Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

RUMOUR CLASSIFICATION IN FOOTBALL TRANSFER WINDOW TWITTER DATA

Mark Murtagh

A dissertation submitted in partial fulfilment of the degree of
Master in Computer Science

Supervisor: Professor Séamus Lawless

Declaration

I, Mark Murtagh, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signature:

Date: 11th April, 2019

Abstract

Fake news is a topic which has been circulating through mainstream media for a number of years now. Fundamental flaws in social media ranking algorithms are being exploited. Football transfer speculation news is an area in which this problem has been present for a long time. Sports media outlets and individuals have predicted player transfers claiming to be “in the know”. This research is aimed to determine to what extent supervised machine learning approaches could be used in predicting the accuracy of a tweet or Twitter account in relation to a football transfer rumour. The research project involved three parts: data gathering; data labelling; classification experiments.

The research details the steps involved in data collection, labelling the data and performing the classification experiments. Two distinct approaches were taken during the classification experiments. One classification approach using a simple multi-layer perceptron model showed promising evaluation metrics when run on unseen data. Another approach using a Separable Convolution Neural Network showed no capability of learning the features of the training data. The problems and causes of overfitting with each approach are also discussed.

Potetential issues with the training set collected were considered. Mainly the concerns with potential biasing in the methods used for data collection. Furthermore, the steps taken to accurately labelling the training data are detailed.

The findings of this research add to the extensive body of research in the area of fake news and football transfer markets. Possible sites for future work which builds on the findings are proposed.

Acknowledgements

I would like to acknowledge the work of Professor Séamus Lawless. Firstly, for the research idea and background information. Secondly for the continuous support throughout the project, and lastly for giving me the autonomy to approach each stage as I wanted.

I would also like to acknowledge the unconditional constant support from my family from the very beginning of my education all the way until now. Karen, Matt and Emma, you made all of this possible. I would also like to acknowledge my grandparents also for showing their support and always being there.

Lastly, I would like to acknowledge the wonderful class I had the pleasure of sharing my five years here with.

Table of Contents

Table of Figures.

1. Introduction

Fake news is a phrase which has been circulating through popular media in recent years. Since the 2016 US presidential election the term has gained increasing traction and has been used to criticize all forms of media. The issue originated with small groups manipulating social media algorithms and online advertising for personal financial gain. It went onto spark worldwide debate about the credibility of the news sources we use today [1]. The term soon became a sound bite and theme coherent with the presidency of Donald Trump.

This scandal exposed a series of vulnerabilities in these social media platforms. It became intertwined with other scandals such as alleged Russian Government and other organizations interfering with elections using fake social media profiles and the hacking of personal emails [2]. This coupled with news that political consulting firms like Cambridge Analytica were able to develop digital profiles that represented individuals of certain political beliefs and demographics provided a scary outlook for the everyday social media user [3].

The fake news frenzy forced US, Irish and EU Government bodies to summon social media companies' representatives before them and to seriously start thinking about heavier regulation for such companies. That being said these social media platforms are still being used daily by millions of users and the same vulnerabilities that were present before still exist today. The potential for personal financial gain still exists on these platforms and the efforts made so far by these companies involve large teams sifting through accounts rather an automated, more scalable approach [4].

The ability to extract meaning or sentiment from a piece of text is something which has been made increasingly possible through natural language processing. In the last decade companies have turned toward Machine Learning solutions to attempt to solve classification and prediction problems. A potential solution to the issue of "fake news" on social media may be possible through the creation of a model capable of classifying social media posts as "fake" or "real" news [5].

An area where rumours and fake news is not new is in sport and particularly in relation to football transfers in the English Premier League. Twice a year there are transfer

window periods where clubs can buy and sell players amongst one another. This leads to a lot transfer rumour speculation amongst the press, increasingly on social media platforms such as twitter. There are numerous accounts which have been set up with the sole purpose of reporting on this, claiming to be the some of the first individuals to be "in the know" [6].

This area gives an ideal test case of a fake news in social media. By taking a specific transfer period in the past one can look at rumours posted on social media, specifically Twitter, and use them as ground truth to label whether this rumour actually became true. Twitter provides a medium to access the hundreds of thousands of rumours and true claims, all of which can be fact checked through official records of confirmed transfers. This gives a training set to potentially develop a model capable of determining the veracity of a new, unseen transfer rumour or a social media post.

1.1 Research Question

The purpose of this research project is to answer the following question:

"To what extent can supervised machine learning approaches be used to predict the accuracy of a Tweet or Twitter account, in relation to a football transfer?"

From this question the following research objectives were defined:

- Data gathering and knowledgebase building: Create python scripts which handle the retrieval of football transfer Tweets. This process also involves creating a database of confirmed transfers which happened and collections of English premier league club names and synonyms.
- A Natural Language Processing (NLP) technique for Name Entity Recognition (NER): This objective involves using existing models to extract information from the data in order to determine the meaning behind the text. This process is to be conducted to ensure the training set examples are labelled correctly.
- Classification model development: This objective involves creating classification models using different supervised machine learning techniques, using the training data gathered in the previous stages.

1.2 Overview of this Dissertation

The first section of this paper gives a background of the research and details the problem which it is trying to address. It also defines the research question itself.

Section 2.0 gives an overview of existing research on the topic and separate topics which are related. It also gives our definition of “fake news”.

Section 3.0 details the methodology of the research. The methodology of this research can be spilt up into three distinct sections: Data Gathering; Named Entity Recognition; Classification.

The data gathering section details the process of gathering the corpus of transfer tweets, English football club names and past transfers which we know to have happened. In order to perform any supervised machine learning task suitable training data is necessary and this section details the steps taken in gathering this data.

The named entity recognition (NER) stage involved extracting meaning from the tweets gathered in data gathering. In order to correctly label each tweet as “happened/didn’t happen” the ability to extract entities and names from the texts of tweets was needed. The NER section details the methods used in extracting entities and player names from the tweets gathered, and how they were labelled.

Lastly, once the training set was available classification experiments were carried out to investigate the accuracy of different methods. The classification section details the different methods involved in constructing the feature set and the different model architectures used.

2. Existing Work

2.1 Fake News

2.1.1 Overview

Manipulating news and media outlets for personal, political and financial gain is not a new concept, and has been around so far as news and media has itself [7]. However, in 2016 we appeared to witness and ill-fitting combination between these practises and social media. The origin of the fake news social media as we have come to know it today can be traced back to the unlikely and infamous Macedonian town of Veles. In a town with an average monthly salary of $371, a group of young teens had figured out a way to make $16000 [8], around about the same time a report found that over one hundred pro-Trump fake news websites were registered to Veles. These two happenings of course were not a coincidence and as it turned out this group of teens had found a way of exploiting social media websites such as Twitter and Facebook to generate thousands of clicks to their websites which would in turn lead to revenue via Google ads for themselves.

This idea of enticing users to a click onto a website in the hope of revenue is also not new, and these so called “clickbait” tactics to generating clicks have emerged ever since it has been incentivised to prioritise clicks over good journalistic reporting [9]. However, its relevance has become increasingly important in this post-truth politics era, due to the easy access to advertising revenue and polarizing political beliefs.

Aside from groups using these techniques for financial gain, a number of reports have alleged that states such as Russia have faced allegations of disseminated false information to influence the 2016 US presidential election [10].

The phrase quickly turned into a sound bite to refer to the “lying press” for politicians as the social media fake news epidemic became mainstream news itself. Although many people primarily associate the phrase with political jargon, the problem of groups using fake news to exploit social media algorithms is still an ever-present issue [11]. Facebooks CEO Mark Zuckerberg even testified before the US Congress as a result. In late 2018 some of the world’s leading tech firms agreed upon a code of conduct to do more to tackle the spread of fake news. However, as many reports have suggested this code of conduct

provided little transparency on how to implement it. Most efforts to deter these practises have come in the form of manual human labour of shutting down payments and preventing the setup of fake accounts. However, automated detection of fake news accounts and posts still poses a real challenge [4].

2.1.2 Definition

Irrespective of the research which has gone into the area, there does not seem to be one agreed upon definition of "Fake News". However, the consensus from most studies is that it can be defined as news which includes false information designed at purposefully misleading readers [12], [13]. The core of the definition of Fake News definition is comprised of misinformation and intent. This is true for both articles and social media posts as the intent behind one binds it to the other. In other words, the sole reason for a fake news social media post is to generate clicks to the article linked in it. Therefore, for the purposes of this research, we define the definition of fake news as follows,

Fake News: A social media post or news article that is created with the intention of misinforming the reader.

2.1.3 Academic research

In terms of fake news there have numerous studies conducted with the aim of investigating fake news and researching possible detection methods. One research paper using the BuzzFeed-Webis [14] fake news corpus investigation detailed the research into what mainly comprises of a fake news post. The report found that hyper-partisan and mainstream publishers all earned verified checkmarks (official account badge) with no favourable bias toward any one type earning the badge. The same report concluded that manual binary classification between fake and real news was infeasible, as most linked articles included true and false news. Despite this, it was noted that the majority of mixed fake/real news articles belonged to hyper-partisan "right-wing" sources. Another report aimed at defining fake news [12] also confirmed the mixed true/false news nature of articles in their corpus, as it was in the BuzzFeed-Webis report.

Aside from research into investigating the contents of fake news and defining it, research has also gone into possible methods of fake news detection. Fake news detection using a naïve Bayes classifier [15] on the same BuzzFeed data set named above produced

interesting results. The implementation aimed to correctly classify the BuzzFeed article dataset as True or Fake news. The research showed that even using a simple classification approach can yield classification accuracy of 75.4%. Despite having a precision value of 0.71 and a high classification accuracy, the classifiers recall value was only 0.13. Each research paper using this corpus of articles reported the presence of mixed true/fake news articles and this low recall value further backs up their claim. The aforementioned paper results suggest that machine learning techniques could be successful in tackling this problem.

Another research papers approach [16] to the detection issue was to extract linguistic features and create linguistic feature sets. Then using said feature sets define and SVM classifier with five-fold cross validation was used in the experiment. This approach showed promise, with one classifier producing accuracy scores of 0.73 and 0.74 on different datasets. The input features were a combination of punctuation, n-grams, syntax and readability features. The same models achieved recall value of 0.74 and 0.73 respectively.

Previous academic research into defining fake news and fake news classification methods provides confidence that further advancements can be made through the use of supervised machine learning techniques to addressing the detection of fake news.

2.2 Football Transfer Rumours

2.2.1 Overview

Fake news within reporting on football transfers has is also prevalent. During each January transfer and Summer transfer window journalists and supposed sports media accounts report on player transfers, of which they claim to be "in the know", in advance of the deal being confirmed. Tactics used during the election are also at play here, however it has not undergone the same scrutiny. Nonetheless, the thousands of football transfer rumour tweets available gives the opportunity to investigate the feasibility of a model capable of classifying the veracity of a rumour, or the account which posted it.

2.2.3 Academic research

In terms of football transfers themselves there has been numerous cites of research. One frequent point of research within the area Is the relationship between club expenditure and success [17]. Research into the increasing prices of players and even using transfer markets to investigate labour mobility and globalization [18] have been conducted.

There have also been research projects more related to the topic of rumour dissemination and transfer likelihood prediction. Ireson and Ciravegna [53] detail the potential opportunity to measure the likelihood of a transfer using data collected from twitter in conjunction with Football Whispers [19]. The research also details the process of named entity recognition practises, in order to identify clubs and players in the tweet. The paper does not however provide a methodology or any classification experiments.

Caled and Silva [52] also detail the opportunity experiments on a dataset of rumours can have for rumour detection. They detail the ongoing efforts with the FTR-18 collection. This is football transfer rumour collection comprised of mostly transfer rumour publications. The entries in this dataset mainly consist of newspaper excerpts. Transfer rumours from the Pheme [20] dataset is also included. This is a dataset containing rumours related to nine "breaking news" events, which are labelled "true", "false" or

“unverified”. The research however did not provide an implementation into twitter rumour classification or detection.

Most noteworthy is the research done by Xavier [54] into investigating the natural language processing techniques and statistical analysis in determining the accuracy of certain Twitter accounts in predicting football transfers. The project defined a system to identify account which are “most accurate” in predicting transfers.
The project also explored machine learning approaches to rumour detection. A support vector machine (SVM) algorithm was implemented. The results suggested that this approach was not useful at all in predicting the veracity of a rumour. The research also stated that the feature set used of uni-grams (n-grams of length=1) were not complex enough. On top of this the paper provided a clear process for retrieving tweets. The project details the process of gathering Tweets using the GetOldTweets open source library.

