token”}|}$$

$$BoW - IDF = BoW * InverseDocumentFrequency \text{ for each token}$$

Bow IDF achieved an accuracy of 0.5-1 % higher accuracy than Bag of Words, but more importantly, performed consistently throughout runs with different models.

This only a hypothesis but this section is here as an invitation for further research on the idea.

6.6 word2vec

As suspected, word2vec without using pre-trained word embeddings performed poorly. The word2vec features performed on average 0.15 - 0.17 Micro F1 score, but also have a clear imbalance of Macro and Micro F1 score as seen in 6.3. This is likely due to the embedder not trained on enough data to produce representative features and thus resulting in mediocre results.

6.7 Developers set results

To further test the best models, run on the developers set was conducted. There are no labels available for the developers set, but it is known that the class distribution is 96% Others and 4% for the set {Happy, Angry, Sad}. Following is a table demonstrating the results of classification with the models from table 6.3.

Performing Model	Others	Happy	Angry	Sad
Dev. Distribution	96%	4% Total		
BoW FFNN Flat	64%	11%	15%	10%
BoW FFNN Hier	76%	10%	7%	7%
TF IDF FFNN Flat	75%	6%	4%	15%
TF IDF FFNN Hier	79%	9%	5%	7%

Table 6.4: *Class distribution of models on dev set*

All this section, due to the lack of labels on developers set, assumes the accuracy of the model as presented at table 6.3 to be somewhat indicative of the performance on the developers set.

As seen at table 6.3, on the test set, the flat and hierarchical architecture of BoW FFNN achieves similar results. However, from the table 6.4 it can be observed that the hierarchical version has a much more accurate distribution than the flat equivalent.

Looking at the TF IDF results of table 6.4, the models seem to perform exceptionally well compare to the results of the experiments conducted on the test set. The class distribution revealed is closer to the actual distribution than the BoW models. The hierarchical model seems to yield even better results, just like the BoW equivalent.

These results demonstrate that the extra effort and complexity of the hierarchical architecture possibly yields much better results in a set with class distribution closer to real data.

6.8 Individual Class analysis

Following is the run of the best performing BoW models from table 6.3 with results split for each class.

Best Result		**Others**		**Happy**		**Angry**		**Sad**	
Arch.	Feature	**Prec**	**Rec**	**Prec**	**Rec**	**Prec**	**Rec**	**Prec**	**Rec**
Flat	BoW	0.77	0.88	0.83	0.71	0.89	0.69	0.76	0.83
Hier	BoW	0.83	0.87	0.73	0.78	0.81	0.81	0.82	0.68

Table 6.5: *Best results per class*

Observing the table 6.5, it is noted that there is a significant difference between the flat and hierarchical runs in terms of consistency per class. The flat classifier at the Others, Happy and Angry Class has a significant difference between precision and recall. Instead, the hierarchical classifier, only has a significant different between precision and recall for the Sad class. This demonstrates that the Hierarchical architecture is more robust and as also seen from 6.4, it is likely to perform better in real data.

After further observation of each class it is noticed that the Happy class has the worst performance. This is because it is the least represented of the 4 classes in the training set and the classification model does not manage to infer many patterns. Additionally, the expression of happiness is strongly context dependent which is also hard to be understood by classification models.

Knowing that the data set was filtered for offensive language, it was expected that the Angry class which is very commonly related to offensive language, would perform poorly. Ignoring the well represented Others class, the Angry class performed best of the 3. This demonstrates that offensive language is not a necessary cue for recognition of emotions of anger. This hypothesis is further supported by results of top performers in the task as seen at [3].

Chapter 7

Conclusions

Through this dissertation, emotion detection using simple statistical NLP methods was presented. The difficulties emerging were demonstrated and solutions were proposed.

At the time of completing this report, I personally still believe there is room for exploration of the ideas mentioned. For example, hierarchical architecture of classification systems seems to perform exceptionally well in the experiments conducted. Therefore, it would be interesting to see further research and experiments on how well it would perform with more advanced Deep Learning techniques and different data sets [13].

Some other hypothesis in this report could be further examined. One that I find particularly interesting, is the idea of using Bag of Words with IDF. In the preliminary experiments, it seems to be performing well but the only way to draw conclusions is by exhausting experiments on different data sets and contexts.

