

**Joseph Marchant**

**Predicting the outcomes of English  
Premier League matches**

Computer Science Tripos – Part II

Robinson College

January 2, 2020



# Proforma

Candidate Number: **2394A**  
Project Title: **Predicting the outcomes of English Premier League matches**  
Examination: **Computer Science Tripos – Part II, May 2019**  
Word Count: **11997<sup>1</sup>**  
Lines of Code: **Around 50 per model + 1000s in Data Curation**  
Project Originator: **2394A**  
Supervisor: **Helena Andres-Terre**

## Original Aims of the Project

To write a dissertation on an investigation into predicting the outcomes of English Premier League matches. This includes: finding data sources, cleaning and preparing data and building models to predict Home Win, Away Win or Draw (H/A/D) at a 40% accuracy or better. Succeeding that, I ambitiously aim to rival expert pundits, hitting a 52% accuracy. I will also analyse the results and investigate the problem in depth to see why it is so notoriously difficult. The project also requires determining a software process to use, in order to iteratively improve the best-so-far model and dataset.

## Work Completed

I collected data from multiple data sources, cleaned each, merged and validated them against one another. I wrote code to achieve this and generate extra features. Analysed problem in depth before working on predictors. Designed and implemented a modified version of the Spiral Software Process with each Spiral containing: dataset adjustments; multiple models and evaluations for each; analysing results and planning next steps. Researched examples of similar work, using those to aid design decisions. Final models achieved 53% accuracy's for the H/A/D prediction which exceeds the 52% achieved by professional human pundits. Wrote this dissertation.

## Special Difficulties

None

---

<sup>1</sup>This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

## Declaration

I, Joseph Marchant of Robinson College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Motivation . . . . .	9
1.2	Relevance . . . . .	9
<b>2</b>	<b>Preparation</b>	<b>11</b>
2.1	Initial Research . . . . .	11
2.2	Predicting the Outcome of Football Matches . . . . .	11
2.2.1	Challenges of Predicting the Outcome of Football Matches . . . . .	11
2.2.2	Previous Work . . . . .	12
2.3	Football . . . . .	12
2.3.1	Potential Features . . . . .	13
2.4	Web Scraping . . . . .	13
2.4.1	11v11 . . . . .	13
2.4.2	Robots.txt files . . . . .	13
2.5	Background Theory . . . . .	14
2.5.1	Supervised Learning . . . . .	14
2.5.2	Neural Networks . . . . .	14
2.5.3	Deep Learning . . . . .	15
2.5.4	Hyperparameters . . . . .	15
2.5.5	Software Packages and Tools . . . . .	16
2.5.6	Data Science . . . . .	17
2.6	Starting Point . . . . .	18
2.7	The Spiral Model . . . . .	18
2.7.1	Why the Spiral Model? . . . . .	18
<b>3</b>	<b>Implementation</b>	<b>19</b>
3.1	Data Preparation . . . . .	19
3.1.1	Data Sources . . . . .	19
3.1.2	Data Cleaning . . . . .	20
3.1.3	Data Generation . . . . .	21
3.1.4	Finalising Dataset for Models . . . . .	22
3.2	Analysis of the Problem . . . . .	22
3.2.1	Unbalanced Data . . . . .	23
3.2.2	Current League Position . . . . .	23
3.2.3	League Table Statistics . . . . .	24
3.2.4	Dimensionality Reduction . . . . .	25
3.3	First Spiral - Experimentation . . . . .	26

3.3.1	Initial Dataset . . . . .	26
3.3.2	Models . . . . .	27
3.3.3	Best Model So Far . . . . .	29
3.3.4	Conclusion and Next Steps . . . . .	29
3.4	Second Spiral - Adding Features . . . . .	29
3.4.1	Adding Features . . . . .	30
3.4.2	Changes to Data - Different Range . . . . .	31
3.4.3	Models and Evaluation . . . . .	31
3.4.4	Conclusion and Next Steps . . . . .	32
3.5	Third Spiral - Feature Importance . . . . .	32
3.5.1	Permutation Importance . . . . .	32
3.5.2	Models . . . . .	33
3.5.3	Conclusions . . . . .	34
3.6	Final Model . . . . .	34
3.6.1	Model Chosen . . . . .	35
3.7	Extensions . . . . .	35
<b>4</b>	<b>Evaluation</b>	<b>37</b>
4.1	Model Evaluations in Spiral Process . . . . .	37
4.1.1	First Spiral Evaluation . . . . .	37
4.1.2	Second Spiral Evaluation . . . . .	38
4.1.3	Third Spiral Evaluation . . . . .	39
4.1.4	Training Speeds . . . . .	41
4.2	Final Model Evaluation . . . . .	42
4.3	Project Evaluation . . . . .	43
4.3.1	What went well . . . . .	43
4.3.2	What I'd do differently . . . . .	43
4.3.3	Ethical Considerations . . . . .	43
4.3.4	What next? . . . . .	44
<b>5</b>	<b>Conclusion</b>	<b>45</b>
	<b>Bibliography</b>	<b>45</b>
<b>A</b>	<b>Project Proposal</b>	<b>49</b>

# List of Figures

2.1	Header of 11v11 Robots.txt file . . . . .	14
2.2	Example fully-connected NN with a 3-Neuron Input Layer, a single Hidden Layer with 5 Neurons, and an Output Layer with 2 Neurons . . . . .	15
2.3	Example of using the interactive environment in Jupyter Notebooks . . . . .	16
3.1	Header of Initial Dataset . . . . .	22
3.2	Pos-diff Distribution across all Games . . . . .	23
3.3	Pos-diff Distribution across all Games, split by result . . . . .	24
3.4	Seaborn Pairplot of League Table Statistics in Initial Dataset . . . . .	24
3.5	Dimensionality Reduction Plots . . . . .	26
3.6	Code Sample and Output showing class prediction spreads for best NN model so far . . . . .	29
3.7	Feature Importance's on Dataset . . . . .	32
3.8	Plot to find optimal value of K Neighbours . . . . .	34
3.9	Code Sample and Output for KNN Classifier where K=109 . . . . .	34
3.10	Header of the Final Dataset . . . . .	35
4.1	Code Sample and Output showing that the KNN Classifier correctly predicted 180 Away Wins, 932 Home Wins but only 6 Draws . . . . .	38
4.2	Code Sample and Output showing the counts of each class in the predictions of the test set for the KNN Classifier . . . . .	38
4.3	Code Sample and Output showing the <b>new</b> counts of each class in the predictions of the test set for the Neural Network (V3) after considering class weights . . . . .	38
4.4	Code Sample and Output for spread of classes in validation set predictions with best NN model from the first spiral . . . . .	39
4.5	Code Sample and Output for spread of classes in validation set predictions with best NN model from the second spiral . . . . .	39
4.6	Feature Importance's on New Dataset where aPosLS, hPosLS, hGD5 and aSpent were removed . . . . .	40
4.7	Code Sample and Output demonstrating the Prediction Accuracy and Time Taken for the SVM with an RBF Kernel . . . . .	41
4.8	Code Samples and Outputs demonstrating the Training of a NN, the time taken to train, as well as the validation set loss and accuracy . . . . .	42
4.9	Code Sample and Output showing class prediction spreads for final adaptive model . . . . .	42

## Acknowledgements

Thanks to both Prof Alan Mycroft and Helena Andres-Terre for their continued support and advice throughout this project.



# Chapter 1

## Introduction

### 1.1 Motivation

As a life-long football fan, predicting the outcome of matches has always been a fun aspect of the whole experience. Beyond just personal satisfaction, football is the largest sport in the world in terms of the betting industry, and it's fairly obvious as to why: There are teams all across the world, and games going on 24/7. Within football, the English Premier League receives the most bets and therefore there's a lot of interest in modelling it. Predicting matches of the Premier League may also have financial benefits which is something I'd like to explore in more detail if time permits.

I also decided to try do so using only information available before the match begins, making the prediction systems usable for real-time predictions. I aim to be able to predict at the accuracy of professional pundits (52%), however my intuition, knowledge and research imply this is a very ambitious goal, and so my formal success criteria is to reach a 40% prediction accuracy.

### 1.2 Relevance

Predicting football results is also known to be notoriously difficult. This makes it perfect for the field of Computer Science, and more specifically, Machine Learning. This project requires many skills from Data Science, from collecting and curating data to building models (such as Neural Networks) to work with that data.