3. Methodology

3.1 Introduction

This section details the methodology of this research. The methodology can be split up into three distinct sections. The first section is the data gathering section. This involved gathering the tweets to be used as the training set for performing the classification experiments. It also involved gathering additional information about football transfers. The second section involved Named Entity Recognition (NER). This section involved ensuring that the data gathered was labelled correctly, in other words was a tweet about a past transfer correctly labelled "happened" or "didn't happen" (fact/rumour). The last section involved constructing different feature sets using different methods. It also involved constructing different classification models using different supervised machine learning approaches.

3.2 Data Gathering

3.2.1 Overview

The tweets required for this research had some distinct characteristics. Firstly, for any given tweet in the data set, the tweet had to be talking about a potential football player transfer. Figure 3.1 illustrates an example of this. It had to be talking about a player either transferring to or from and English Club from another team, whether it eventually happened or not.

Figure 3.1

It also had to have been posted during a specific transfer window period. During section three if the transfer in a tweet was confirmed to have happened, the tweet had to be checked against the official transfer confirmation date. If the tweet took place after this date, then it was discarded from the dataset as it was not speculation.

In order to perform named entity recognition (NER), other information had to be gathered. To check if the entities (clubs) extracted from the tweet text contained an English club, a database of English teams had to be constructed. Also, as people tend to use nicknames and synonyms for English clubs this information needed to be retrieved also. In order to check if transfer tweet was in fact a rumour or not, a database of known transfers to have happened during these transfer periods also needed to be constructed.

3.2.2 Methods Used

BeautifulSoup is an open source python library which was used in this stage to scrape information from web pages in order to store information about confirmed transfers, club names and club synonyms. The library allows you to parse the html of a given URL into a tree of Python objects [21]. From here one can extract the elements of the html they want.

The gathered tweets and information were stored on a local database. In order to have fast access to transfer tweets, known transfers, club names and club synonyms a database program called Mongo DB was used. Mongo DB is a NoSQL database program which

allows one to store JSON-like documents [22]. The attraction to Mongo DB is that it allows for fast access to database information, while providing a simple method for defining a database schema and making entries. It also allows for easy deployment in any location and has extensive documentation. For the purpose of this project Mongo DB was deployed to a local machine using PyMongo, a python driver for MongoDB. This allowed for information to be scraped from the source websites and then be saved directly into a database collection. The *"reset_collections"* method in the *"db.py"* module is set up that so as long your local machine has MongoDB installed, the method will scrape and store locally the transfer information corresponding to the provided links.

Twitter provides the means to search through the history of tweets posted on their site through the Twitter API [23], however the tweets retrieved is limited to tweets within the last 7 days for non-paying users. *"GetOldTweets"* is an open source project which allows one to bypass this [24]. This project takes advantage of the JSON loader used by modern web browsers and allows you to query tweets that go as far back as Twitters beginning. The tool allows you to pass in a query term or sentence along with date parameters so one can choose the specific period to retrieve tweets from. This tool was used to build the training data set.

3.2.3 Implementation

Gather Confirmed Transfer Information

The first information collected was the known confirmed transfers to have happened. For known transfers the source used was Wikipedia [25].

Each transfer window Wikipedia page contains a table of transfers and loans which happened during a given window. The first information that was scraped and stored were confirmed transfers. Using BeautifulSoup this information was parsed, extracted and written to a local database. A MongoDB database was created for the whole research project called *"transferdb"*, and a collection (table) was created in this to store confirmed transfers. As detailed in section 3.3 this information was eventually used to check if the transfer being discussed in a tweet actually happened or not.

Gather Known True Tweets

Two types of transfers tweets needed to be retrieved. Tweets which contained transfer speculation which eventually went onto becoming confirmed (true tweets), and tweets containing transfer speculation which never ended up happening (rumours).

GetOldTweets allows you to execute the script with command line arguments, as shown in figure 3.2.

```
python GetOldTweets/Exporter.py --querysearch "hello" --maxtweets 4000 --since
  2018-01-01 --until 2018-01-31 --output res.txt
```

Figure 3.2

Figure 3.2 shows an example which will return all tweets containing the word "hello", from the 1st – 31st January 2018, limiting them to the most recent 4000 and outputting the results to *res.txt*.

Due to the massive amount of content on Twitter the quality of the content returned from GetOldTweets solely depends on the *querysearch* parameter. Vague query terms lead to unrelated tweets being returned. In other words, terms like "in the know" or "deadline", which are usually associated with football transfer content, returned tweets completely unrelated to football when used by themselves.

The first approach taken was to take the known transfers which had been retrieved earlier and using them generate a query containing the name of the player and club. This way the results returned would be narrowed down to specific tweets talking about a specific football transfer. Figure 3.3 shows the commands generated for a given know transfer. The methods in the *db.py* module were responsible for generating commands like these.

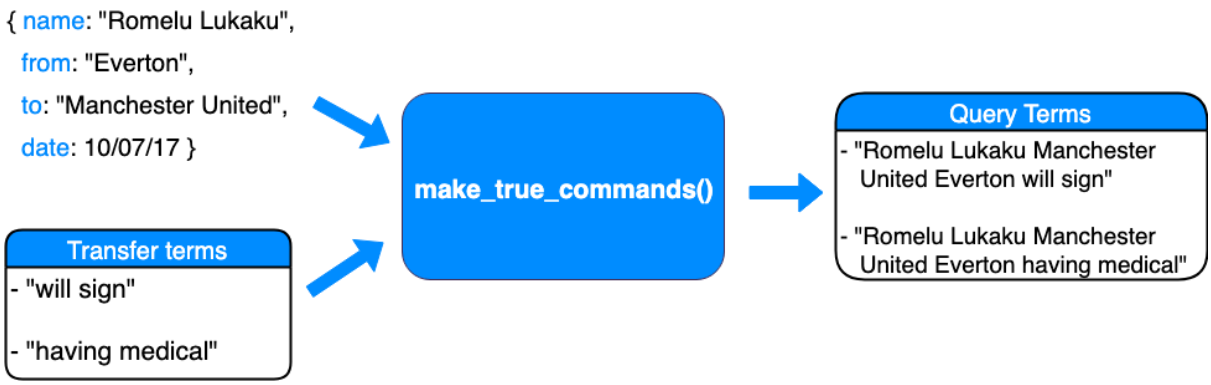


Figure 3.3

This was done for every known confirmed transfer for a given window. Many of the functions present in *relations.py* module were created with the intention of iteratively

going through each confirmed transfer, generating the query terms for that transfer and writing the GetOldTweets command corresponding to it to a bash script. The *"—until"* flag of the command was set to the day before each transfer was officially announced, this way only speculative tweets would be retrieved and not tweets after the transfer had been confirmed.

The *realations.py* module handled all of this and created a bash script containing all the confirmed transfer GetOldTweets commands. The functions in the module were set up to write a "wait" command every ten lines, so the script could run ten Python processes whilst not slowing the operating system of the machine. This way, upon running the script, ten different GetOldTweets commands with different query variations would execute and write their result to a shared text file. Figure 3.4 shows a snippet from the bash script generated for summer 2015 confirmed transfers.

```
python ../GetOldTweets/Exporter.py --querysearch "N'Golo Kanté  Caen Leicester
  City transfer to" --maxtweets 100 --since 2014-12-01 --until 2015-8-2
  --output ../info/true.csv &
python ../GetOldTweets/Exporter.py --querysearch "N'Golo Kanté  Caen Leicester
  City will sign" --maxtweets 100 --since 2014-12-01 --until 2015-8-2 --output
  ../info/true.csv &
python ../GetOldTweets/Exporter.py --querysearch "N'Golo Kanté  Caen Leicester
  City having medical" --maxtweets 100 --since 2014-12-01 --until 2015-8-2
  --output ../info/true.csv &
```

Figure 3.4

Figure 3.4 shows three different commands. All commands query terms contain the player *"N'Golo Kanté"*. However, each query term differs in that they all contain different transfer talk phrases appended at the end. Each of these commands with different "*–querysearch"* parameters ran in parallel and wrote their results to the shared text file. Also note the "*–until"* parameter is set to the day before this signing's official announcement date of August 3rd, 2015. The transfer talk phrases appended to each query term was a set of phrases. This set was constructed after manually reading through known transfer news Twitter accounts and observing the most common words used in these Tweets.

After running the bash script for each transfer window period, a collection of known true tweets were returned. Although they were retrieved using query terms generated from known true transfers, they were not labelled true until the named entity recognition methods were performed, as detailed in section 3.3. The results text file generated from

running this bash script was then written to a database collection using methods constructed in the *db.py* module.

Gather Rumour Tweets

The previous process resulted in gathering tweets which mainly contained claims of transfers which happened. However as stated before, for a balanced training set you need examples of transfers which didn't happen (rumours).
Generating the query terms for these GetOldTweets commands provided to be much more difficult. For this, three main approaches were used. It's important to note that none of the tweets could be labelled as a false rumour until the named entity recognition methods described in section 3.3 were performed, so in many ways there was a lot of back and forth between the two sections.

The first approach was to observe the already gathered tweets and determine the most frequently occurring Twitter accounts in the collection. In other words, identify accounts that actively tweeted about football transfers. By doing so one can retrieve all the tweets they sent within a given period using the GetOldTweets *"–username"* flag instead of using a query phrase. Pandas [26] is Python Library which allows you read data into a Dataframe structure. This structure has a range of functions and operations associated with them to extract data or operations on the data contained in it [26]. Using this the *"find_top_tweeters"* method was constructed in the *relations.py* module, which read the collection of tweets gathered in the previous section and returned the top *N* tweeting accounts. Then using the account usernames, each account's tweets within a transfer window time period were retrieved. This method was somewhat effective but lead to the retrieval of unrelated, non-transfer related tweets. Despite having methods dedicated to filtering out non-transfer talk in section 3.3, there still were un wanted Tweets which slipped through the filtering. Also, the larger the corpus that had to be filtered the longer this process took, so having a data gathering method which retrieved as much useless information and useful information was massively inefficient.

The second approach involved creating general "transfer talk" query terms and passing them as the "*—querysearch"* parameter to the GetOldTweets command. These query terms contained transfer phrases such as "in the know", "having a medical", "close to signing". The idea was that a corpus of Tweets would be stored, and the methods defined in 3.3 would filter out the Rumours and True Tweets. Similar to the previous approach,

this returned numerous Tweets unrelated to transfers, or containing vague material not talking about an exact player transfer. For the same reasons mentioned in the previous approach this would require spending time filtering a corpus of Tweets primarily consisting of useless entries, some of which would make it through the filtering phase.

The third method used, and most effective for retrieving Tweets containing the most amount of rumours, was by using an approach similar to the first. The initial query terms were generated using known confirmed transfers, the same method was applied here. However, given that rumours are in fact rumours and didn't end up happening there were no transfer relationships to make the query terms. For this reason, synthetic relationships (false relationships) were generated. These were essentially a mapping of a player to a club during a transfer window which was known not to have happened. These relationships were mainly generated using manual research into the top transfer rumours during certain periods. Football Whispers [19] provided as an excellent resource to find recent transfer rumours. An example of these relationships can be seen in figure 3.5, where a given player is mapped to several clubs. These relationship mappings were then used to generate query terms, and from this GetOldTweets commands could be generated. The transfer language phrases appended to the query terms were the same ones used in generating the commands for the "known true" tweets. This method resulted in a large corpus of possible rumour tweets about the synthetic (known false) transfers generated.

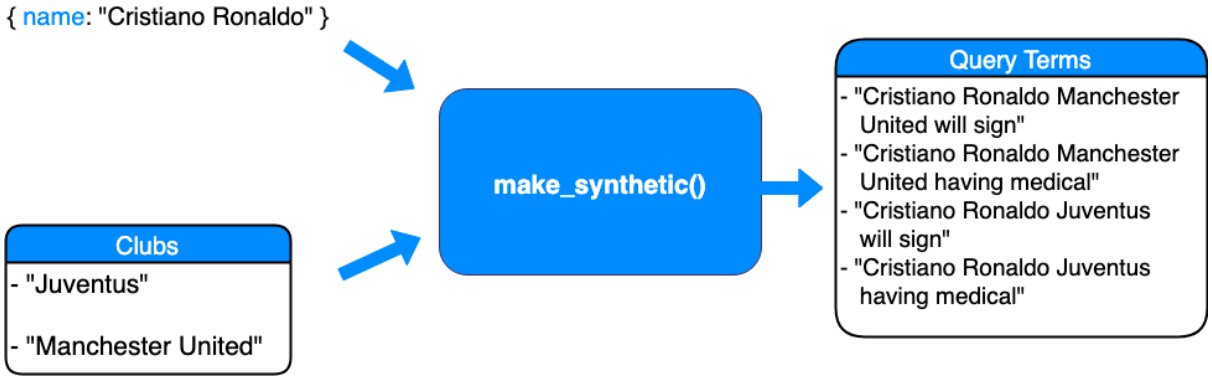


Figure 3.5

Gather Clubs Names

In order to extract possible entities from a tweet's text field, a collection of the entities being searched for was necessary. Unless the results of the NER model used in section 3.3

id	name	league	nicknames
5c85e7f2	"Wolverhampton Wanderers"	1	"Wolverhampton", Wanderers","Wolves"

could check the possible club names of the tweet text with actual club names then its use would be futile. A Wikipedia page consisting of a table of all English football clubs and their respective divisions was used as the scraping source for this [27]. The source was updated to the current season (2018/2019). Due to the number of clubs and the tendency of the media and individuals to tweet about club transfers in the higher divisions, the top four divisions of English football were used for the purpose of this research paper. The table also included each club's official nickname which was stored with each club entry, as club nicknames are commonly used to refer to clubs.