A very promising idea that was not explored in this project is the idea of a collective per text emotion classification system. What this means is a classification system that classifies each utterance (3 per data sample) separately and the resulting emotion for the entire sample is the weighted combination of the 3. This idea is incredibly hard to implement and optimise but the results of such a sophisticated system could be incredible.

To conclude, emotion detection and Natural Language Processing in general has a lot of room for exploration, exploration that could lead to great effects for the society as a whole.

Bibliography

[1] Andra Adams, Marwa Mahmoud, Tadas Baltrusaitis, and Peter Robinson. Decoupling facial expressions and head motions in complex emotions. In *2015 International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 274–280. IEEE, 2015.

[2] Cenk Bircanoglu and Nafiz Arica. A comparison of activation functions in artificial neural networks. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2018.

[3] Ankush Chatterjee, Kedhar Nath Narahari, Meghana Joshi, and Puneet Agrawal. SemEval-2019 task 3: EmoContext contextual emotion detection in text. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 39–48, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics.

[4] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning, 2008.

[5] Alan S Cowen and Dacher Keltner. Self-report captures 27 distinct categories of emotion bridged by continuous gradients. *Proceedings of the National Academy of Sciences of the United States of America*, 114(38):E7900–E7909, 2017.

[6] Xiaowen Ding, Bing Liu, and Philip S. Yu. A holistic lexicon-based approach to opinion mining. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, page 231–240, New York, NY, USA, 2008. Association for Computing Machinery.

[7] Ke-Lin Du and M. N.S. Swamy. *Neural Networks and Statistical Learning*. Springer Publishing Company, Incorporated, 2013.

[8] Paul Ekman. Are there basic emotions? *Psychological Review*, 99(3):550–553, 1992.

[9] Kavita Ganesan. All you need to know about text preprocessing for machine learning & nlp, Nov 2019.

[10] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, pages 315–323, 2011.

[11] Haibo He and E.A Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[12] Djoerd Hiemstra. A probabilistic justification for using tf idf term weighting in information retrieval. *International Journal on Digital Libraries*, 3(2):131–139, 2000.

[13] Chenyang Huang, Amine Trabelsi, and Osmar R. Zaïane. Ana at semeval-2019 task 3: Contextual emotion detection in conversations through hierarchical lstms and bert. 2019.

[14] Makin J. Backpropagation. 2006.

[15] S Jimenez Vargas and A Gelbukh. Baselines for natural language processing tasks based on soft cardinality spectra. *International Journal of Applied and Computational Mathematics*, 11(2):180–199, 2012.

[16] Siddharth Krishna Kumar. On weight initialization in deep neural networks. 2017.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[18] Saif M. Mohammad. Sentiment analysis: Detecting valence, emotions, and other affectual states from text. In Herb Meiselman, editor, *Emotion Measurement*. Elsevier, 2016.

[19] Saif M. Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. Semeval-2018 Task 1: Affect in tweets. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2018)*, New Orleans, LA, USA, 2018.

[20] Kevin P Murphy and Kevin P. Murphy. *Machine learning [electronic resource] : a probabilistic perspective*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2012.

[21] G.E. Nasr, E. Badr, and C. Joun. Cross entropy error function in neural networks: Forecasting gasoline demand. pages 381–384, 01 2002.

[22] Renzo Navas. Sentiment analysis in microblogging: a practical implementation. 10 2011.

[23] Sherna Noah. Texting overtakes talking as most popular form of communication in uk. *The Independent*, Jul 2012.

[24] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, page 79–86, USA, 2002. Association for Computational Linguistics.

[25] Robert Plutchik. Emotions and sociopathy. *Behavioral and Brain Sciences*, 18(3):570–571, 1995.

[26] L. Prechelt. Early stopping - but when? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700:53–67, 2012.

[27] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[28] Kashfia Sailunaz, Manmeet Dhaliwal, Jon Rokne, and Reda Alhajj. Emotion detection from text and speech: a survey. *Social Network Analysis and Mining*, 8(1):1–26, 2018.

[29] Carlos Silla and Alex Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[30] Jonathan J. Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*, COLING '92, page 1106–1110, USA, 1992. Association for Computational Linguistics.

[31] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. 2017.