Experts have been hired by sports news services to predict the outcomes of matches themselves, given decades of experience following the game. Despite this, they struggle to exceed a 52%<sup>1</sup> prediction accuracy. Along with humans struggling to predict at a high accuracy, software models struggle to beat that too. This is detailed more in section **2.2.2 Previous Work**.

---

<sup>1</sup><http://eightyfivepoints.blogspot.com/2017/07/how-are-lawrenson-and-merson-beating.html>



# Chapter 2

## Preparation

### 2.1 Initial Research

As this has been an interesting idea and topic for me for years, I already had the background knowledge around football that allowed me to focus more of my time on the Data Science. My initial research consisted of searching for similar projects, and a requirements analysis. This outlines the sort of skills and techniques I needed to learn and use to increase the likelihood of a successful project. This is all detailed further in the following sections.

### 2.2 Predicting the Outcome of Football Matches

#### 2.2.1 Challenges of Predicting the Outcome of Football Matches

Football is notoriously difficult to predict the outcome of, for various reasons. Firstly, it's the complexity of the game. Each game consists of hundreds of decisions per player and manager, decisions that would be impossible to model individually. This makes the problem difficult to challenge with simulation, and so you must predict the outcome 'blindly'. Any one of these decisions can change the whole outcome. A player can slip, a goalkeeper could drop a routine save or a player could get sent off. All of this combined makes Football seem random at time, with few patterns to rely on - which is exactly part of the appeal of the game and betting on the result. Upsets happen regularly and unpredictably, which is the beauty of Football.

It's particularly difficult for humans to predict because we support teams, dislike others, and have our own perspectives on teams that aren't necessarily true. Even the expert pundits with decades of experience and knowledge surrounding the sport can only predict just over half the games correctly.

It's also hard for computers to predict because most factors impacting a football match cannot be modelled nor passed into a classifier. For example, how do you model and represent a teams confidence? What about their mood? Does the referee have a bias? Computers can't even begin to consider these sort of questions.

### 2.2.2 Previous Work

Most work on this topic is very new, as good data and effective algorithms are fairly new too. Here I'll detail some previous work with varying relevancy to my project, but all on the topic of predicting the outcome of football matches.

Using in-game statistics (such as shots and cards), many people such as github user RudrakshTuwani (who was able to predict at a 69% accuracy [6]), worked on the football-data.com<sup>1</sup> dataset to predict the outcomes of matches. The problem attempted here is a much simpler problem than mine for multiple reasons. Firstly, he only predicted Home Win? - making it a two-way classification problem, as opposed to my three-way classification problem. Additionally, using in-game statistics makes the problem far simpler, as you are using information you wouldn't have had before the game began to aid your predictions.

Similarly, Daniel Pettersson and Robert Nyquist built a Recurrent Neural Network to predict the outcome of the match at different stages throughout the match, demonstrating high accuracy's near the end[5]. At the beginning of matches when they only had teams and lineups, their best models reached a 45.90% accuracy. This accuracy reached high 90's once the model has information about the game, i.e. the match had begun and minutes had passed. They concluded that "This is no surprise since more information should lead to better predictions. However, the increased prediction accuracy over minutes played in a match indicates that the network is able to learn about football.", which is a promising sign.

In 2006, Anito Joseph, Norman E Fenton, and Martin Neil[1] looked at predicting a similar problem to mine - win, draw or loss - but for a specific team. They tried multiple models, including variants of Bayesian Nets, a decision tree and a K-Nearest Neighbours implementation. It also only used data available from before the match began, making it much more similar to my project. They specifically designed their own features for their "Expert Bayesian Net". They also looked at training models on short time periods separately, potentially allowing for more accurate overall predictions. Their "Expert BN" was able to reach accuracy's of around 59%, implying that approaching this problem from the perspective of one team at a time could be an effective approach. However, I don't think all teams would behave the same.

More recently, in 2011 Josip Hucaljuk and Alen Rakipovi looked at a single season of Champions League games[4] with similar features to what I plan to use, achieving higher accuracy's however more unreliable results due to the lack of data used. Additionally, in 2007 Burak Galip Aslan and Mustafa Murat Inceoglu managed to achieve a 53% accuracy[2] on this problem by using Linear Vector Quantisation methods, retraining the model after each week of results. Reaching an accuracy around this would be an incredibly successful project, given the outcomes of all above relevant work.

## 2.3 Football

Football is a simple game consisting of 22 players (11 on each team), and a referee. The goal is to score more than the other team. Each match has 3 outcomes: Home Team Win, Draw, Away Team Win. It's well known that there exists a "home-team advantage",

---

<sup>1</sup>football-data.com/englandm.php

i.e. the Home Team wins more often than any other outcome (detailed further in section **3.3.1 Unbalanced Data**).

In the Premier League, every team plays each other team both at Home and Away during the season. This leads to each team playing 38 games a season, as the League consists of 20 teams. Therefore each season of data has 380 games.

It's also important to note that games are played in order, and hence I must keep the dataset in order. For example, I may want to use a feature such as "Form", representing how well a team has done in the last X matches.

### 2.3.1 Potential Features

Combining research with my own experience with Football, I was able to build a long list of potential features: Date, Home Team, Away Team, Result (target variable), Referee, Manager, Pitch Size, Form data (last 3 game win%), League Stats in current season, Injury Data, Last Season League Position, Transfer Window Spending, Player Data and more yet to be decided upon. As the project progressed, I continued to work with and researched for more features, consistent with the Spiral Model described later in this chapter.

I also decided that the most effective first dataset to use for modelling and predictions is just using League Stats in the current season. This includes features such as: "Goals Per Game", "Goals Conceded Per Game" and "Points Per Game". I chose this subset because it accurately and simply represents how well the team is performing so far this season, and all columns are numerical which is far easier to work with.

## 2.4 Web Scraping

Since I need various statistics for **all** games played, I knew I'd have to consult multiple data sources in order to validate the data.

### 2.4.1 11v11

Whilst searching, I found 11v11<sup>2</sup> provided the best range of both games and features per game. After looking around the site, I worked out that all matches could be found by starting from a link in this format: **<https://www.11v11.com/competitions/premier-league/1993/matches/>**. With this link for each season, I could also acquire all individual game links for that seasons games. Each match page holds various bits of information - we'll need to acquire this data for every match, which we obviously cannot do manually since there are over 10000 matches - so we must scrape the data. In order to scrape data off a website, you must first refer to its robots.txt file.

### 2.4.2 Robots.txt files

To check the allowed access for scraping, you go to **name.com/Robots.txt** to view the Robots.txt file. This details what pages you're allowed to scrape off of. Usually, sites allow

---

<sup>2</sup><https://www.11v11.com/>

---

```

User-agent: *
# stop google treating fb relative links as ours
Disallow: /plugins/feedback.php
# tags not available - should they be? remove from sitemap if need be
Disallow: /tags/
Disallow: /404.php
# old cms stuff
Disallow: /subscribers/index.php?pageID=286
# vbulletin stuff there is no need to index
Disallow: /forums/admincp/
Disallow: /forums/clientscript/
Disallow: /forums/cpstyles/

```

Figure 2.1: Header of 11v11 Robots.txt file

you to scrape data from all pages of their website except a few, and just give you some access limits so you don't overwhelm their servers. The first few lines of the Robots.txt file for 11v11.com are shown in Figure 2.1.

Robots.txt files are strictly for disallowing certain scraping, therefore if a url isn't mentioned you are free to use it. For example, in Figure 2.1 you can see that the link **11v11.com/forums/cpstyles** has been tagged as *Disallow*, meaning we cannot scrape from that page. In the file, there is no mention of the **/matches** links and therefore we are free to scrape data from them.

## 2.5 Background Theory

Here I'll detail relevant background theory that I had to learn or confirm knowledge on. Everything in this dissertation should be accessible with this theory in mind.

### 2.5.1 Supervised Learning

Supervised Learning is the process of training a model to predict a target variable, given a set of input features, where the model is **given the real values**, in order to *learn* from its predictions. On the flip side, Unsupervised Learning is where we don't give the models the real output values, and instead looks for patterns or clusters. This project uses Supervised Learning, as we know the outcomes of the matches and aim to investigate how often we can accurately predict them.

### 2.5.2 Neural Networks

A typical brain consists of around 100 billion cells called **neurons**, each with a number of connections coming from it. Neural Networks are designed to simulate this vast collection of connected neurons, allowing them to learn things and recognise patterns in a more *human-like* way. On top of this, you don't have to program Neural Networks, they learn by themselves. Note, these are technically called **Artificial Neural Networks (ANNs)**.