During this stage there was also effort put into generating club "abbreviations". Its common practise for fans, individuals and media outlets to use abbreviations when referring to a club. For example, a club like "A.F.C Bournemouth" is commonly referred to as just "Bournemouth". The "*generate_syns*" function in db.py was created to handle this, and each club name entry into the collection also contained abbreviations to ensure all club names, nicknames and abbreviations were stored for the NER phase. This was done by essentially splitting the club name on whitespace. Abbreviations that such as "FC", "AFC" were not included as they are common to many club names. Figure 3.6 show and example database entry associated with a club name.

Entry: Wolverhampton Wanderers

Figure 3.6

3.2.4 Issues

Making the queries themselves provided some issues and concerns. Coming up with a method of generating suitable query terms for the GetOldTweets commands proved

difficult. The primary concern at this stage was if the differing methods used in retrieving *true* transfer tweets and *rumour* tweets would infer some sort of a bias. The relationships defined to generate the synthetic false transfers were heavily influenced by manually searching reports from media outlets. The concern was that less tweeted about rumours would be missed in this data gathering phase due to the focus on "more newsworthy" rumours about top players.

Due to the "retweet" feature on twitter there were many duplicate entries in the data set. There were also tweets from different accounts with the exact same text where several accounts were all quoting the one source. During this stage there was deliberation as to whether this was an issue or not. On one hand, during classification, if the model is seeing the same tweet text frequently that is not ideal. On the other hand, if the model was to have several other input features like "retweets" and "likes" then It may be useful to research the relationship between identical tweet texts with different reach. The decision was made to remove duplicate entries. However, after making these efforts to removing duplicates some still remained, due a slight character difference.

3.3 Data Labelling

3.3.1 Overview

This stage involved labelling the tweets gathered in section 3.2 as a *"true transfer"* or *"rumour"*. In section 3.2 transfer tweets, club names and club synonyms were gathered. This section details the process of taking all this information and creating a module which could take all this information and correctly label the data. Writing a module to do so

involved a lot of different parts. The module which primarily handled all of this is called *ner.py*. It handles all of the filtering, and labelling of the gathered tweets. For the training of the classification models it was important to ensure that the examples it was being trained on were labelled correctly.

3.3.2 Named Entity Recognition

In order to determine whether or not a past tweet was a rumour or a true claim, or if it was even talking about transfers at all, a method for extracting this information was needed. Figure 3.7 shows a sample tweet. The tweet is clearly about Zlatan Ibrahimović moving from Manchester United to LA Galaxy. However, this needed to be done for every tweet in a dataset of over 140'000 entries. Given the confirmed transfers and English club names for each window were gathered in section 3.2, the information was present to verify this claim, and label it correctly.

Figure 3.7

This is where Natural Language Process (NLP) techniques were used. Natural Language processing is an area which is concerned with how programs can process and analyse large amounts of data. Powerful toolkits have been developed in the area to process text by tokenizing it, tagging it (part-of-speech tagging), and recognising entities in it [28].

In order check if a tweet contained an English football club the potential entities had to be extracted from the text. In order to confirm whether a transfer or not happened, potential clubs and player names needed to be extracted.
There were two main approaches to extracting entities from tweets in this stage.

The first approach was to perform part-of-speech tagging on a tweets text using the Natural Language Toolkit (NLTK) [29]. Part-of-speech tagging (POS tagging) is the process of marking up a word in a text as corresponding to a part of speech [30]. NLTK is an open source Python module which provides NLP methods such as tokenization, stemming and part-of-speech tagging. The initial approach to recognising entitles was using NLTKs POS tagging function and from there separating the NN's and NNP's (nouns, nouns plural). This however resulted in a lot of miss tagging of the tweet text.

The second approach involved using a different library called SpaCy. SpaCy is an open source NLP library that provides a wide range of pre-trained models and extensive documentation on retraining existing models [31]. Spacy aims to provide production ready models over research implementations, so integrating it with existing code in the project was a seamless process. SpaCy provides a NER model which was trained on the OntoNote 5 text corpus [32]. This was is a corpus comprised of various different text documents from news, telephone speech and blog in three different languages. It was a collaborative effort by the University of Colorado, University of Pennsylvania and the University of Southern California. The SpaCy NER recognition model attempts to extract possible entities from a text corpus and does so by returning a tokenized and tagged object. There are several entity types which it supports, including: "PERSON", "ORG" (organisation), "NORP" (national or religious political group), "GPE" (Countries, cities), "DATE" and "MONEY". The SpaCy NER recognition model was integrated with the ner.py module. Figure 3.8 shows the result of the NER model on a particular tweet text field.

Tweet text: *"Zlatan Ibrahimović set to leave Manchester United imminently and to join MLS side LA GALAXY"*

Terminal output:

```
$ python ner.py
[(u'Zlatan Ibrahimovic', u'PERSON'),
 (u'Manchester United', u'ORG'),
 (u'MLS', u'ORG'),
 (u'LA GALAXY', u'ORG')]
```

Figure 3.8

3.3.3 Implementation

The *ner.py* module was developed to take the data set of gathered Tweets in previous stage and label them correctly, as a rumour or true, by performing named entity recognition and verifying the clubs and players mentioned in the tweet using the database of confirmed transfers and English football cubs. Figure 3.9 illustrates the filtering process each Tweet went through before being labelled.

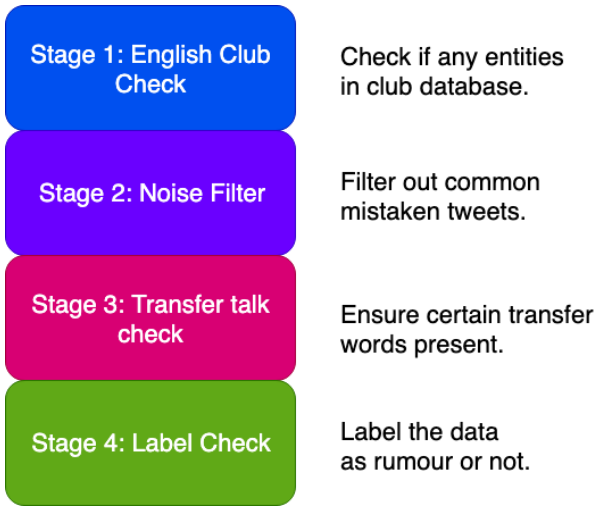


Figure 3.9

The *"process_tweet"* function in ner.py is the focal point of the module. From here the gathered data is read and one by one each tweet is processed and labelled. Figure 3.9 conceptually breaks down what's happening here as there are many functions in the module which handle of this.

For a given tweet the elements of the Tweet are separated, namely the username and the tweet text field contents. Then the entities are extracted using the SpaCy's NER model.

These entities are then passed to other functions which attempt to extract potential players and clubs from the entities. Next the "confirmed transfers" database is queried to see if the potential players are present.
Then the four filters mentioned in figure 3.9 are applied. Stage 1 checks whether or not and English was present in the entities. If not, this tweet is skipped as this research project is only concerned with transfers involving English football clubs. Stage 2 separated out known keywords which are predominant with tweets which aren't talking about transfers but are similar enough to be picked up the data gathering phase. Examples of these tweets are matchday line-ups, betting odds, injury news and contract extensions. Stage 3 ensures that some of a small set of transfer phrases are present in the Tweet. The keywords in this filter were constantly adjusted to cater to a more lenient or strict filtering process.

If a tweet past the first three stages of filtering it was then time to determine whether it was a rumour or not. If there was a player mentioned in the Tweet text that was present in our database, then check if the club the player moved to was present in the Tweet Text. If so, if this tweet is talking about a transfer which happened, label "True" and store it in the corresponding database collection. Else, if there were no known club names or club nicknames associated with our database records then label this Tweet as "False" (rumour) and store it in the corresponding database collection. If there were initially no known players in the Tweet text that were in the database, check if there were still potential players in the text. If so, store as rumour, if not discard Tweet.

Below is a pseudo code implementation of the filtering and labelling described previously.

```
def process_tweet():
    transfers = datebase.read()
    for i in transfers:
        tweet_text = i["text"]
        username = i["username"]
        entities = get_entities(tweet_text)
        potential_players = get_potential_players(entities)
        potential_clubs = get_potential_clubs(entities)
        db_result = check_confirmed_transfers(x)

        if english_club_check(pclubs):
            if noise_filter(tweet_text):
                if transfer_talk_check(tweet_text):
                    if len(db_result)>0:
                        if(already_confrimed(db_result,
tweet_text)):
                            true_database.store(i)
                        else:
                            false_datababse.store(i)
                    else:
                        if len(pplayers)>0:
                            false_datababse.store(i)
```

3.3.4 Issues

Not picking up clubs as entities

The first predominant issue was that the SpaCy NER model wasn't picking up some English football clubs as entities. The NER model used was trained on a corpus of news and conversational text. The entity words in these examples are different to the entity words of football teams. For example, if we take the football club "Arsenal F.C". In the context of football an individual with background knowledge of English football clubs could easily identify this as a London club. However, Arsenal in the context outside of football could be referring to military artillery or guns. This happened for numerous other examples.

The solution to this was to manually read through Tweets where no entity had been identified. Usually there were several examples of a football clubs not being picked up as an entity's in numerous different tweets. SpaCy provided mechanisms for retraining their models. Therefore, retraining methods were implemented in *ner.py* to retrain the SpaCy standard English NER model. After the functions were implemented the clubs which were recognised had to be collected along with examples of them being used in a sentence or piece of text. Using this the NER model was retrained and the previously not recognised clubs were recognised as entities.

The exact same issue happened for players whereby the NER model would not recognise some players as a person. The same process was used to retrain the model. Naturally, there's a trade-off to retraining a model for an example and the frequency of that example. For this reason, only some examples were chosen to be fixed. In other words, retraining a model for a player or club which was only present in less than ten tweets would not be an efficient use of time.

Ambiguous non-specific transfer tweets

Upon inspection, the line drawn between "true" and "rumour" is not clear cut. Ambiguity lies on where some examples fall into the "true" or "rumour" category. An example of this kind of tweet was a transfer that had sensationalised language, a link to a page or article but wasn't talking about a specific transfer. Many of the tweets implied a certain player transfer by naming their nationality but without specifying their actual name.

Another ambiguous type were tweets which contained transfers which were known to be true but other transfers which were known to be false, as shown in figure 3.11. Many tabloid publications seemed to group current transfer rumours in a "round up" or "summary" tweet and provide a link to an article on their website.

Figure 3.10

These examples provided uncertainty as to whether they should be included in either dataset. The worry was that by labelling these examples as false or true during this phase it could end up effecting the model weights during training in section 3.4.

To tackle this issue random samples from the labelled data were selected and manually checked. The tweets which contained a mix of true and false transfers claims ended up labelled "true" as the stage 4 filter picked up a confirmed transfer in the text. The decision was made to keep some of the tweets in the dataset labelled as they were on the basis that a tweet claiming to know about a transfer which never happened still falls under our definition for "fake news". Some tweets were the exception to this, mostly in cases where the rumour was the minority compared to the other true transfers present in the tweet, and they were removed.

In the case where no explicit player name was mentioned in the tweet, these examples were removed. The functions created in *db.py* removed any occurrence of the parameter passed, and usually there were many occurrences of the same tweet from different accounts with this example case. This is most likely because tabloid news outlets frequently use tactics like an "implying headline" but not give any information to the reader until they clicked the re-directing link to their site. Although this re-direction sounds like the "fake news" definition defined in this research, there was no automated way of accurately labelling these tweets unless a player was mentioned which could be cross referenced with the database of known transfers.