Chapter 8

Appendix A - Train Classification Models Code Snippet

```
def train_ff_backprop_nn_dynamic(eta, n_epoch, train_features, train_labels,
    val_features, val_labels, num_of_classes,
                                    activation="relu", neurons_list=[50, 50]):
    """ Train the neural network using the training and validation data (
    dynamic amount of hidden layers) 0 for Log_Reg.
            Input:
                eta: The learning rate of the neural network.
                n_epoch: The number of epochs (repetitions).
                train_features: the features of the training data.
                train_labels: the labels for the training data.
                val_features: the features of the validation data.
                val_labels: the labels of the validation data.
                num_of_classes: the number of classes in our set.
                activation: The activation function "sigmoid" or "relu".
                neurons_list: The list of neurons. [] for Log Reg.
            Output:
                The weights and bias weights lists(results of training).
    """

    # define the size of each of the layers in the network
    print(train_features.shape)
    print(train_labels.shape)
    val_errors = []
    input_size, n_samples = train_features.shape
    n_input_layer = input_size
    n_output_layer = num_of_classes

    batch_size = math.ceil(n_samples / NUM_BATCHES)

    weights = []
    bias_weights = []
    prev_layer = n_input_layer

    prev_weights = []
    prev_biases = []
    prev_error = 10000000

    if activation == "relu":  # He Init
        numerator = 2
    else:  # Xavier_Init for Sigmoid
        numerator = 1

    for neurons in neurons_list:  # Initialise weights and biases
        weights.append(np.random.randn(neurons, prev_layer) * np.sqrt((
    numerator / prev_layer)))
        if activation == "sigmoid":
            bias_weights.append(np.zeros((neurons,)))
        else:
            bias_weights.append(np.random.randn(neurons)/100+0.01)
        prev_layer = neurons

    # Initialise the last layer
    if len(neurons_list) == 0:  # Logistic Regression Case
        weights.append(np.random.randn(n_output_layer, n_input_layer
            * np.sqrt((1 / n_input_layer)))
    else:  # 1 or more Hidden Layers
        weights.append(
            np.random.randn(n_output_layer, neurons_list[-1])
                * np.sqrt((1 / neurons_list[-1])))  # Xavier even in relu.

    bias_weights.append(np.zeros(n_output_layer))

    # Train the network
    errors = np.zeros((n_epoch,))
    for i in range(0, n_epoch):
        # print(i)
        # We will shuffle the order of the samples each epoch
        shuffled_idxs = np.random.permutation(n_samples)
        confusion_matrix = np.zeros((num_of_classes, num_of_classes), int)
        for batch in range(0, NUM_BATCHES):
            # Initialise the gradients for each batch
            dif_weights = []
            dif_bias_weights = []
            for index in range(len(weights)):  # init each batch
                dif_weights.append(np.zeros(weights[index].shape))
                dif_bias_weights.append(np.zeros(bias_weights[index].shape))
            # Loop over all the samples in the batch
            for j in range(0, batch_size):
                # Input (random element from the dataset)
                if batch * batch_size + j >= len(shuffled_idxs) - 1:
                    break
                indx = shuffled_idxs[batch * batch_size + j]
                x = train_features[:, indx]
                acts = []
                outs = []
                out_delta = []
                desired_output = train_labels[indx]
                prev_out = x
                #  Feed forward
                for index in range(len(weights)):
                    # Neural activation
                    acts.append(np.dot(weights[index], prev_out)
                    + bias_weights[index])
                    if index == len(weights) - 1:  # Softmax for Last layer
                        exps = np.exp(acts[index]
                            - np.max(acts[index], axis=0, keepdims=True))
                        outs.append(exps / np.sum(exps, axis=0,
                            keepdims=True))
                    elif activation == "sigmoid":  # Activation with Sigmoid
                        outs.append(1 / (1 + np.exp(-acts[index])))
                    else:  # Activation with Relu
                        outs.append(acts[index] * (acts[index] > 0))
                    prev_out = outs[index]
                # Compute the error signal
                e_n = np.sum(desired_output * np.log(outs[-1]))

                # Backpropagation for Softmax layer
                out_delta.append(desired_output - outs[-1])
                if len(neurons_list) == 0:  # Logistic Regression Case
                    dif_weights[len(dif_weights) - 1] +=
                        np.outer(out_delta[0], x)
                    dif_bias_weights[len(dif_bias_weights) - 1] +=
                        out_delta[0]
                else:
                    dif_weights[len(dif_weights) - 1] +=
                        np.outer(out_delta[0], outs[-2])
                    dif_bias_weights[len(dif_bias_weights) - 1] +=
                        out_delta[0]

                # Backpropagation for the rest of the layers
                for index in range(len(outs) - 2, -1, -1):
                    #  Backpropagation: Current Layer to the previous layer
                    if activation == "sigmoid":
                        out_delta.append(outs[index] * (1 - outs[index])
                            * np.dot(weights[index + 1].T, out_delta[-1]))

                    if activation == "relu":
                        out_delta.append(np.heaviside(outs[index], 0)
                            * np.dot(weights[index + 1].T, out_delta[-1]))

                    if index == 0:  # Last Layer, input feat instead
                        dif_weights[index] += np.outer(out_delta[-1], x)
                    else:
                        dif_weights[index] +=
                            np.outer(out_delta[-1], outs[index - 1])
                    dif_bias_weights[index] += out_delta[-1]

                correct_value = np.argmax(desired_output)
                predicted_value = np.argmax(outs[-1])

                confusion_matrix[int(correct_value), int(predicted_value)] += 1
                # Store the error per epoch
                errors[i] = errors[i] + (-1 / n_samples) * e_n

            # After each batch update the weights using accumulated gradients
            for index in range(len(weights)):
                weights[index] += eta * dif_weights[index] / batch_size
                bias_weights[index] += eta * dif_bias_weights[index] /
```

```
   batch_size

        # print(confusion_matrix)
        print("Validation set Results :")
        conf, error = classify_with_ff_backprop_nn_dynamic(
            val_features, val_labels, weights, bias_weights,
                num_of_classes, activation)
        val_errors.append(error)
        if i % 5 == 0:
            if (error+MINIMUM_IMPROVEMENT) < prev_error:
                print("PREV ERROR IS: ")
                print(prev_error)
                prev_error = error
                prev_weights = weights
                prev_biases = bias_weights
            else:
                validation_errors.append(val_errors)
                return prev_weights, prev_biases
        print(error)
    validation_errors.append(val_errors)
    return weights, bias_weights
```