A Neural Network (NN) consists of a number of **layers**, each consisting of a number of **neurons**. Each NN has three types of layers: the **input layer**, which is the input dataset; the **hidden layer(s)**, which transform the data, and the **output layer**, which has the same number of neurons as the number of classes of the target variable. An example is shown in Figure 2.2.

Most NNs are **fully connected**, i.e. all nodes of adjacent layers are connected. These edges hold **weights**, representing the importance of that neuron for use in the next layer. The network learns by **forwarding** the information through the layers. Then, via the feedback process called **back-propagation**, it compares the output (predictions) with the expected output (real values), using some sort of loss or difference function. It then uses these differences to update the edge weights in the network, and repeats this process until the training step stops. Therefore the NN converges on the weights that decrease the difference between the predictions and the real values, thus *learning* the prediction weights. This manifests itself as a network that now has been trained on training data, and can be presented with *completely unseen data* and told to predict the output class for these examples.

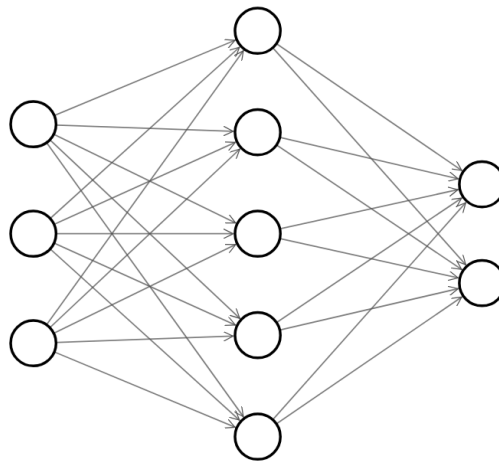


Figure 2.2: Example fully-connected NN with a 3-Neuron Input Layer, a single Hidden Layer with 5 Neurons, and an Output Layer with 2 Neurons

### 2.5.3 Deep Learning

A single hidden layer in a Neural Network is enough to solve most simple problems effectively. However some problems are more complex (such as the problem tackled in this paper - more information in section **3.3.4 Dimensionality Reduction**). These problems require a more flexible model to pick up the extra complexities. This is where **Deep Learning** comes in - it is simply a reference to the depth of the NN, i.e. if we have more than one hidden layer. The models in this project were all **Feed-Forward (FF)** or **Deep Feed-Forward (DFF)** models, depending on the number of hidden layers.

### 2.5.4 Hyperparameters

A hyperparameter is simply a parameter that is set *before the training process begins*. Different models and problems require different hyperparameters, but for example, a K-Nearest Neighbours Classifier requires you to specify the K used in the training process for the number of neighbours to use. Alternatively, the Hyperparameters of a Neural Network include the number of neurons, the number of epochs, the use of Dropout etc.

## 2.5.5 Software Packages and Tools

### Tools

All code for this project has been written in Python, using an IDE called Jupyter Notebooks<sup>3</sup>. This creates an interactive environment, allowing me to view states at run time as shown in Figure 2.3. Here I was able to see how often two columns agreed on the value, after merging two datasets with common columns. It also allows me to add Markdown. It's perfect for data science and all forms of exploratory analysis.

```
In [8]: def cols_agree(row,col1,col2):
        return (str(row[col1]) == str(row[col2]))
df['home_agree'] = df.apply(cols_agree,args=('home','home2'),axis=1)
df['home_agree'].value_counts()

Out[8]: True      6840
        Name: home_agree, dtype: int64
```

Figure 2.3: Example of using the interactive environment in Jupyter Notebooks

### Packages

The code for this project has all been written in Python, using the following packages:

- **Pandas**<sup>4</sup> - Primarily used for converting CSV files into Pandas DataFrames, which are easy and fast to work with, allowing me to manipulate my data in a convenient way. Good prior experience.
- **Keras/Tensorflow**<sup>5</sup> - Keras is a Deep Learning Python Library for building Neural Networks, and it was implemented within **Tensorflow**. Tensorflow is an alternative that was introduced to me during the Part II Data Science Course, however through research and recommendation from my supervisor, I learned that Keras allowed a lot more flexibility. No prior experience, test models produced before project usage.
- **Scikit-learn**<sup>6</sup> - Gives lots of useful functions to use in ML and Data Science applications, such as dimensionality reduction and classification. Very limited prior experience.
- **requests**<sup>7</sup> - Used to send HTTP requests to websites, allowing me to web scrape. No prior experience.
- **BeautifulSoup**<sup>8</sup> - A package for pulling data out of HTML (and XML) files. This allowed me to parse and read the responses from websites collected from requests, which I could then crawl through and collect data from. No prior experience.

<sup>3</sup><https://jupyter.org/>

<sup>4</sup><https://pandas.pydata.org/>

<sup>5</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras](https://www.tensorflow.org/api_docs/python/tf/keras)

<sup>6</sup><https://scikit-learn.org/stable/documentation.html>

<sup>7</sup><https://2.python-requests.org/en/master/>

<sup>8</sup><https://www.crummy.com/software/BeautifulSoup/bs4/doc/>



- **matplotlib**<sup>9</sup> - The industry standard visualisation package. Works interactively with Jupyter Notebooks. Some prior experience.
- **seaborn**<sup>10</sup> - Another Python Visualisation Library, build on top of matplotlib. Creates cleaner, more feature-full and complex graphs in far fewer lines of codes. Limited prior experience.
- **Numpy/Scipy**<sup>11</sup> - Scientific Computing Packages, used to work with data in more efficient ways. Some prior experience.

## 2.5.6 Data Science

### Train-Dev-Test Split

This refers to how we separate the dataset for usage in training (training set), continuous testing through development (development set), and for final predictions on never-before-seen data (test set). Usually, about 5-20% of the data is separated to be the test set, and the rest is split in a train:dev ratio ranging from 60:40 to 80:20. Therefore most goes towards the training set. This is because more data is better for models to learn from - the more examples a model has for reference, the more likely the model is to converge.

There are multiple methods for splitting the dataset. For example, we could simply randomly split the data. Alternatively, you could use a method called **StratifiedShuffleSplit**. This splits the data by trying to put similar proportions of different values in each split, ensuring that each split is as similar as possible in representing the original larger set. This would usually be the go-to choice, however my problem requires a different solution.

In my project, I used an Ordered Split. Here, I simply set the last couple seasons apart for the test set, the next few newest seasons for the development set, and the rest for the training set. This is mainly due to the fact that all predictions need to be based only on information collected from the past. Additionally, each season has very similar spreads of feature and output ranges, so this technique naturally gives us the benefits of a **StratifiedShuffleSplit**.

### Why Data Curation is so important

A very large portion of this project falls under the vague term **Data Curation**. This includes determining sources of data, collecting that data, cleaning the data, adding other data, merging datasets, transforming data, splitting the data and using the data in the model. Each of these phases are vitally important to the success of any data science project.

Determining sources of data required a lot of research, finding mostly unusable data sources. As outlined in section **2.4.2 Robots.txt files**, many sites and sources don't actually allow users to collect their data.

Cleaning the data was a necessary but tedious task as models such as Neural Networks cannot work with missing values. I was able to fill all missing values with their correct

---

<sup>9</sup><https://matplotlib.org/>

<sup>10</sup><https://seaborn.pydata.org/>

<sup>11</sup><https://docs.scipy.org/doc/>

values, detailed in section **3.1.2 Data Cleaning**. This also required merging datasets, which introduces an extra level of complexity.

## 2.6 Starting Point

At the initial research phase, I only had knowledge of predicting football matches as a human, having no real concept of how to approach it as a Computer Scientist. After my research and requirements analysis, I found I had no experience with the following vital packages: Requests, BeautifulSoup, Keras and Seaborn. Alongside this, I had a little experience with Tensorflow, Matplotlib, Scikit-learn, Pandas and Numpy, feeling most comfortable working with Pandas and Numpy. This allowed me to quickly get started, but my lack of prior experience with other packages caused delays elsewhere.

Most package choices were done after research and confirming choices with my supervisor. Data Science in Python tends to be done in various ways, but there's almost always a default option that most professionals use in certain scenarios, which is always the choice I made sure to follow.

## 2.7 The Spiral Model

The Spiral Model is a software development process model, first described by Barry Boehm in 1986 in his paper "A Spiral Model of Software Development and Enhancement" [3].