Incorrectly Labelled False

Another issue which arose in the early stages of NER was the issue of incorrectly labelling tweets as "false" even though they were true transfers. This issue arose from the lack of filtering and poorly trained initial SpaCy NER model. It's important to note that the

decisions to implement the filtering stages and retraining of the model were not methodologies planned from the outset but rather a result of checking the labelled data and seeing they were necessary.

Label Accuracy

All the observations and conclusions described above about the accuracy observed and the issues which arose was a result of manual checks. When developing the *ner.py* module consistent checks of samples of the tweets were performed to check the accuracy of the data labelling.

The final testing was performed by taking 7 random sets of 100 from a corpus of 3000 labelled Tweets.

Results:

Correctly labelled	Ambiguously Labelled	Incorrectly Labelled
74%	17%	9%

Due to the data gathering methods used it tended to be "false" rumours which were labelled more incorrectly. This is most likely due to the specific query construction mechanisms in retrieving "true" transfer tweets, and the synthetic query generation of the "false" transfer tweets retrieval.

Due to the limited time frame of the research project the decision was made to accept the accuracy and falsely labelled scores of the filtering stage and make a best effort to clean the corpus of known ambiguously labelled tweets.

3.4 Classification Experiments

3.4.1 Overview

This stage involved taking the labelled corpus of tweets from section 3.3 and performing different text classification experiments. Text classification is an aspect of supervised machine learning (ML). Text classification is implemented in many different web applications today, from email spam filtering to review sentiment analyses. Classification where something is sorted into a topic, such as an email being "spam" or "not spam", is an example of topical classification. Sentiment analysis is another form of text classification, whereby the goal is to the determine the polarity of the texts content. This could be a binary or multi-class classification process. Essentially sentiment analysis provides a mechanism to take a logistic regression approach to text classification and output a probability score that a piece of text belongs to a certain class.

The training data collected in this research project was separated into one of two classes, "true" or "false", depending on whether the football transfer spoken about in the tweet was verified to have happened or not in section 3.3. This dataset provides a basis to examine the prediction performance of different models trained on our football transfer tweet dataset.

There were two main approaches to text tokenization, vectorization and model architecture during this research project. This section details the two different approaches taken and the reason behind the them. It also aims to provide the implementation of each process in chronological order and the reasoning behind both approaches. For the sake of this research topic we will refer to the two different approaches as *approach A* and *approach B*.

3.4.2 Tokenisation/Vectorisation.

In order to feed the training data into a ML model it must be in a format the model understands. For typical linear regression and logistic regression problems the data will already tends to be in numeric form, whereas text examples retrieved will be in their text form. For example, if one was trying to construct a model to predict the housing price in an area, the training data would likely be a set of house prices from the area. Our training is not in numeric form. It is tweets with text fields, usernames, retweet numbers and date values. This means that the text field values need to be converted in to numerical vectors.

There are two steps to this process, *tokenisation* and *vectorisation*.

Tokenisation: This is the process of dividing the text into sub texts which are called "tokens". This enables a generalization of the relationship between data and label. Tokens can be divided as small as words, groups of words, or even whole sentences. This process determines the vocabulary of the dataset.

Vectorisation: This is the process of defining a numeric measure to characterize the texts in the dataset.

Approach A

Tokenisation

In this approach the tokenization was performed using n-grams. N-grams are sequences of adjacent items, which in our case are words. If we take the sentence *"Ben Watson poised to sign deal with Nottingham Forest"* the subsequent n-grams are as follows:

N=1 (unigram) : ["Ben", "Watson", "poised", "to", "sign", "deal", "with", "Nottingham", "Forest"]

N=2 (bigram) : ["Ben Watson", "Watson poised", "poised to", "to sign", "sign deal", "deal with", "with Nottingham", "Nottingham Forest"]

N=3 : ["Ben Watson poised", "Watson poised to", "poised to sign", "to sign deal", "sign deal with", "deal with Nottingham", "with Nottingham Forrest"]

With n-gram representation word order and grammar is discarded to an extent. N-grams with n>1 can maintain some partial ordering but a larger value of n runs the risk of overfitting [33]. This approach was used in conjunction with a model which does not take ordering into account, as detailed in section 3.4.3. This is called the "bag-of-words" approach and is considered a simplified approach to tokenization.

Vectorisation

Term frequency – inverse document frequency (Tf-idf) is a measure that reflects how important a word is in a corpus. Tf-idf score increases with the amount of times a word appears in an example and is offset by the number of examples in the corpus. This penalizes words which appear frequently in all documents, words which are not unique to an example document.

This was the approach used for vectorisation in *approach A*. Scikit-learn [34] is an open source machine learning library containing many features from text vectorization to classification algorithms. They provide a tf-idf vectorizer which converts a corpus into a matrix of tf-idf features, and vectorizes words based on this.

Over 140'000 tokens were present after Tf-idf vectorization. Not all tokens contribute to label prediction, especially if they occur very rarely in the dataset. In order to calculate a feature "importance score" in predicting the output, an 'f_classif" function was used to select the top K important features in the dataset. F_classif is Scikit-learn's implementation of the Anova F ratio, which calculates the ratio between two mean square values [35]. K was set at a value of 20'000 based on a similar text classification implementation [36]. Varying values for K were tested form this point.

Approach B

Approach A's text vectorisation approach and model, which is discussed in section 3.4.3, are implementations which do not take word order into account. Approach B's

vectorisation and model architecture was implemented to take sequences into account. It was done so on the belief that order matters in tweets. For example *"Ben Watson poised to sign deal with Nottingham Forest"* can only be fully understood when read in that order.

Tokenisation

For this approach tokens were represented as words. This was done on the basis that you can map the semantic similarities between words but the frequency of phrase consisting of numerous words would be low, especially in a data set of 140'000 Tweets. The Keras standard text pre-processing tokenizer was used to split these texts into words [37].

Vectorisation

Vectorisation for approach B was done using word embeddings. Sequence models like convolution neural networks (CNNs) can infer meaning from an order or sequence. The use of word embeddings in text classification have been adopted for their capability to extract semantic similarities [38]. The idea behind a word embedding is that a word can be represented in a vector space, and semantically similar words can be located closer to each other. This way the location and distance between two points can represent how similar or different they are semantically, as represented in figure 3.1.

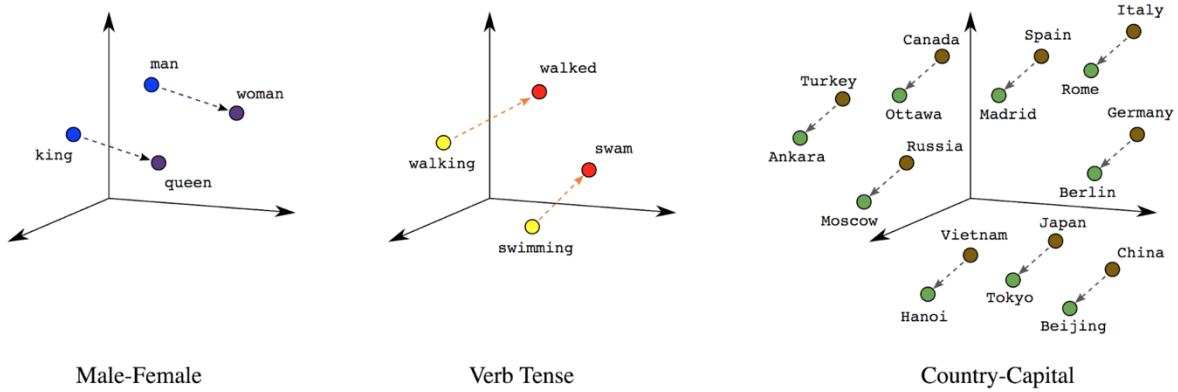


Figure 3.11 Preparing data [36]

For *approach B* this vectorization technique was used by having an embedding layer present in the model architecture, as detailed in section 3.4.3.

3.4.3 Model Architecture

As detailed in the previous section the text tokenisation and vectorisation approaches in *A* and *B* differed mainly in disregard and regard for sequence respectively. The architectures of the models followed suit.

Approach A

A multilayer perceptron model (MLP) was used for *approach A*. An MLP model is an artificial neural network which consists of at least three layers. This simple model does not take into account sequences and requires low computation. MLPs have also been shown to provide high accuracy scores in text classification. Some research suggests that MLPs can match hidden Markov model accuracy levels, which are widely used in speech recognition [39].

Approach B

Unlike *approach A*, the model defined in *approach B* was done so to take advantage of the adjacency of tokens. A model which does so is referred to as a sequence model. Convolutional Neural Networks (CNN) have been at the forefront of image classification when it comes to machine learning. A CNN is a deep learning algorithm which takes an image as an input and uses techniques in convolution layers to reduce the image into a

form easier to process. It does this without losing the important features which are used to make a good prediction [40]. Research into CNNs for text classification has also been prevalent and has shown promising results for tasks such as sentiment analysis [41].

Depthwise Separable Convolutional Neural Networks is a CNN implementation with separable convolutions. A depthwise convolution is one where the input channels are kept separate and the two-dimensional filter is applied across each channel, as oppose to a regular convolution where the filter can be as deep as this input. Figure 3.13 shows the difference between a Basic and depthwise convolution with 4 dimensions.

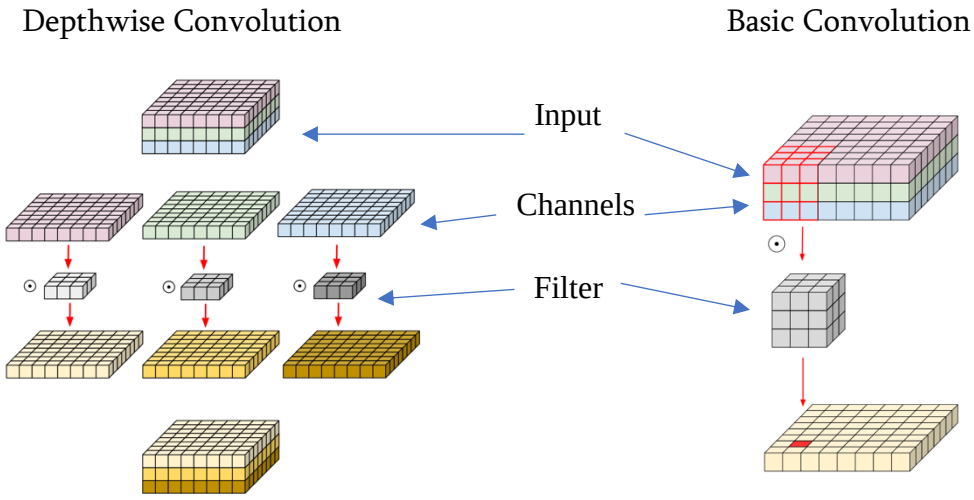


Figure 3.12 Depthwise separable convolutions for machine learning [55]

A depthwise separable convolution involves one extra step to the depthwise convolution described above. An additional step is performed across all channels as shown in figure 3.13

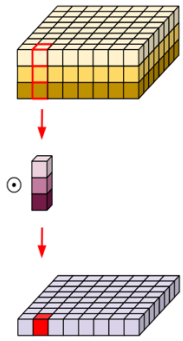


Figure 3.13 Depthwise separable convolutions for machine learning [55]

Depthwise separable convolutions in model architectures have shown increased classification performance when compared to Inception V3, a network architecture which had been favoured considerably for optimum CNN performance [42]. For this reason, the separated CNN model architecture was chosen for this approach.

The embedding layer was set up using a Keras layers. The model architecture was set up so pretrained embeddings or fine-tuned embeddings could be used. Pre-trained embeddings are when embeddings learnt from another dataset is transferred into a model's embedding layers. This has the possibility of giving the model a "head start" on training, but equally could be counter intuitive if the embeddings were learnt from a different context.

3.4.4 Implementation

The experiment process for both approaches was the same. Define an architecture based off previous research and the previous experiment iteration. Train the model and observe the accuracy, area under the curve (AUC) and loss scores during training and validation. Tune the models hyperparameters based on values observed. Once the model and parameters used showed promising training results, evaluate the model on the test set.

A hyperparameter is a parameter of a model whose value is set before the learning process begins. Its common practice in ML to choose initial hyperparameters based off other related research. However, the first choice for these values will not ensure the best results. The hyperparameters tuned for both approaches were different, as detailed in figure 3.14, and figure 3.15 explains the role of each parameter.