Chapter 9

Appendix B - Hierarchical Architecture Classification Code Snippet

```

def classify_with_ff_backprop_nn_dynamic_hierarchical(
    test_features_first_layer, test_features_second_layer,
                                                        test_labels,
    weights_list, bias_weights_list, activation_list):
    """  Classifies the data using the weights and
         bias weights trained through our 2 classifiers.
            Input:
                test_features_first_layer: the test data we want to classify
                in the feature form of the first layer.
                test_features_second_layer: the test data we want to classify
                in the feature form of the second layer.
                test_labels: the correct labels for the test data.
                weights_list: The weights of our 2 trained neural networks.
                bias_weights_list: The bias of our 2 trained neural networks.
                activation_list: The activation function for each classifier.
            Output:
                prints the macro_f1, confusion_matrix and accuracy.
                returns the cross entropy loss.
    """

    print(test_features_first_layer.shape)
    n = test_features_first_layer.shape[1]
    confusion_matrix = np.zeros((TOTAL_CLASSES, TOTAL_CLASSES), int)
    error = 0
    unclassified_indexes = []

    for i in range(0, n):
        x = test_features_first_layer[:, i]
        desired_output = test_labels[i]
        correct_value = np.argmax(desired_output)
        acts = []
        outs = []
        prev_out = x
        # Feed forward Neural Network
        for index in range(len(weights_list[0])):
            # Neural activation
            acts.append(np.dot(weights_list[0][index], prev_out)
                + bias_weights_list[0][index])
            if index == len(weights_list[0]) - 1:  # Softmax for Last Layer
                exps = np.exp(acts[index] - np.max(acts[index], axis=0,
                    keepdims=True))
                outs.append(exps / np.sum(exps, axis=0, keepdims=True))
            elif activation_list[0] == "sigmoid":  # activate with Sigmoid
                outs.append(1 / (1 + np.exp(-acts[index])))
            else:  # activate with Relu
                outs.append(acts[index] * (acts[index] > 0))
            prev_out = outs[index]
        # Compute the error signal
        predicted_value = np.argmax(outs[-1])
        # Count the total number of correct classifications
        if predicted_value == 0:
            confusion_matrix[int(correct_value), int(predicted_value)] += 1
        else:
            unclassified_indexes.append(i)
    print(confusion_matrix)
    for i in unclassified_indexes:
        x = test_features_second_layer[:, i]
        desired_output = test_labels[i]
        correct_value = np.argmax(desired_output)
        acts = []
        outs = []
        prev_out = x
        # Feed forward Neural Network
        for index in range(len(weights_list[1])):
            # Neural activation
            acts.append(np.dot(weights_list[1][index], prev_out)
                + bias_weights_list[1][index])
            if index == len(weights_list[1]) - 1:  # Softmax for Last Layer
                exps = np.exp(acts[index] - np.max(acts[index], axis=0,
                    keepdims=True))
                outs.append(exps / np.sum(exps, axis=0, keepdims=True))
            elif activation_list[1] == "sigmoid":  # activate with Sigmoid
                outs.append(1 / (1 + np.exp(-acts[index])))
            else:  # activate with Relu
                outs.append(acts[index] * (acts[index] > 0))
            prev_out = outs[index]
        # Compute the error signal
        predicted_value = np.argmax(outs[-1])
        # Count the total number of correct classifications
        confusion_matrix[int(correct_value), (predicted_value + 1)] += 1

    print(np.sum(np.diagonal(confusion_matrix)))
    print(macro_f1(confusion_matrix))
    print(micro_f1(confusion_matrix))
    print(confusion_matrix)
    return confusion_matrix, error
```