The bulk of my project is following a Spiral Model, where each loop allows the following process to occur: Modify working dataset; experiment with model prototypes; narrow down to best performing models; analyse and evaluate in more depth; decide next steps and then go through to the next loop.

### 2.7.1 Why the Spiral Model?

Using a Spiral Model allows both the working dataset and the best-so-far model to be incrementally improved, allowing lots of freedom to learn and experiment with the problem in detail, whilst moving towards the goal of a better prediction accuracy. I also have experience working with this Software process, as it is the recommended way to approach Data Science project and it's how I approached my project from the Part II Data Science course. Also, speaking to my supervisor and researching it, it was quickly clear that this model was the one to go with.

# Chapter 3

## Implementation

### 3.1 Data Preparation

In this section I detail how I went from knowing about data sources from the Preparation phase, all the way up to having a dataset ready for working with.

#### 3.1.1 Data Sources

After my initial research and preparation, I knew which sources I was going to use to collect all the data I should need for the project. Therefore, I ended up with these data sources:

- **11v11**<sup>1</sup>: After over 350 lines of code for exploration, web scraping and formatting, I scraped over 10000 pages using python packages *requests*<sup>2</sup> and *BeautifulSoup*<sup>3</sup> to collect features such as data links, teams, date, referee, stadium, players, substitution details, id and scorers from every single page.
- **premierleague**<sup>4</sup>: After nearly a couple hundred lines of code for exploring and collecting the data, I web scraped 11149 pages using *requests* and *BeautifulSoup* to collect features such as referee, teams, date, score and unique id from each page.
- **football-data**<sup>5</sup>: Downloaded CSV per season, collecting around 50 features (details on features can be found on the site<sup>6</sup>). I then merged them together and removed unwanted columns in Jupyter Notebooks. Examples of features include same as above entries, as well as betting odds and in-game data such as number of corners, cards, etc.
- **physioroom**<sup>7</sup>: Manually collected per-season injury data (number of injuries in season), then merged and reformatted.

---

<sup>1</sup><https://www.11v11.com/competitions/premier-league/>

<sup>2</sup><https://2.python-requests.org/en/master/>

<sup>3</sup><https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

<sup>4</sup><https://www.premierleague.com/results>

<sup>5</sup>[www.football-data.co.uk/englandm.php](http://www.football-data.co.uk/englandm.php)

<sup>6</sup>[www.football-data.co.uk/notes.txt](http://www.football-data.co.uk/notes.txt)

<sup>7</sup><https://www.physioroom.com/news/>

- **wikipedia**<sup>8</sup>: Used *wiki-table-scrape*<sup>9</sup>, a third party wikipedia table scraping python package to collect data on all managers throughout the history of the premier league.
- **transfermarkt**<sup>10</sup>: Web scraped around 30 pages (one per season) to collect details about transfer income and expenditure for each team, then merged into one file to work with.

After this, I theoretically had all the data I would need throughout the project, and now just needed to work towards getting a subset of this data ready for modelling and predictions.

### 3.1.2 Data Cleaning

In order to use the data (or a subset) for modelling later on, the data needs to be correct and complete. Since each prediction uses information from all past matches, the data has to have no errors and therefore I chose to validate my main dataset from 3 different sources: **11v11**, **premierleague** and **football-data**.

To validate these datasets against each other, I need to set them up for merging, which requires cleaning each dataset individually before merging them and cleaning the result. Overall, I was very surprised at how messy and incomplete these datasets were, and it is something that took much longer than intended to fix. This is certainly something I would plan better for next time.

#### 11v11 Cleaning

Both the referee and stadium columns are difficult to create features from, but I still cleaned them up regardless. This required generating the unique sets of each to spot duplicates, missing values and incorrectly spelt entries. This also required a lot of manual replacing, taking far longer than it should have.

The DataLink column also conveniently contained the team names, date and id for every match, allowing me to easily create them columns from the link values. I also chose to put all string columns into lower case formats, and reduce all team names to their shorter versions, as is standard Data Science practice.

#### premierleague Cleaning

Thanks to previous cleaning requiring that columns in this dataset are also cleaned, many values here were already correct and consistent. One major issue that arose with this dataset was the appearance of non-PL games. The easiest way to remove all games that weren't premier league games, was to create a blacklist of teams that appeared in the dataset, but have never played in the premier league.

After looking into this problem, I discovered that the issue was when past or future premier league teams played each other, whilst not in the Premier League. Fixing this required me to create a dictionary with key:value set as team:list-of-seasons-in-EPL, which I had to then check every match with.

---

<sup>8</sup>[https://en.wikipedia.org/wiki/List\\_of\\_Premier\\_League\\_Managers](https://en.wikipedia.org/wiki/List_of_Premier_League_Managers)

<sup>9</sup><https://github.com/rocheio/wiki-table-scrape>

<sup>10</sup>[https://www.transfermarkt.co.uk/premier-league/startseite/wettbewerb/GB1/saison\\_id/2017](https://www.transfermarkt.co.uk/premier-league/startseite/wettbewerb/GB1/saison_id/2017)

### Merging 11v11 and premierleague datasets

Before merging, I had to complete a pre-merge clean to each dataset, including arranging all columns in similar orders, renaming columns, reordering rows to chronological order and other small changes.

Merging them, I created a **test\_equal** function for each column that had a value from each dataset, allowing me to quickly check which columns agreed and which didn't. Merging also raised some issues that needed cleaning, some manually.

Then I went on to clean this merged dataset, giving the following header: {'premid', '11id', 'season', 'date', 'home', 'score', 'away', 'ref', 'stadium', 'h\_scorers', 'a\_scorers', 'h\_players', 'a\_players'}.

### football-data Cleaning

I chose to hold onto **Shots** and **Shots on Target**. This allowed me to (see section 3.5.1 **Changes to Data - Adding Features** create columns called “**Shots Per Game**” and “**Shots Per Target on Game**” for each team so far.

Alongside removing most columns, I also reformatted the date column using the **to\_datetime** function from Python Package **Pandas**. I used this same technique to format the dates in other datasets too. Another very important cleaning step from this phase was the data range. Since the football-data dataset only had good data from the seasons since the year 2000, I chose to limit the main database to this range too.

### Final Merge

Now all I had to do was merge the 11v11 and premierleague merged dataset with the football-data cleaned dataset. Using the same techniques as before, I merged and cleaned them successfully, with minimal disagreements and issues, which hugely increased my confidence in the validity of the dataset I am working with.

### 3.1.3 Data Generation

In order to create some very useful and easy to work with features to add to our dataset, I decided to create a **league table generator** function, that allows me to add features for each team for each game.

In order to simulate the League Table I would need a league table to consist of 20 teams. Following this encapsulation model, I created a **Team** class consisting of the following attributes: **name**, **points**, **goals\_for**, **goals\_against**, **wins**, **draws**, **losses**, **played** and **clean\_sheets**.

Then, I added a **add\_result** function, that took the goals scored from each team in the match, and a **isHome** team boolean indicator. This function simply compared the goals scored from each team, and used the **isHome** value to decide how to modify the current teams league data. After adding each result to the teams individual scorecards, I then used a **finalise\_table** function to arrange the teams into league table position, and output the league table. Position is solved by the following values respectively: Points, Goal Difference, Goals Scored. Adding a position column gave me the final table.

	season	hPos	hGSPG	hGCPG	hCSPG	hPtsPG	aPos	aGSPG	aGCPG	aCSPG	aPtsPG	target
<b>0</b>	1	1.0	0.0	0.0	0.0	0.0	14.0	0.0	0.0	0.0	0.0	0
<b>1</b>	1	4.0	0.0	0.0	0.0	0.0	16.0	0.0	0.0	0.0	0.0	1
<b>2</b>	1	5.0	0.0	0.0	0.0	0.0	13.0	0.0	0.0	0.0	0.0	2
<b>3</b>	1	6.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	1
<b>4</b>	1	7.0	0.0	0.0	0.0	0.0	19.0	0.0	0.0	0.0	0.0	1

Figure 3.1: Header of Initial Dataset

Running this generator on each entry in the dataset gave me a full up-to-date league table to work with at each game. From this, I pulled some useful statistics. After some thinking, I went for the following features to extract from the league tables for both home team and away team (e.g. hPos and aPos are two features, covering the league position of the home team and the away team respectively): **hPos** (League Position), **hGCPG** (Goals Scored Per Game), **hGCPG** (Goals Conceded Per Game), **hCSPG** (Clean Sheets Per Game) and **PtsPG** (Points Per Game).