Approach A	Approach B
Learning Rate	Learning Rate
Epochs	Epochs
Batch Size	Batch Size
Layers	Layers
Units	Units
Dropout Rate	Dropout Rate
	Filters
	Kernel Size
	Embedding Dimensions
	Pool Size

Figure 3.14

Hypermeter	Description
Learning Rate	The learning rate ("step size") determines the amount the weights are updated during training.
Epochs	Number of times all the training vectors are used once to update the weights.
Batch Size	Number of training examples used in one iteration.
Layers	Layers refers to the number of hidden layers in the model, excluding input and output layers.
Units	The number of nodes in each hidden layer
Dropout Rate	Regularization form. Number of units who have their activations randomly dropped for a gradient step.
Filter	Dimensions of filter applied to input which produces convoluted feature.
Kernel Size	Window dimensions of convolution.

Embedding Dimensions	Dimensions of embedding vectors.
Pool Size	Pooling size. Reduces the dimensions of feature map whilst retaining important information.

Figure 3.15

Jupyter Notebook is an open-source web application which allows for the output of code snippets to be displayed in the browser window [43]. This environment was used for the classification stage of this research project.

The final corpus of tweets gathered contained 140'000 Tweets. The number of samples per class was almost spilt exactly evenly between "true" and "false" rumours. The dataset was therefore balanced.

Common practise in ML is to split your gathered data into set for training and testing. The set you should set aside for training should be further divided into "training" and "validation" sets in a ratio of 80:20. Figure 3.16 shows the breakdown of the of the data set for this research project.

Data set size: 140'000

Figure 3.16

The data was loaded straight from the local mongo database into Pandas dataframes. Pandas [26] is a Python library for data manipulation and analysis, which offers a dataframe object with many different methods. After separating the data in to their respective sets, the sets were shuffled to ensure an even distribution of classes.

The MLP model was defined using Keras layers. The *"mlp_model"* function is designed to take in layer, units, dropout rate, input shape and class number as parameters and create a model accordingly. The main activation function used was Relu. Sigmoid was also experimented with but the initial and final activation function used was Relu. The Relu activation function has been favoured in NNs because of its ability to solve the "vanishing

gradient" problem and accelerated convergence speed [44]. A dropout rate was set at each layer and different values were experiment with. Dropout is a form of regularization used in neural networks [45]. It works by randomly dropping unit activations for a single gradient step, at a predefined rate.

The hyperparameters used in the MLP model were also used in creating the sepCNN model, with the addition of other parameters specific to CNN's. The first layer defined in the sepCNN was an embedding layer followed by the convolution blocks. The pooling values for each convolution was also experimented with but as described in section 5.0 a set model was defined for the purposes of tuning other hyperparameters. The set pooling was max one-dimensional pooling in all hidden convolutions with the exception of average pooling in the convolution before the output layer.

On top of the functions created to define the two models, there were ones created to measure the performance metrics. Training and validation loss and accuracy functions were created, as well as a function for plotting the receiver operating characteristic curve.

4. Results

4.1 Overview

This section details the results of the classification experiments performed in section 3.4.4. It also explains the metrics used to evaluate the classification performance of both approaches.

4.2 Metrics Used

4.2.1 Accuracy

Accuracy is defined as the *number of correct predictions out of the total number of predictions* [46]. For binary classification accuracy can be calculated as shown in figure 4.1. Accuracy essentially tells us the fraction of predictions which the model got correct.

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

Figure 4.1

Where TN = True Negatives, TP = True Positives, FP = False Positives, and FN = False Negatives.

In a problem where there is class imbalance accuracy can be misleading. In other words, if class A represents 99% of the set then a binary classification model could have a bias to always predict for class A. Here a high accuracy score may be achieved but it doesn't mean the model is good at predicting whether or not an example belongs to a specific class. The training set for this research topic is almost perfectly balanced, therefore we can rely on accuracy as a good evaluation metric.

4.2.2 Loss

Loss is an indication of how bad a model is performing on an example. Binary cross entropy (Log loss) was the loss function used for this experiment. This loss function measures the performance of model whose output is between 0 and 1 [47]. Essentially the loss value increases as the predicted probability moves away from the examples label.

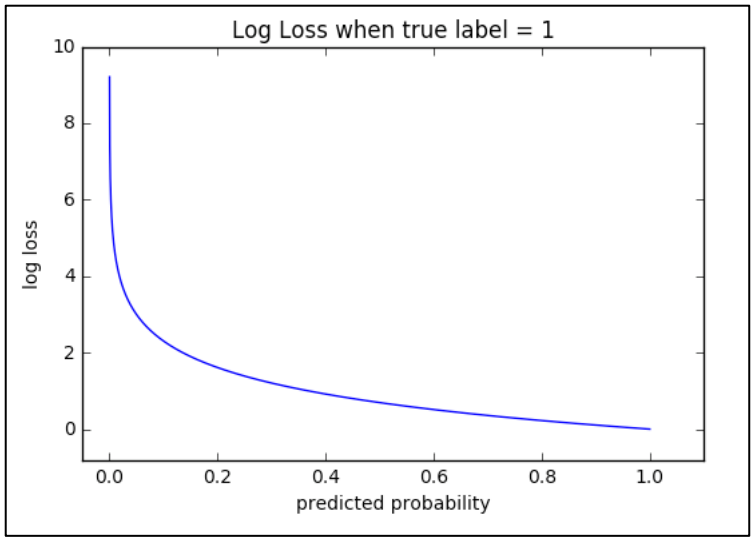


Figure 4.2

4.2.3 Area Under Curve

A classification threshold is the threshold a model uses for separating an example into a given class. For example, if we set the classification threshold for our model to 0.70, it means we would only mark an example as true if the output of our model exceeded the 0.70 mark. True negatives, true positives, false positive and false negative give us a useful error categorizing tool. They give us two important more metrics: True positive rate (TPR); False positive rate (FPR).

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

$$FPR = \frac{FP}{FP + TN}$$

Figure 4.3

While training a model it may not be clear what the classification threshold should be. A *receiver operating characteristic curve* (ROC Curve) is a graph showing the performance of the model at different classification thresholds [48]. It's a created by plotting the TPR against the FPR.

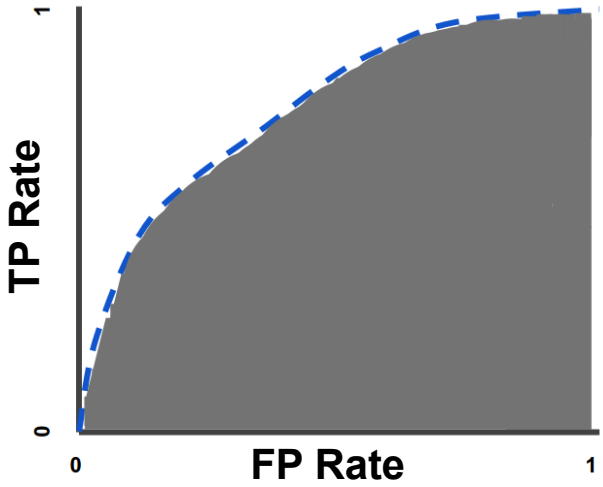


Figure 4.4 Classification: ROC Curve and AUC [56]

Evaluating the model at different classification thresholds would give these values but it would be very inefficient. By calculating the area under the curve (AUC) we get an aggregate measure for performance across all thresholds. This is much more efficient than evaluating the model several times with difference thresholds.

4.2.4 Machine Learning metrics

It's important to note that common practise amongst ML approach evaluations is to not solely rely on one metric to measure performance. Section 4.2.1 details how accuracy may be misleading. The same goes for loss. Loss can be an indication as to how well your model is fitting the training data, but the loss value in testing may be completely different. This is a common occurrence called "overfitting" which essentially happens when the model learns features of the training data too well.
In essence one of these metrics alone isn't sufficient to determine how good a model is. Multiple metrics need to be taken into account when evaluating model performance.

4.3 Approach A

4.3.1 Overview

The approach and results for each iteration of model design and hyperparameter change is be detailed in this section. In each iteration several variations were tried, and the results were taken from the most promising models. Discussion and explanation of said results is detailed in section 5.0. It's important to note that for all training and testing the data was shuffled and randomly selected from the training data. It's also important to note that the AUC scores were calculated using test data rather than training data. The reason for this was training accuracy and loss tended to be very high and low respectively. For this reason, it was decided that the test data would a more accurate prediction of classification performance at different thresholds.

It's also important to note that a constant batch size of 64 and learning rate of 0.001 was set. These values were found by trial and error. The learning rate of a model is different for each use case, depending on the gradient of the loss function. Too high and the minimum point may never be found. Too low the model would take too long to learn.

For text classification problems a starting dropout rate of 20%-50% has been cited as a good starting point [49]. Several research experiments have shown value in this range to perform well with text classification approaches [50], [51]. All iterations for each approach started with a dropout rate of 40%, and this value was adjusted from there.

4.3.2 Iteration 1.0

This iteration involved an MLP model with a relatively large number of hidden layers and units per layer in relation to this project. The following are the sample results from training and testing of several variations.

1.1

Hidden Layers = 6, Units/layer = 8, Dropout = 0.40, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.989722	0.034	0.790	0.79	0.812
2	0.9861	0.070	0.780	0.82	0.636
3	0.9891	0.041	0.796	0.82	0.635
4	0.9873	0.055	1.522	0.812	0.699
5	0.98577	0.063	0.725	0.826	0.479

Mean AUC	Mean Test Acc	Mean Test Loss
0.6522	0.81	0.922

1.2

Hidden Layers = 6, Units/Layer = 16, Dropout = 0.40, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.990	0.034	1.275	0.805	0.801
2	0.989	0.036	0.940	0.796	0.690
3	0.989	0.035	1.033	0.835	0.441
4	0.9890	0.0372	1.14	0.795	0.767
5	0.988	0.040	1.21	0.803	0.536

Mean AUC	Mean Test Acc	Mean Test Loss
0.64	0.80	1.12

1.3

Hidden Layers = 5, Units/Layer = 8, Dropout = 0.40, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.989	0.055	1.15	0.774	0.54891
2	0.98	0.045	0.92	0.785	0.4636
3	0.98	0.04	1.503	0.76	0.654
4	0.98	0.049	1.352	0.81	0.4660
5	0.98	0.049	0.94	0.79	0.751

Mean AUC	Mean Test Acc	Mean Test Loss
0.57	0.7778	1.172

Sample learning performance curves:

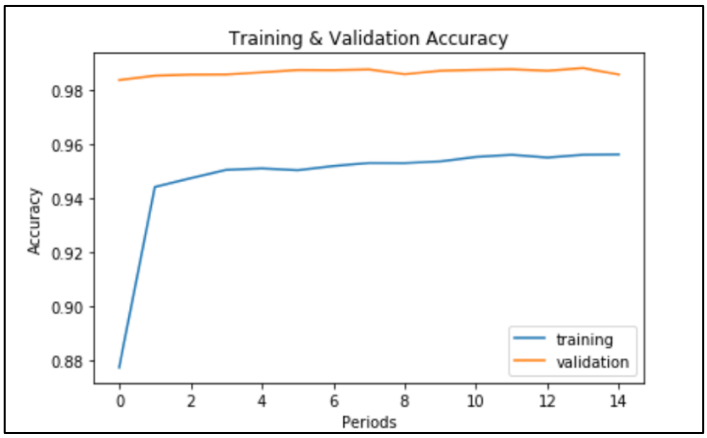


Figure 4.5

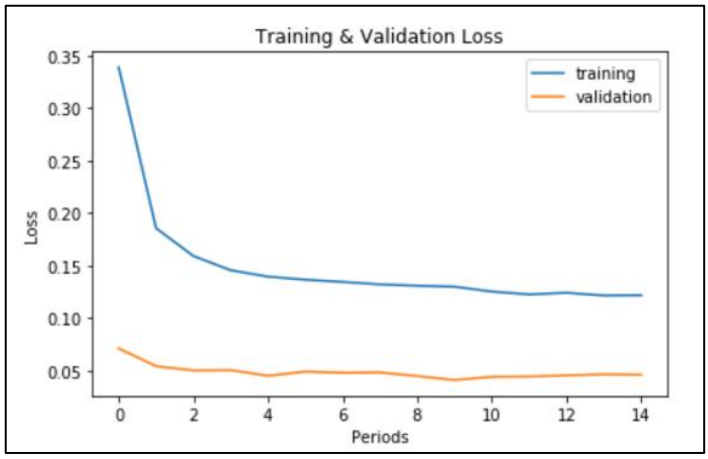


Figure 4.6

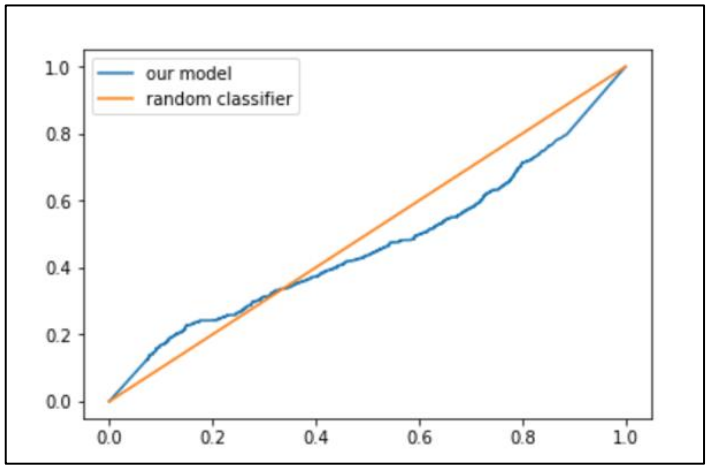


Figure 4.7

4.3.3 Iteration 2.0

2.1

Hidden Layers = 2, Units/Layer = 8, Dropout = 0.4, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.989388	0.034	0.6852	0.830	0.7894862
2	0.99055	0.0294	1.032	0.819	0.562368
3	0.99	0.0316	0.8206	0.80	0.69813825
4	0.99061	0.028	1.143	0.8054	0.560691
5	0.989	0.03	0.85	0.825	0.57902