Chapter 10

Appendix C - Experiments and Results Hyperparameters

All results are available in a seperate CSV, in this appendix only the hyperparameters will be presented.

10.1 Best Results

Hyperparameters used to achieve results in table 6.3.

Feature: word2vec						
Arch.	**Cl. Model**	**Batches**	**L. Rate**	**HL Neurons**	**Epochs**	**Stop. N**
Flat	FFNN Relu	100	0.13	(50 , 50)	50	5
Hier	FFNN Sigmoid	100	0.15	(50, 50) , (50, 50)	50	5
Feature: Bag of Words						
Arch.	**Cl. Model**	**Batches**	**L. Rate**	**HL Neurons**	**Epochs**	**Stop. N**
Flat	FFNN Relu	100	0.08	(120, 60)	50	5
Hier	FFNN Relu	100	0.14	(120, 60) , (120,60)	50	5
Feature: TF_IDF						
Arch.	**Cl. Model**	**Batches**	**L. Rate**	**HL Neurons**	**Epochs**	**Stop. N**
Flat	FFNN Relu	100	0.14	(80,80)	50	**5**
Flat	FFNN Relu	100	0.14	(80,80) (80,80)	50	**5**

10.2 Hierarchical Classification & Flat Classification

Comparison of Hierarchical and Flat Architecture through multiple experiments:

Common Parameters

$Batches = 100$, $Epoch = 50$

10.2.1 Computationally Light

- Log Reg word2vec and Bow 0.02 - 0.09 eta
- Sigmoid (50) TF IDF 0.02 - 0.09 eta
- Relu (50) BoW 0.02-0.09 eta

10.2.2 Computationally Heavy

- Relu (50, 50) BoW 0.02 - 0.09 eta
- Relu (80, 80) w2v 0.02 - 0.09 eta
- Relu (120, 58-80) Bow, TF IDF 0.04 eta
- Relu (120, 60) Bow 0.02-0.09 eta

10.3 Relu & Sigmoid FFNN

Comparison of Relu and Sigmoid activated FeedForward Neural Networks

Common Parameters

$Batches = 100$, $Epoch = 50$

10.3.1 Computationally Light Experiments

- Flat & Hier (50) Bow, TF IDF (0.02 - 0.09 eta , 0.12 - 0.19 eta)
- Flat & Hier (80) BoW 0.02-0.09 eta

10.3.2 Computationally Heavy Experiments

- Flat (50, 50) BoW 0.02 - 0.09 eta
- Flat (80, 80) w2v 0.02 - 0.09 eta
- Flat (120, 60) Bow 0.02-0.09 eta
- Flat (120, 52-80) BoW, TF IDF 0.04 eta

10.4 Bag of Words & TF IDF

Comparison of Bag of Words and TF IDF features.

Common Parameters

$Batches = 100$, $Epoch = 50$

10.4.1 Computationally Light

- Flat, Hier Log Reg 0.02 - 0.09 eta
- Flat, Hier Sigmoid (50) 0.02 - 0.09 eta
- Flat, Relu (50) 0.02-0.09 eta

10.4.2 Computationally Heavy

- Flat, Hier Relu (50, 50) BoW 0.02 - 0.09 eta
- Flat Sigmoid (80, 80) w2v 0.02 - 0.09 eta
- Flat, Hier Relu (80, 80) Bow, TF IDF 0.12-0.19
- Flat, Hier Relu (120, 60) Bow 0.02-0.09 eta