After adding these features to the dataset, I went on to finalise the dataset for the models.

### 3.1.4 Finalising Dataset for Models

#### Target Column

Although I experimented with various ways of representing the result as the target column, the main objective of the project is to be able to predict Home Win, Draw or Away Win, and therefore I set the target column to values 2, 1 or 0 respectively to represent those results.

#### Removing Unwanted Columns

I needed a dataset that is easy to use immediately after this stage, and therefore I removed columns that are not usable in their current state. One type of column I removed is string columns, such as “referee”, as columns need to be categorical or numerical. Additionally, I removed the id columns, as they’re only useful for reference not predictions. Removing these, and the players and scorers columns, I was left with the dataset header shown in Figure 3.1.

## 3.2 Analysis of the Problem

The goal of the first spiral is to get some initial results, experiment a bit with models and get a more in-depth understanding of the problem and chances of success. In order to do this, I first decided to analyse the problem of predicting football matches as a data scientist, looking for trends and patterns within the data.

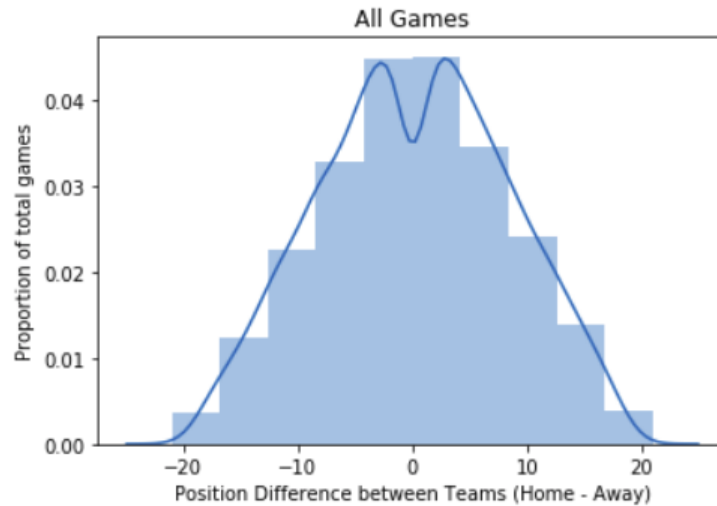


Figure 3.2: Pos-diff Distribution across all Games

### 3.2.1 Unbalanced Data

The first observation I made when exploring the data was the fact that the target values are not evenly represented. More specifically, in the dataset of 10126 matches: 4675 are Home Wins, 2775 are Away Wins whilst 2676 are Draws. This simple statistical fact raises an interesting property of the problem that is as follows: **By simply always predicting the Home Team to win the match, we can achieve an accuracy exceeding 46%.**

### 3.2.2 Current League Position

With a few lines of code, we can determine that in 46% of the games, the team with the better league position at the beginning of the match wins. Note that this method ignores draws, and simply checks how often the higher ranked team is the winner. And after some experimenting, trying to consider draws only makes this accuracy worse. This does imply that draws are going to be difficult to predict, which happens to be a large factor in this project.

This lead me to look at how, for each match outcome, the difference between the league position of the teams is distributed. Firstly we look at how the position difference is distributed across the whole dataset, so we can compare each outcome to the underlying trend. This is shown in Figure 3.2.

As expected due to all teams playing each other an even number of times, we get a very symmetrical normal distribution. After this, we look at the same distribution but this time split by outcome. A couple interesting observations can be taken from this, shown in Figure 3.3 below. The standard normal distributions slightly shifted to the left for Away Wins and Home Wins implies there exists a trend towards a higher place team performing well. Additionally the plot for Draws stands out by showing a linearly declining proportion of games as we increase in position difference between the teams. This simply implies that draws are far more likely when teams are close together in the league table, which is exactly what we'd expect.

All of these pieces of information together suggest that whilst positional difference can be a useful predictor, it doesn't represent the whole problem well enough as there are just

as many cases where it isn't a useful predictor as there are cases when it is.

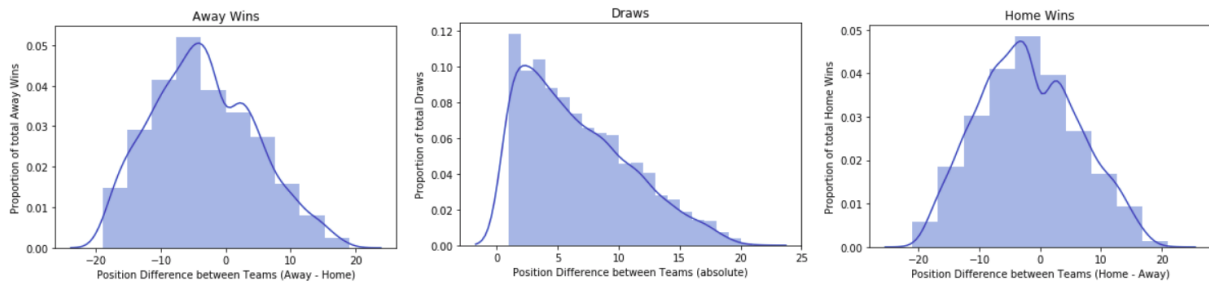


Figure 3.3: Pos-diff Distribution across all Games, split by result

### 3.2.3 League Table Statistics

There are various statistics we can collect just from the league table standings, many of which are outlined earlier in section 3.1.3. We can plot a **pairplot** from python package **seaborn**, as shown in Figure 3.4 below. This gives us a scatter plot of all pairs of features for the Home Team specifically, allowing us to look at their correlations.

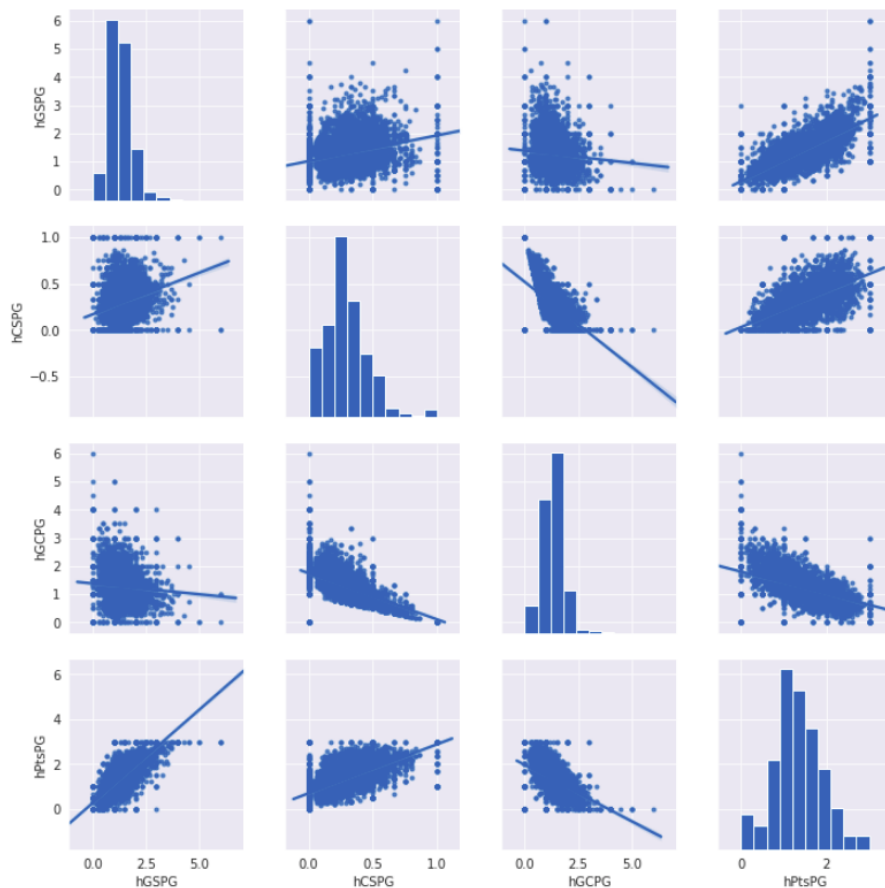


Figure 3.4: Seaborn Pairplot of League Table Statistics in Initial Dataset

The most stand out correlations are the obvious ones, for example hCSPG and hGCPG. Clean sheets are defined as conceding zero goals in that game, therefore it's



no surprise that the average number of clean sheets per game correlates with the average number of goals conceded per game fairly strongly. On the flip side, one pairing that seems to lack a significant correlation is between hGSPG and hGCPG. This implies that goals scored and goals conceded aren't strongly correlated, which is what you'd expect after thinking about it.