Mean AUC	Mean Test Acc	Mean Test Loss
0.63	0.81	0.90

2.2

Hidden Layers = 2, Units/Layer = 8, Dropout = 0.25, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.9911	0.02	0.94	0.8024	0.868
2	0.990722	0.03	0.93	0.82	0.68

3	0.9892	0.028	0.7620	0.9905	0.9123
4	0.990388	0.02	0.69	0.831	0.59
5	0.989	0.03	0.39	0.835	0.626

Mean AUC	Mean Test Acc	Mean Test Loss
0.74	0.815	0.74

2.3

Hidden Layers = 2, Units/Layer = 16, Dropout = 0.4, Learning Rate= 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.991055	0.027	0.879	0.809	0.889323
2	0.989	0.0304	0.94	0.8160	0.787072372
3	0.99038	0.03059	0.872	0.8020	0.72313
4	0.9906	0.0260	0.879	0.822	0.634843
5	0.9905	0.0276	0.977	0.816	0.7479936

Mean AUC	Mean Test Acc	Mean Test Loss
0.75	0.81	0.9094

2.4

Hidden Layers = 2, Units/Layer = 16, Dropout = 0.25, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC

1	0.9910	0.026	0.955	0.817	0.886483
2	0.990	0.0282	0.890	0.8281	0.769487
3	0.98855	0.0340	0.8202	0.830	0.6089
4	0.991	0.02512	0.968	0.871	0.746
5	0.971	0.028	0.948	0.850	0.836

Mean AUC	Mean Test Acc	Mean Test Loss
0.76	0.831	0.902

Sample learning performance curves:

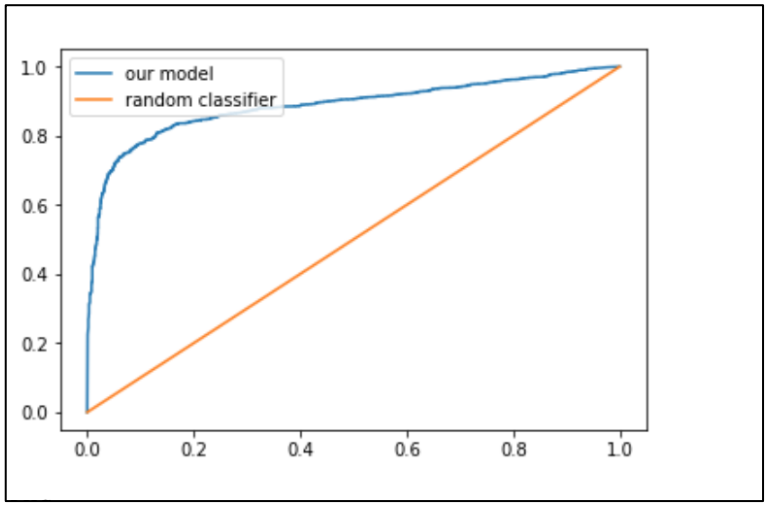


Figure 4.8

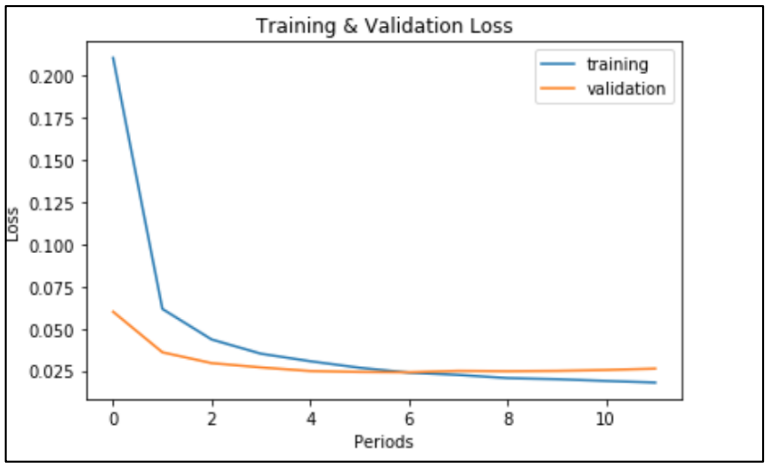


Figure 4.9

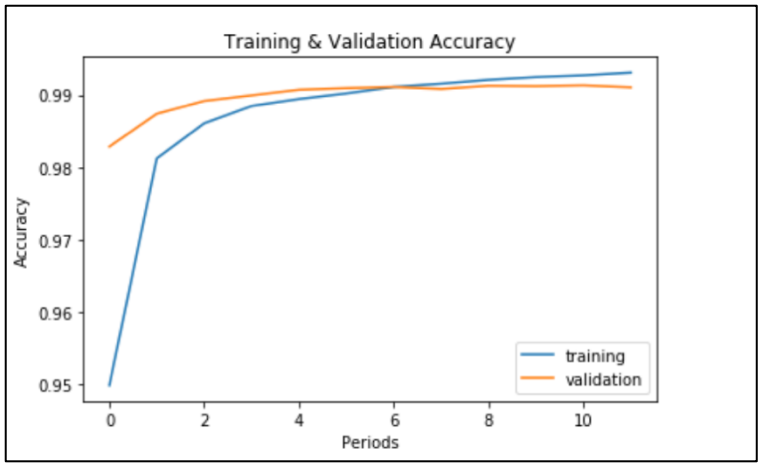


Figure 4.10

4.3.4 Iteration 3.0

3.1

Hidden Layers = 4, Units/Layer = 4, Dropout = 0.4, Learning Rate = 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.9893	0.1239	1.302	0.7933	0.72
2	0.9863	0.0999	1.33	0.8220	0.80
3	0.988	0.0600	0.463	0.815	0.82
4	0.987	0.052	0.449	0.822	0.73

5	0.9763	0.070	0.449	0.798	0.63

Mean AUC	Mean Test Acc	Mean Test Loss
0.74	0.8078	0.7898

3.2

Hidden Layers =4, Units/Layer=4, Dropout=0.25, Learning Rate= 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.9893	0.067	0.6919	0.8461	0.87613
2	0.9896	0.0647	0.715	0.8475	0.616897
3	0.9901	0.055	1.11	0.814	0.7141914
4	0.989	0.0815	0.538	0.82	0.8551531
5	0.9878	0.064	1.33	0.8062	0.75282

Mean AUC	Mean Test Acc	Mean Test Loss
0.76	0.82	0.87

3.3

Hidden Layers = 4, Units/Layer = 16, Dropout = 0.4, Learning Rate= 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.9884	0.03755	1.15	0.80	0.8879
2	0.9896	0.0358	1.30	0.81	0.57818

3	0.9902	0.033	1.08	0.822	0.7490
4	0.9898	0.035	1.236	0.7859	0.6359
5	0.9911	0.0317	1.303	0.778	0.5242

Mean AUC	Mean Test Acc	Mean Test Loss
0.675	0.79	1.213

3.4

Hidden Layers =4, Units/Layer=16, Dropout=0.25, Learning Rate= 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.99155	0.02893	1.2909	0.8186	0.7688
2	0.99083	0.03238	1.25	0.8240	0.4545
3	0.991	0.033	1.31	0.7837	0.6500
4	0.990	0.03636	1.251	0.8248	0.8788
5	0.9912	0.0321	1.278	0.8177	0.6381

Mean AUC	Mean Test Acc	Mean Test Loss
0.678	0.81376	1.28

3.5

Hidden Layers =4, Units/Layer=32, Dropout=0.25, Learning Rate= 0.001

Run	Validation Acc	Validation Loss	Test Loss	Test Acc	AUC
1	0.990	0.03547	1.3	0.80	0.66209
2	0.9908	0.0342948	1.266	0.812	0.6260
3	0.9917	0.03258	1.3310	0.8014	0.7224
4	0.991055	0.034	1.319	0.7993	0.6362
5	0.99144	0.030	1.2164	0.813	0.52402

Mean AUC	Mean Test Acc	Mean Test Loss
0.63	0.8026	1.26

Sample learning performance curves:

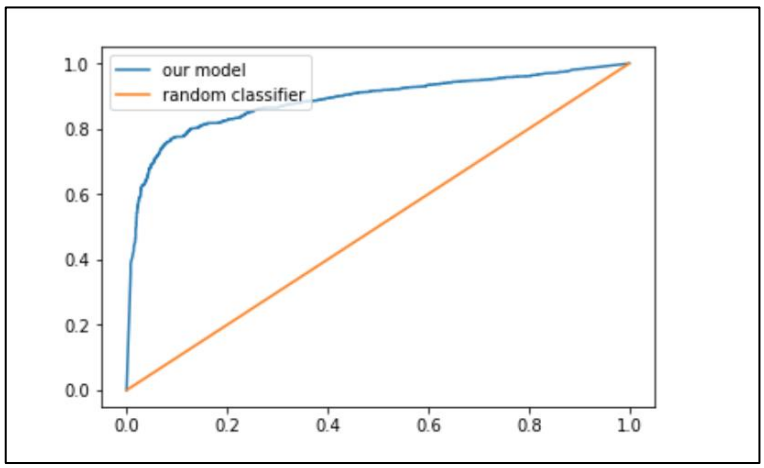


Figure 4.11

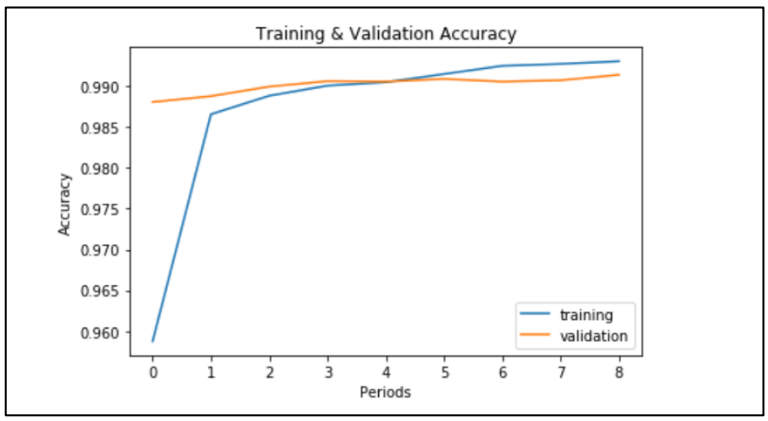


Figure 4.12

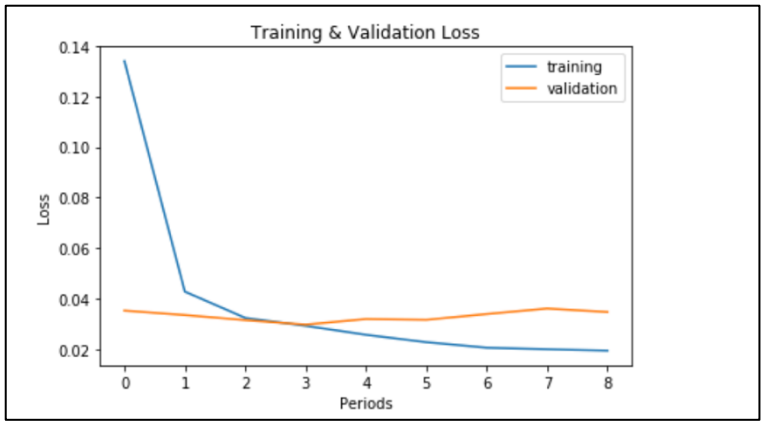


Figure 4.13

4.3.3 Top K

As mentioned in section 3.4.2 a feature importance function was applied to the feature set. The top K features were used in training as not all of the set will contribute to the output prediction. The initial value used was 20'000, which significantly reduced the size of the feature set to from over 140'000. Several values for K were experimented with, ranging up to 40'000 and down to 2500. The experiment found that increasing the value for k didn't have much effect on the accuracy of any of the models but reducing it below the 5000-threshold led to a lower performance.