### 3.2.4 Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of dimensions in your data. For this project, that means trying to represent the 10 features described above as a small set of features. There's a few benefits to this, such as having a simpler model that is quicker to work with, but it's mainly used to determine if the data is **linearly separable**.

#### Linearly Separable

Visually, linear separability is demonstrated by reducing the dimensions of the data down to 2 dimensions, and seeing if you're able to draw a line between clusters that separates the different entries by some categorical value. Therefore linear separable datasets are far easier to predict with, because the outcomes are easily split up based on some combination of features. Discovering if your dataset is linearly separable before building any models is hence a useful and important step.

#### Principal Component Analysis (PCA)

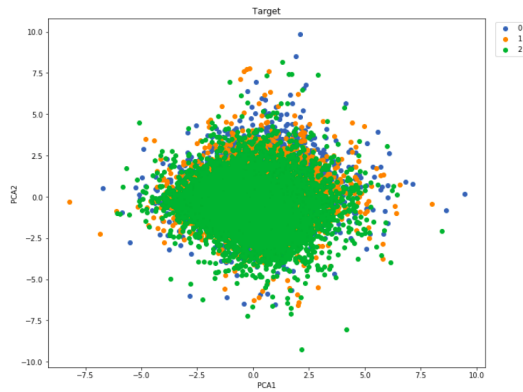
Principal Component Analysis (PCA) works similarly to how linear regression does. Linear regression looks to build the best fitting line to a plot by reducing the error. PCA generalises this, allowing us to reduce an arbitrary number of data dimensions down to an arbitrary number of target dimensions, by minimising the Mean Squared Error (MSE) ( $MSE = \frac{1}{n} \sum_{t=1}^n e_t^2$ ). Doing this to a target of 2 dimensions using the **sklearn.decomposition.PCA** Python package, gives us a nice visualisation of our features plotted on one graph shown by Figure 3.5(a).

#### t-SNE

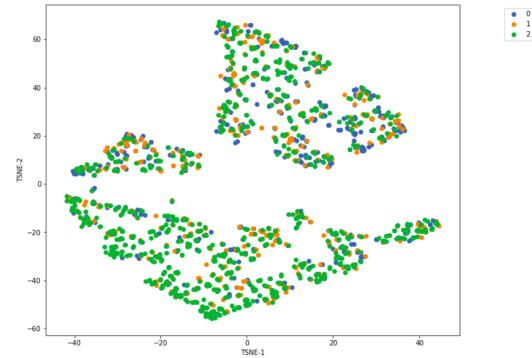
t-SNE stands for stochastic neighbourhood embedding with the t distribution, and is another method of dimensionality reduction. t-SNE works by computing the **nxn** distance matrix between all n data-points in the original dataset, defining a loss function, then generating a **2x2** distance matrix of the same data-points that minimises this loss function. Applying this method with the **sklearn.manifold.TSNE** Python package and colouring the scatter points by output class gives the plot shown in Figure 3.5(b).

#### Analysis of Plots

From both plots, it's fairly clear that the data is not linearly separable. This is demonstrated by the lack of clustering or separation of the different outcomes (Blue, Orange and Green), showing severe overlapping. Therefore as expected this problem is far more complex than a usual prediction problem.



(a) 2D PCA Applied to Feature-set



(b) 2D t-SNE Applied to Feature-set

Figure 3.5: Dimensionality Reduction Plots

### 3.3 First Spiral - Experimentation

There were a few main goals to this first spiral, mainly centred around experimentation and improving my understanding of the problem. I experimented with new models, techniques and dataset representations to better understand the problem.

Note: all models outlined in these sections and future sections come with their prediction accuracy's on the validation set. However, I completed more in-depth evaluations and an analysis of the results for each, detailed in Chapter 4 **Evaluation**.

#### 3.3.1 Initial Dataset

The initial dataset to be used is a slightly modified version of the one shown in Figure 3.1, details of which I outline below.

##### Train-Dev-Test Split

As detailed in the **Train-Dev-Test Split** section of section 2.5.5 **Data Science**, splitting the data for usage in training, testing during development, and final tests, is a very important part of the entire predictive process.

For this project, I decided the following splits:

- **Training Set:** All Seasons up to and included the 2009-10 season
- **Development Set:** Seasons 2010-11, 2011-12, 2012-13, 2013-14, 2014-15 and 2015-16
- **Test Set:** Seasons 2016-17 and 2017-18

This ends up as roughly a 69:23:7.7 Train-Dev-Test split, fitting comfortably inside the professionally recommended splits ratios.

##### Normalising Features

Features tend to work on different scales, for example a dataset for predicting house prices may have two features: the number of bedrooms, and the last sale price of the house. The

bedrooms feature are small single-digit numbers, whereas the last sale price is likely a six-figure number. Normalisation is only required when features have different ranges, which our values do. For example, hCSPG already has a range between 0 and 1, whereas hPOS can vary from 1 to 20, and so they'd be weighted very differently in the NN regardless of their usefulness.

### 3.3.2 Models

Since we know the data is not linearly separable, this controls what sort of models we'd expect to be effective. For example, if we build a Neural Network we're going to need at least one hidden layer to model the complexity of this problem, potentially trying a deep learning approach too where we use more hidden layers. Here are the models I tried at this stage.

#### Naive Bayes Classifiers

Using the `sklearn.naive_bayes` Python package, I built a couple Naive Bayes Classifiers. Firstly, I built and trained a BernoulliNB classifier. After training on the dataset, it achieved a 45.08% accuracy on the development set. This is a very poor accuracy, as it doesn't even beat the naive predictor described in section **3.3.1 Unbalanced Data**. Secondly, I built and trained a GaussianNB classifier, which performed far better after training, achieving a 48.55% accuracy.

#### K-Nearest Neighbours Classifier

Converging on a value for the Hyperparameter K (number of neighbours), I find that with K=193 the model manages to reach an accuracy of 49.04% on the development set, the best prediction accuracy so far. The classifier, and the methods used to converge on the optimal K, are outlined later in section **3.6.2 Models** as it's used more effectively there.

#### Keras NN V1 - An Initial Attempt

For my first Neural Network (NN) model, I decided to build a simple one in Python package **Keras**. For this first NN, I used all standard and recommended hyperparameters (refer back to section **2.5.3 Hyperparameters** for more information) to get some quick and likely good predictions: Sequential Model; One hidden layer; ReLU activation function; soft-max output function; Adam Optimiser; Sparse Categorical Cross-entropy Loss function and varied the number of neurons in the hidden layer as well as the number of epochs to end up with 12 neurons and 15 epochs performing the best under these conditions. This model was able to predict at a 48.95% accuracy on the development set.

#### Keras NN V2 - Extra Hidden Layers and Overfitting

For this model, I experimented with extra hidden layers, using the same hyperparameters as above. A phenomenon I noticed during this step was **overfitting**. Overfitting refers to when a model gets too good at predicting results from the training data, and loses its ability to generalise. You can often see examples of overfitting when the training accuracy greatly exceeds the test accuracy, which is something I noticed multiple times during this

stage. Overfitting is often caused by a model that is *too flexible*, as it moulds too closely to the training set examples, rather than generalising and being closer to the underlying patterns. This makes it worse at predicting unseen matches.

As I added more layers and neurons to a Neural Network, I saw this overfitting occur. For example, when I tried using 3 hidden layers each with 12 neurons, after 40 epochs my model was reaching almost 50% in training data accuracy, but when used on the test set the accuracy shot down to 47%.

There are a few ways to deal with overfitting, outlined below.

### Overfitting: Reducing Network Capacity

This is simply reducing the number of neurons, epochs and hidden layers. After experimenting, I found that no matter what combination I tried, it was very difficult to get a significantly better result. Exceeding the 49% accuracy level on the development set was rare and if surpassed it was insignificantly so. This does however potentially give us some room to try other overfitting techniques, to see if we can use these more complex models to get the higher scores on the development set.

### Overfitting: Regularisation

Another technique for addressing overfitting is regularisation. Here, we apply costs to the larger weights, through the loss function of the network. It constraints the coefficient estimates towards zero, discouraging the learning of a complex model. Regularisation also significantly decreases the variance of a model, without huge changes in its bias. This gives us a simpler model that is forced to learn only the most relevant patterns.

Experimenting with varying networks and L2-regularisation values, I found that many variants converged to an accuracy of 46.44%, and stayed static over many epochs. To me this implies that the resultant model became *too simple*. After investigation, I discovered that the model was predicting the Home Team to win *every single time*. This is clearly a problem, this is a far too simple model.

As I outlined in section **3.3.1 Unbalanced Data**, predicting Home Team every time is a fairly good predictor. It seems the model has figured that out too, and then has identified some cases where predicting the away team is also a better idea, to bump that prediction accuracy a little beyond 46%. Therefore we have a problem - we're going to need to handle this unbalanced data problem, and try to get the model to not just always predict the Home Team to win. The models are never predicting draws, despite them being over 25% of the outcomes which is clearly not the optimal strategy, so we're going to have to tell the model not to do this.

This problem is handled in the next version of the Keras model.

I did also experiment with Dropout, with no success again.

### Keras NN V3

The best way to deal with unbalanced data is to either: Balance the data manually - i.e. remove/ add data points; or tell the model that the data is unbalanced, getting it to apply weights to the examples found in the training set.

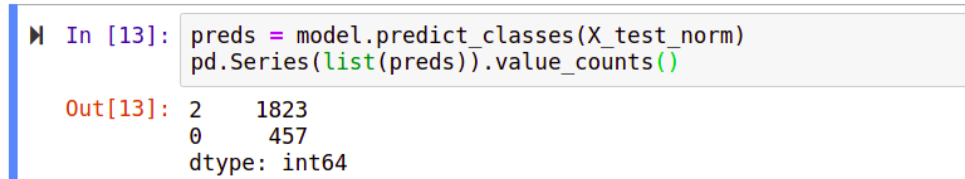
For our problem, it seems counter-intuitive to remove a chunk of matches where the Home Team wins, especially since each match prediction relies heavily on knowing information of past results. Adding data points is also not feasible, as this dataset is all real world matches and we cannot just invent results without skewing the problem.

That leaves us with the second solution: applying class weights. This is fairly simple with **Keras**, allowing us to simply supply class weights to the training process. These weights are calculated by determining the proportion of entries in the training set (must be only the training set, since this is all we know at the time of the test set predictions) that have each outcome, and using these proportions to calculate a weight to assert to each row of the dataset. After calculating this, we end up with the following weights: Home = 1, Draw = 1.7294, Away = 1.7394.

Applying `class_weight` however drastically decreases prediction accuracy to 45.7% in the best case. However at least the model is predicting different classes. This is certainly a trade-off I had to consider.

### 3.3.3 Best Model So Far

The best performing models were actually our initial NN model - Keras NN V1 (when not using class weights), alongside the KNN Classifier. We achieved a 48.95% accuracy with 12 neurons on a single hidden layer, and 15 epochs, with all other hyperparameters outlined so far, whereas the KNN with K=193 gave a prediction accuracy of 49.04%. The class weights only seemed to make predictions worse. Furthermore, the initial model(s) were predicting the Home Team to win 80% of the time, predicting an Away Win the rest of the time as shown in Figure 3.6.



```
In [13]: preds = model.predict_classes(X_test_norm)
         pd.Series(list(preds)).value_counts()

Out[13]: 1    1823
         0     457
         dtype: int64
```

Figure 3.6: Code Sample and Output showing class prediction spreads for best NN model so far

### 3.3.4 Conclusion and Next Steps

Due to this overfitting issue, and knowing that class weights don't seem to help, we're going to need a completely new approach. Possibly a hybrid of a statistical predictor that hands over to a Neural Network in the tougher cases. We can also try to add more features and use that to improve our prediction accuracy - this is the approach I went on to try, as it seemed the most relevant and promising.

## 3.4 Second Spiral - Adding Features

In this spiral, I focused on how I can improve the dataset to try and get the best possible predictor out of a model that uses class weights. This allowed me to try out various

professional data science techniques.

### 3.4.1 Adding Features

Using the dataset we already have, alongside data collected from other data sources, I put it all together into a larger (in terms of features) dataset that holds most of the data we'll be using.

#### Shots and Shots on Target Per Game

We have the Shots and Shots on Target features from the football-data dataset already, however we need to turn them into something usable for the models, and therefore we can only use data *Known before the start of the current match*. To do this, I created a column that gives the number of shots (and shots on target) averaged per game so far this season, as of the beginning of this current match. This required walking through the dataset and collecting this data as I go, adding the values where necessary. These features should give a good overview of how often and how effectively the team is attacking.

#### Shooting Percentage

This feature is easy to add thanks to the other data we already have: Shots and goals scored. A shooting percentage for a team is simply the percentage of shots that end up in goals, i.e. the shooting *accuracy* of a team. This is an average so far this season.

#### Form Stats

A potentially hugely important and relevant feature yet to be added is **Form**. Form represents how well a team has been doing in their very recent games, which is likely going to be a great predictor of near-future success. I decided to collect 2 Form stats per team, at 3 different intervals: Points Per Game and Goal Difference, each in the last 3, 5 and 7 games. These encapsulate the recent success of the team, in varying levels, as well as the severity of the success (or failure) given roughly by the Goal Difference. For example if a team has a high average of points per game in their last 3 games (i.e. near 3 points per game), we know they're doing well recently. Additionally, if their Goal Difference is very high as well, we know that they're not just winning but they're winning comfortably.

The reason for collecting these stats in various intervals (last 3, 5 and 7 games) is because of their varying reliability and sensitivity to outliers. Also, a 7 Game PPG score is based on more data-points, and is therefore far more likely to be statistically representative of their overall "form", whereas although a 3 Game PPG may be less reliable, it picks up form shifts very quickly.

These features were added by walking through the dataset chronologically, and collecting the results and scores as needed.

#### Last Season Finish

Whilst information about this current season is incredibly useful in most cases, it doesn't tell the whole story, and especially at the beginning of the season it sometimes does more

harm than good. Therefore, adding the league position from last season allows us to take into consideration more than just this season. A team may perform very well one season, and then start the next season poorly. But knowing their performance of last season, can help us predict a potential turnaround at the start of this next season.

This feature was added using the existing league table generating functions I had built earlier on in the project, referenced in section **3.1.3 Data Generation**,

### Transfer Data

Using data collected from data source **transfermarkt**, I was able to add 3 features per team: **Income**, **Spent** and **Net Spend**. These are vital to consider because a teams spending can drastically change their season.

These values were formatted from the dataset, then merged into the master dataset used in the first spiral.

### 3.4.2 Changes to Data - Different Range

The football-data dataset only had valid shot stats from 2000 onwards, therefore I decided to use data only from 2000 onwards. This means I can use this extra feature set, as well as only using more recent results. Experimenting the effect of this dataset change on the best performing models, I saw no decrease in model accuracy, and actually saw improvements in some models.

### 3.4.3 Models and Evaluation

#### Keras NN V1

Using the same hyperparameters that performed the best on the previous spiral (12 neurons on single hidden layer, 15 epochs), I was able to produce the best results so far with a prediction accuracy of 50.79%! This being so much higher than the previous results with class weights added, implies that at least some of the newly added features are very useful in predicting the outcomes of football matches. In this result, the model predicted the Home Team to win 48.22% of the time, (getting over half of these predictions correct), which is much closer to the true percentage of Home Wins in the underlying dataset.

#### Keras NN V2

After trying many combinations of the hyperparameters, I was able to reach an impressive 51.78% (average over 10 attempts, many times exceeded 52.5%) accuracy with the following parameters: Two hidden layers, each with 12 neurons, 50 epochs, other hyperparameters same as before.

This is now by far my best model so far, and is about as good as I'd ever aimed to achieve at the beginning of this project (note the mention of a 52% accuracy in the Project Proposal).

### 3.4.4 Conclusion and Next Steps

This was an incredibly successful Spiral, with my feature additions and model tuning giving me a very good prediction accuracy. This accuracy is near the upper end of what I thought was possible for this problem, so I was very happy to see these results after all the hard work on Data Curation, as it backed up my points made in section **2.5.6 Data Science** on why Data Curation is such an important part of Data Science projects.

## 3.5 Third Spiral - Feature Importance

Again, all data is normalised before being put through a model, and the same Train-Dev-Test splits were used.

### 3.5.1 Permutation Importance

Feature Importance is the process of determining which features are more important in the prediction process. The technique I used is called **Permutation Importance**. This works by randomly permuting the values of a column, and seeing how the overall prediction accuracy is effected. If the prediction accuracy worsens significantly, we know that this feature must be an important feature in predicting the outcome of the game.