4.3.4 N-Gram length

The N-gram length was limited to uni-grams (n=1), and bigrams (n=2) for the purpose of this research. The reason was due to low character limit on Tweets, the sentences and expressions contained in the data set was small. Therefore, having n-grams of a large length would lead to the model learning sentences as features rather than aspects of language used in a tweet. Experimental runs with n set to high values, such as three, four or five, led to extremely poor performance in testing.

4.4 Approach B

4.3.1 Overview

The experiments performed for *approach B* took the same format as approach A's experiments. Iterations were performed with different hyperparameters and the performance metrics were recorded. However, due to the metric scores retrieved early on in training this approach was not investigated to the extent *approach A* was. This approach provided much less promising results. It's important to note that the layers hyperparameter here represents a convolution block. For example, a "layers=2" would correspond to two blocks of separable convolutions. Its also important to note the training results were so poor there was no testing evaluations carried out on the models constructed approach B's iterations.

4.3.2 Iteration 1.0

Batch size=64, Layers=2, Kernel size=2, Dropout =0.2

Run	Validation Acc	Validation Loss	AUC
1	0.4990	0.693	0.5718053
2	0.5011	0.693	0.453859
3	0.4981	0.6931	0.518

Sample learning performance curves:

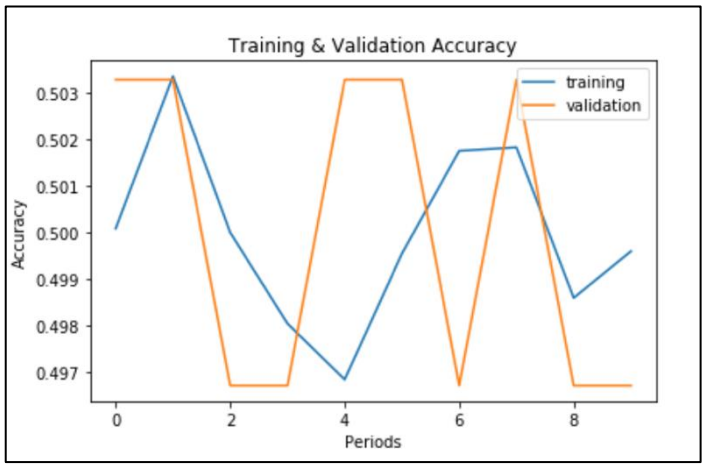


Figure 4.14

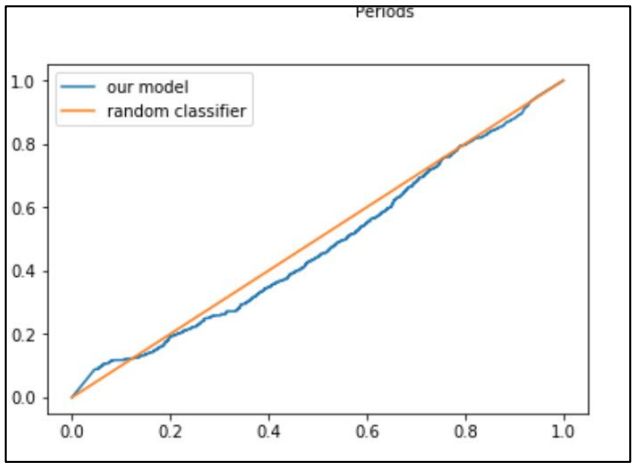


Figure 4.15

Figure 4.16

4.3.3 Iteration 2.0

Batch size=64, Layers=4, Kernel size=4, Dropout =0.2

Run	Validation Acc	Validation Loss	AUC
1	0.49672222	0.70099701	0.47
2	0.498777	0.693	0.59747
3	0.499833	0.693	0.567

Sample learning performance curves:

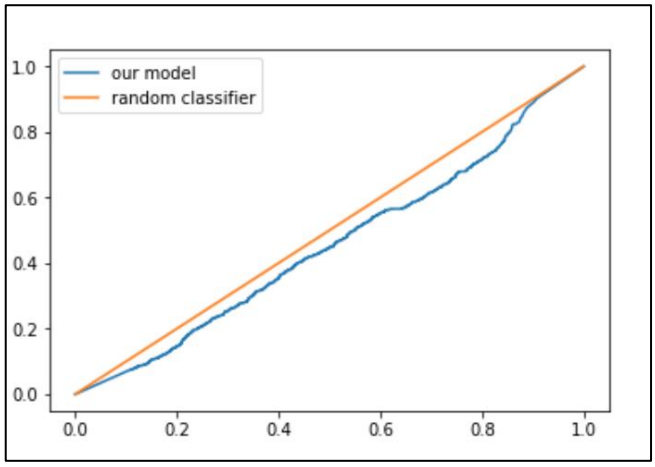
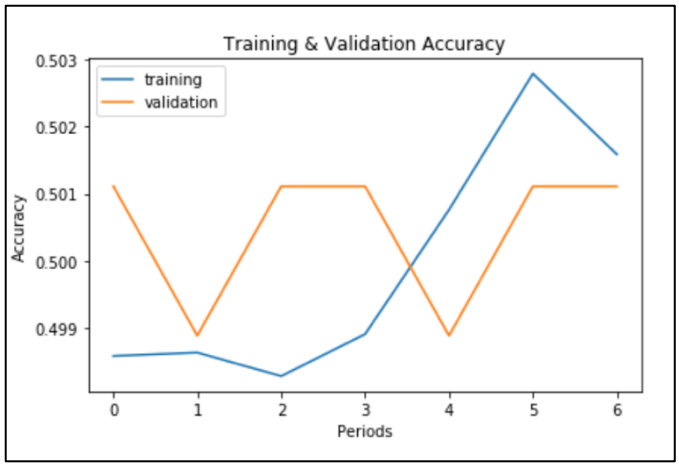


Figure 4.17

Figure 4.18


Training & Validation Accuracy
training
validation
0.503
0.502
0.501
0.500
0.499
Accuracy
0
1
2
3
4
5
6
Periods
Figure 4.19

5. Evaluation of Results

5.1 Overview

The following section provides evaluations of the results generated in section 4.0. This section details the causes for the varying results corresponding to the different model hyperparameters.

5.2 Approach A

5.2.1 Iteration 1.0

The first iteration of the MLP were deep models with many units in each layer. This research found that this model design performed the worst out of the other iterations on our sample set. The reasons for this are detailed as follows.

Overfitting

Overfitting was a problem in every iteration. The training and validation accuracy scores were very high, and they converged quite quickly. The model was also able to minimize loss to a low level after only two or three epochs on average. This indicated that the model was learning features specific to the training set, and this was confirmed when compared to the testing metrics.

From the sample runs variation 1.1, 1.2 and 1.3 all provided low AUC values, indicating a high number of false positives for low classification thresholds.
This indicates that the model is largely incapable of distinguishing between the two classes, despite all iterations having high test accuracy scores. The testing loss scores for all three variations were high. This research found this attributed to the high value of dropout regularisation used in training. All variations started with a dropout value of *0.40* to compensate for the model's ability to learn features specific to the training set. The

motivation behind this high dropout value was to penalize the model for learning training set specific features. Whilst this slightly increased the model's accuracy scores in testing, the model's capability to adapt to new unseen data still suffered. The trade off to having lower dropout values led to worse accuracy scores in testing.

This iteration found that the research projects training set features could be overfit too easily by complex MLP designs. From this, the decision was made to make the following iterations consist of less layers.

5.2.2 Iteration 2.0

Due to the nature of the results in iteration 1.0 the second approach consisted of less complex models. For starters all variations in iterations 2.0's design only had two hidden layers as oppose to the previous iterations design.

On top of high testing accuracy, all variations had higher AUC scores than the models in Iteration 1.0. High AUC scores indicated that all models would be able to maintain a high TPR with a low FPR at a higher classification threshold than iteration 1.0. Variations 2.1, 2.3 and 2.4 had higher loss scores than variation 2.2. The research attributed this to variations 2.1 and 2.3 having a higher dropout value than variation 2.2. Variation 2.4 had the same dropout value as variation 2.2, however it did not achieve as low loss values. As the complexity of NNs increases its ability to pick more specific features does. A larger number of units per hidden layer can lead to the model picking up training set specific features and overfitting data. This research attributes the high loss values in variation 2.4, despite a lower dropout rate, to the higher number of units per layer learning training set specific feature which are present in the test data.

The decision to lower the dropout rate in variation 2.2 was to compensate for 2.1's high loss scores. A NN's ability to adapt to unseen data is shown by its ability to minimize loss in testing. This is just as important as achieving high accuracy scores in testing when ensuring it can perform on unseen data.

Whilst lowering the dropout rate provided better loss values in testing, variations such as 2.3, were carried out to test if it were possible to still use dropout regularization. Although efforts to keep training and testing data completely separate this project acknowledges that features specific to the training set may also be present in testing, and

not in outside examples. Therefore, efforts to define a model with this in mind were made.

It's important to note that the mean of sample training runs does not tell the whole story of an iteration. Another considerable measure is the variance in values. The variance for all variations in iterations 2.0 is quite high. This coupled with high loss values in testing were the two main aspects aimed to be addressed in iteration 3.

5.2.3 Iteration 3.0

This was the final iteration of the MLP model design. The efforts made in this iteration were due to the findings in iteration 1.0 and iteration 2.0. The aim was to develop an iteration with more layers than iteration 2.0 but less than 1.0, in hope of providing a model capable of performing better than the two previous iterations. As observed in iteration 2.0, larger number of units in each hidden layer led to the model's inability to adjust to unseen data.

After testing several NN depth values it became apparent that a model with four hidden layers was providing promising results. Variation 4.1 was a simple design of four layers and four units. It provided as high AUC score as most of the variations in Iteration 2.0, whilst providing a low loss value in testing as well.

Although the accuracy scores achieved here matched the ones of previous iterations, the loss scores in testing and variation in all testing metrics was not fixed by the new design.

Iterations 3.3, 3.4, and 3.5 furthered the research's statement that the more units added to each hidden layer the more the model will overfit the training data.

5.2.4 Summary

The results from approach A indicate that less complex models were more capable at performing well on unseen data than ones which tended to have a high number of layers or units. The research attributes the performance of the MLP models to several reasons.

Overfitting

The reason overfitting occurred in training for each iteration can be attributed to the nature of the data used. The final data set used was over 140'000 entries, however this proved to be not big enough. Efforts were made in the data gathering stage to eliminate duplicates but its common practise on Twitter to "retweet" someone's original tweet. This involves essentially quoting this tweet to all of one's followers, but it is a built-in feature to Twitter and commonly used. Many situations arise where an individual will retweet someone else's tweet and add a small comment of their own, usually in relation to the tweet. Upon inspection of the data gathered this common practise allowed for essentially the same tweet in slightly different form to by-pass the projects duplicate tweet filtering. This in turn led to the MLP model seeing more or less the same tweet numerous times in testing, and thus leading to overfitting.
This research also acknowledges that the relatively small size of the training set compared to MLPs capabilities of learning non-linear features was quite small. Perhaps with a larger training set less overfitting would occur.

N-grams

The partial ordering maintained in representing the input features as n-grams also contributed to the model's ability to learn training set specific features, which did not appear outside of training. Although the size of *n* was kept at a relatively low value of one and two (n=1, n=2), the results imply this led to overfitting. In order to explain this, consider the following example.

Tweet Text: *"Transfer News: Liverpool to sign Karim Benzema, #Benzema #Liverpool"*

This is a transfer rumour from 2017, Liverpool did not sign Karim Benzema. Our system would label this tweet false accordingly and it would be added to the training data.

As an n-gram input feature, where *n=2,* we would have the following n-grams as input features.

N-grams: ["Transfer News:", "News: Liverpool", "Liverpool to", "to sign", "sign Karim", "Karim Benzema", "*Benzema, #Benzema*", "*#Benzema #Liverpool*"]

The n-gram in red font colour is an example of the issue in question. Its suspected the cause of overfitting the training data is the models tuning their weights to player to club

relationships, rather than the language in all of the tweet. A rumour about a high-profile player transferring to a high-profile club would be tweeted and retweeted a lot. When the model continuously saw player to club relationships in the text during each epoch it likely updated its weights accordingly to the player name and club, rather than to the type of language used.
Player second names themselves are quite unique and irregular to normal text, so the models also likely tuning their weights to them alone during training. This gives rise to potential future work which is detailed in section 7.

5.3 Approach B

5.3.1 Overview

The results in all *approach B's* iterations tell the same story. The sample learning performance curves summarise both iteration's struggles with approach B. The training accuracy curve showed that the accuracy levels in training all lay around the 0.5 mark. The loss curve also shows the model didn't at all converge to the training data. Immediately this was alarming as *approach A* showed stark signs of overfitting in training immediately, in all iterations of the model. The results for both iterations in *approach B* showed that this model architecture was essentially guessing the label for the training and validation examples. The sample ROC curve's shown in figures 4.15, 4.17 show that the model performance almost perfectly aligns with a random classifier, showing that this architecture cannot distinguish between its respective classes.

5.3.2 Summary

The results show that this approach was far too complex for the problem at hand.
The models were not able to learn any features of the data. The idea behind using word embedding for this research project was to attempt to make a connection between the language used and the label prediction. In other words, try and model the sensationalised language used in fake news tweets. This research found that the model could not make a connection between the semantics of the vocabulary used and the predicted label.

This research identifies the small vocabulary size as the reason for this. The vocabulary used was not big enough to provide embeddings worth representing in a dense vector space. Furthermore, when pretrained embedding were used, they still did not provide useful in classifying the examples. This implies that the approach itself was too complex for the problem issue.

6. Conclusions

6.0 Conclusion

This research builds on a significant body of research in to fake news detection and football transfer market prediction. The question this research project asked was:

> *"To what extent can supervised machine learning approaches be used to predict the accuracy of a tweet or Twitter account, in relation to a football transfer?"*

The research indicates that supervised machine learning approaches can be taken to predict the accuracy of a tweet or Twitter account in relation to a football transfer. The findings show that one approach, dubbed *approach A*, is capable of making accurate predictions on unseen data. Out of the two approaches taken the less complex and less computationally expensive approach provided more promising testing scores. It's important to note that variations of this method imply that its performance can be attributed to the data set used. Despite this, one variation of the method showed promising performances on unseen data.

Significantly the dataset collected has similarities with the BuzzFeed-Webis corpus [14], and the Pheme [20] corpus. The dataset collected during this research had entries which

had a mix of rumours and true transfers similar to how the Buzzfeed-Webis corpus contained entries which were a “fake” and “true” news mix. Also, the Pheme dataset had a third set of “unverified” entries, which didn’t fall into the “true” or “false” category.

The research also indicates that the dataset used in the approaches is relatively small in the context of the problem. To make a more definitive conclusion about the research question more experiments, with a more varied and larger data set, are needed.

Furthermore, the attempts made to construct the dataset used in the classification experiments provide useful insights. The main insight which can be taken from this is careful consideration should be taken in data gathering and labelling for future work as to not bias the data set.

It’s also significant to note that an approach aiming to take into account word ordering and semantics, dubbed *approach B*, performed much worse on the same data set as *approach A*. The results from this approach outline one approach which is certainly not suitable when trying to answer the research question.

The machine learning approaches for fake news classification of Buzzfeed-Webis corpus, mentioned in section 2.1.3, provided high accuracy but low recall values. The finding of the *approach A* build on this as high testing accuracy was accompanied with a high loss value and fluctuating AUC values.

Furthermore, the results from *approach A* contradict Xavier’s [54] claims that unigrams may not be a complex enough input feature. They do however build upon that a different approach to support vector machines is suited to the use case.

To conclude, the results imply that supervised machine learning approaches are capable of predicting the accuracy of tweet or twitter account in predicting a football transfer. The research acknowledges the results may have been affected by a bias in the data set, and the size of the data set. The research identifies another clear approach which is not capable of predicting a transfer rumours accuracy. Despite this further work is needed to make a definitive statement on the research question.

6.1 Future Work

The results of this experiment found some interesting insights into the data used and the approaches taken. These findings should be taken into consideration for future experiments in order avoid similar outcomes.

Entity freezing

The results showed a tendency to overfit the data in the training of the models. Even the most promising model architecture overfit the data in training. The research attribute this to the neural networks learning player names and player to club relationships. One potential improvement on this would be to use the same data set but with all the player names and clubs removed or replaced with a variable name. The following is an example of this.

Tweet text: *"Eden Hazard confirmed to be joining Real Madrid this summer, sources close to the player have confirmed !!!!! http://www. dailymail.co.uk/sport/football /article-604488"*

Entities removed: *"_ confirmed to be joining _ this summer, sources close to the player have confirmed !!!!! http://www. dailymail.co.uk/sport/football /article-604488"*

Notice how the player (Eden Hazard) and the club (Real Madrid) were replaced with underscores and the rest of the tweet was left. Instead of giving the model a chance to learn unique player to club connections to the predicted label, we could focus on training it to recognise features of the language used.

This way we could ensure the model would only try to predict based on words in the text around the players and club. Perhaps this might result in model being able to pick up on sensationalised fake news/tabloid style language in tweets.

References

[1] The Atlantic Emily Bruder https://www.theatlantic.com/video/index/577033/fake-news-fairytale/

[2] The Guardian https://www.theguardian.com/us-news/2016/dec/16/qa-russian-hackers-vladimir-putin-donald-trump-us-presidential-election

[3] New York Times Kevin Granville https://www.nytimes.com/2018/03/19/technology/facebook-cambridge-analytica-explained.html

[4] Reuters Foo Yun Chee https://uk.reuters.com/article/us-eu-tech-fakenews/facebook-google-to-tackle-spread-of-fake-news-advisors-want-more-idUKKCN1M61AG

[5] MIT Lincoln Laboratory Kylie Foy https://www.ll.mit.edu/news/using-machine-learning-detect-fake-news

[6] BBC Anisa Subedar https://www.bbc.com/news/blogs-trending-40574049

[7] Wired Zeynep Tufekci https://www.wired.com/story/free-speech-issue-tech-turmoil-new-censorship/?CNDID=50121752

[8] Wired Samantha Subramanian https://www.wired.com/2017/02/veles-macedonia-fake-news/

[9] BBC Ben Frampton https://www.bbc.com/news/uk-wales-34213693

[10] BBC https://www.bbc.com/news/technology-46590890

[11] The Guardian https://www.theguardian.com/commentisfree/2019/feb/28/facebook-twitter-fake-news-eu-elections

[12] llcott, Hunt, and Matthew Gentzkow. 2017. "Social Media and Fake News in the 2016 Election." *Journal of Economic Perspectives*, 31 (2): 211-36. (available online: https://www.aeaweb.org/articles?id=10.1257/jep.31.2.211)

[13] Mustafaraj, E. and Metaxas, P.T., 2017, June. The fake news spreading plague: was it preventable?. In Proceedings of the 2017 ACM on Web Science Conference (pp. 235-239). ACM. (available online: https://dl.acm.org/citation.cfm?id=3091523)

[14] Potthast, M., Kiesel, J., Reinartz, K., Bevendorff, J. and Stein, B., 2017. A stylometric inquiry into hyperpartisan and fake news. *arXiv preprint arXiv:1702.05638*. (available online: https://arxiv.org/abs/1702.05638)

[15] Granik, M. and Mesyura, V., 2017, May. Fake news detection using naive Bayes classifier. In *2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON)* (pp. 900-903). IEEE. (available online: https://ieeexplore.ieee.org/abstract/document/8100379)

[16] Pérez-Rosas, V., Kleinberg, B., Lefevre, A. and Mihalcea, R., 2017. Automatic detection of fake news. *arXiv preprint arXiv:1708.07104*. (available online: https://arxiv.org/pdf/1708.07104.pdf)

[17] Liu, X.F., Liu, Y.L., Lu, X.H., Wang, Q.X. and Wang, T.X., 2016. The anatomy of the global football player transfer network: Club functionalities versus network properties. *PloS one*, *11*(6), p.e0156504. (available online: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0156504)

[18] Frick, B., 2007. THE FOOTBALL PLAYERS'LABOR MARKET: EMPIRICAL EVIDENCE FROM THE MAJOR EUROPEAN LEAGUES. *Scottish Journal of Political Economy*, *54*(3), pp.422-446. (available online: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-9485.2007.00423.x)

[19] Football Whispers https://www.footballwhispers.com/

[20] Pheme dataset http://www.zubiaga.org/datasets/

[21] Beautiful Soup Documentation https://www.crummy.com/software/BeautifulSoup/bs4/doc/#making-the-soup

[22] MongoDB https://www.mongodb.com/

[23] Twitter API https://developer.twitter.com/en/docs/tweets/search/overview

[24] GetOldTweets https://github.com/Jefferson-Henrique/GetOldTweets-python

[25] English clubs confirmed transfers https://en.wikipedia.org/wiki/List_of_English_football_transfers_summer_2018

[26] Pandas https://pandas.pydata.org/

[27] English Clubs https://en.wikipedia.org/wiki/List_of_football_clubs_in_England

[28] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. and McClosky, D., 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* (pp. 55-60). (available online: https://www.aclweb.org/anthology/P14-5010)

[29] NLTK https://www.nltk.org/

[30] Sachin Malhotra Part-of-speech tagging https://medium.freecodecamp.org/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24

[31] Spacy https://spacy.io/

[32] Onto Note release 5.0 https://catalog.ldc.upenn.edu/LDC2013T19

[33] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T., 2016. Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759. (available online: https://arxiv.org/abs/1607.01759)

[34] Scikit Learn https://scikit-learn.org/stable/

[35] Anova F-test https://blog.minitab.com/blog/adventures-in-statistics-2/understanding-analysis-of-variance-anova-and-the-f-test

[36] Preparing data https://developers.google.com/machine-learning/guides/text-classification/step-3

[37] Keras Pre-processing https://keras.io/preprocessing/text/

[38] Lai, S., Xu, L., Liu, K. and Zhao, J., 2015, February. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*. (available online: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewPaper/9745)

[39] Bourlard, H. and Wellekens, C.J., 1989. Links between Markov models and multilayer perceptrons. In *Advances in neural information processing systems* (pp. 502-510). (available online: http://papers.nips.cc/paper/163-links-between-markov-models-and-multilayer-perceptrons.pdf)

[40] Guide to Convolutional Neural Networks Sumit Saha https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[41] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T., 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*. (available online: https://arxiv.org/abs/1607.01759)

[42] Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258). (available online: https://arxiv.org/pdf/1610.02357.pdf)

[43] Jupyter https://jupyter.org/

[44] Xu, B., Wang, N., Chen, T. and Li, M., 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*. (available online: https://arxiv.org/pdf/1505.00853.pdf)

[45] Training Neural Neworks https://developers.google.com/machine-learning/crash-course/training-neural-networks/best-practices

[46] Classification Accuracy https://developers.google.com/machine-learning/crash-course/classification/accuracy

[47] Loss Functions https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropys

[48] Understanding AUC - ROC Curve Sarang Narkhede https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[49] Dropout Regularization in Deep Learning Models With Keras Jason Brownlee https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[50] Wager, S., Wang, S. and Liang, P.S., 2013. Dropout training as adaptive regularization. In *Advances in neural information processing systems* (pp. 351-359). (available online: http://papers.nips.cc/paper/4882-dropout-training-as-adaptive-regulariza)

[51] Zhou, C., Sun, C., Liu, Z. and Lau, F., 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1511.08630* (available online: https://arxiv.org/pdf/1511.08630.pdf)

[52] Caled, D. and Silva, M.J., 2018. FTR-18: Collecting rumours on football transfer news. *arXiv preprint arXiv:1812.00778*. (available online: https://arxiv.org/pdf/1812.00778.pdf)

[53] Ireson, N. and Ciravegna, F., 2017, October. Football Whispers: Transfer rumour detection. In *CEUR Workshop Proceedings* (Vol. 1963). CEUR Workshop Proceedings. (available online: http://eprints.whiterose.ac.uk/124566/)

[54] Xavier, J., School of Computer Science and Statistics O'Reilly Institute, Trinity College, Dublin 2, Ireland. (available online: https://scss.tcd.ie/publications/theses/diss/2018/TCD-SCSS-DISSERTATION-2018-022.pdf)

[55] Depthwise separable convolutions for machine learning https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/

[56] Classification: ROC Curve and AUC https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

Table of Abbreviations

Abbreviation	Term
NER	Named Entity Recognition
SVM	Support Vector Machine
CNN	Convolutional Neural Network
MLP	Multilayer Perceptron
sepCNN	Separated Convolutional Neural Network
NN	Neural Network
DB	Database
ML	Machine Learning
AUC	Area Under Curve