In order to apply Permutation Importance, you need a simple model. Here, I decided to use a Decision Tree Classifier, detailed further in the next section. On the first run of Permutation Importance on the dataset, I got the feature weights shown in Figure 3.7.

Weight	Feature
$0.0796 \pm 0.0199$	aPosLS
$0.0274 \pm 0.0130$	hPosLS
$0.0154 \pm 0.0083$	hGD5
$0.0025 \pm 0.0010$	aSpent
$0 \pm 0.0000$	aNet
$0 \pm 0.0000$	aCSPG
$0 \pm 0.0000$	aSTPG
$0 \pm 0.0000$	aSPG

Figure 3.7: Feature Importance's on Dataset

Here we see that the following features positively helped the predictions: aPosLS, hPosLS, hGD5 and aSpent. Interestingly, this implies that the position the teams finished last season is actually the best predictor of football matches, which is fairly believable since teams won't tend to deviate much from how they performed last season. All other features have the weight  $0 \pm 0.0000$ , implying that they don't effect predictions when randomised. Does this mean they're useless for predicting the outcome of football matches? Well, no, it just means they're overpowered by the above features.



### 3.5.2 Models

#### Decision Tree Classifier

The first model, mentioned above, was a Decision Tree Classifier that managed to perform surprisingly well, achieving an accuracy of 51.91% with the criterion gini index, and a 52.04% accuracy with the criterion information gain, which is therefore the best model so far.

#### Keras NN V1

After applying permutation importance over multiple iterations, I was able to reduce the working set of features from 36 features down to 19. Applying the best-so-far model to this dataset (Two Hidden Layers, each with 12 neurons, with 50 epochs of training), the model achieved an **average** of a 51.6% accuracy over 10 attempts on the development set.

#### Keras NN V2

In order to find the best hyperparameters, I applied a GridSearch to find the optimal number of layers, neurons on each layers and epochs. After running that over all sensible ranges (1-3 hidden layers, 6-24 neurons, 10-100 epochs), the best performing model was the V1 mode. Tested 10 times again, the accuracy of this model being trained then tested on the development set ranged from 50.01% up to 52.9%, which equated to an average of 51.67%.

#### Naive Bayes Classifiers

Firstly, I built and trained a BernoulliNB classifier, achieving a 48.88% accuracy on the development set. Secondly, I built and trained a GaussianNB classifier, which performed far better after training, achieving a 51.84% accuracy, identical to the best model so far (Decision Tree Classifier).

#### K-Nearest Neighbours Classifier

For this Classifier, I chose a suitable K (number of neighbours used in the algorithm) that optimises prediction accuracy to unseen data. In light of this I built 500 models, covering all values of K from 1 to 500 inclusive. I then fit them on the training data, and return their accuracy. With this information, I plotted a graph of K vs Accuracy shown in Figure 3.8.

As you can see from the plot, many values of K were able to build a model that could exceed an accuracy of the previously-highest 52%, with the optimal accuracy being achieved at K=109. Retraining a model using K=109 and testing it on the validation set gave an astonishing 53.36% accuracy as shown in Figure 3.9, firmly performing the best of all models so far, and again outperforming expert pundits.

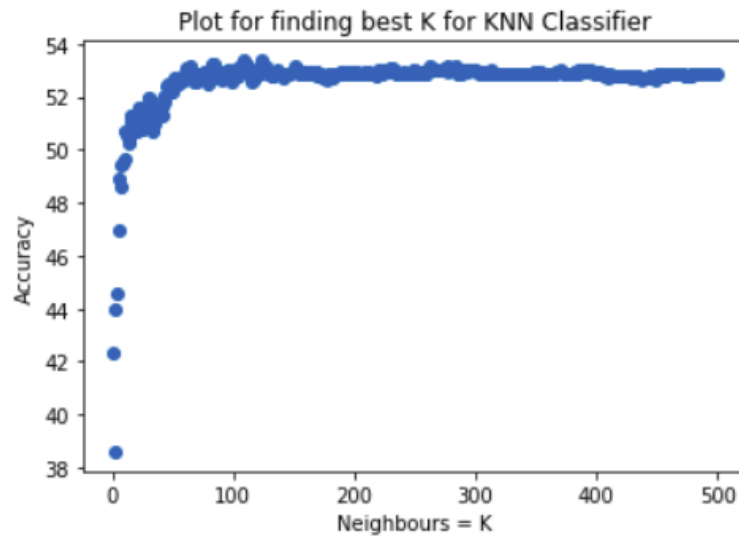


Figure 3.8: Plot to find optimal value of K Neighbours

```
In [10]: knn_subset = KNeighborsClassifier(n_neighbors=109)
knn_subset.fit(X_train_subset,y_train)
print(knn_subset.score(X_test_subset,y_test)*100)

53.35526315789474
```

Figure 3.9: Code Sample and Output for KNN Classifier where K=109

### Support Vector Machine

For this Classifier I tried each kernel type (linear, RBF and polynomial), and the RBF Kernel performed best, hitting a 53.09% accuracy.

### 3.5.3 Conclusions

An interesting set of conclusions can be reached when we look into why the KNN Classifier outperformed all Neural Networks built in this project. Usually Neural Networks require a lot more data in order to perform well, and therefore the lack of prediction accuracy compared to models like the KNN Classifier may be highlighting the lack of data issue, where around 4000 matches just isn't enough to train a network to pick up complex patterns.

## 3.6 Final Model

Most classifiers trained very quickly, so I didn't worry too much about training speed when deciding on a best model. We may as well also just pick for accuracy, as it is the main goal of this project. With these things in mind, I chose the model with the highest prediction accuracy on the validation set, as detailed below.

### 3.6.1 Model Chosen

The model chosen was the K-Nearest Neighbour model detailed in section **3.6.2 Models**. After converging on the hyperparameter K (number of neighbours) of 109, the model was able to predict at a 53.36% accuracy on the validation set. This far exceeds any ambitions and goals I had at the beginning of the problem/ This all implies that the extensive work on curating and improving the input dataset was worth the effort.

As stated at the beginning of section **3.4 First Spiral**, an in-depth evaluation of the final model and other models is given in the Evaluation Chapter.

However, the aim of the project is to predict new games given past information, and that value of K is specific to the target range. So I instead propose an adaptive strategy, depending on the season you are predicting. Furthermore, we design the model as follows:

Say we want to predict the outcomes of matches from the current season, and we have 20 past seasons of data to work with too. This details a dataset consisting of 21 seasons of data. Set Train:Dev:Test to be around 16:4:1 seasons respectively. Then we tune the model parameters so that when it's trained on the training set, the development set prediction accuracy is maximised. This ensures we tune a model to be accurate in the most recent data we have access to in the training set. Then, we use this tuned model to make our predictions, on the held-out season of matches. This model is implemented as a KNN Classifier as it performed the best previously, where K is the adaptable hyperparameter that is tuned for the development set.

A detailed analysis of this adaptive KNN model is supplied in section **4.2 Final Model Evaluation**.

### Final Dataset

The final dataset has 6820 rows (380 games a season for 18 seasons from 2000-01 to 2017-18), and 19 features, shown by Figure 3.10. Note, the dataset is always normalised using `tf.Keras.utils.normalize` before being passed to models.

	aPosLS	hPosLS	hGD5	aSpent	aPtsPG	hSPG	hGCPG	aPos	aSTPG	aGSPG	hSTPG	aSPG	aGD7	hPos	aGD5	hPtsPG	hSpent	aPTSPG7	hGD7
0	18	18	0.0	18.07	0.0	0.0	0.0	13.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	18.72	0.0	0.0
1	9	5	0.0	8.92	0.0	0.0	0.0	20.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	47.15	0.0	0.0
2	12	14	0.0	29.52	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	17.10	0.0	0.0
3	15	16	0.0	0.00	0.0	0.0	0.0	17.0	0.0	0.0	0.0	0.0	0.0	7.0	0.0	0.0	11.99	0.0	0.0
4	13	3	0.0	31.03	0.0	0.0	0.0	8.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	47.84	0.0	0.0

Figure 3.10: Header of the Final Dataset

## 3.7 Extensions

Some extensions were completed. I covered the statistical predictor in section **3.2.1 Unbalanced Data**. Deep Learning was first implemented in section **3.3.2 Models**.



# Chapter 4

## Evaluation

This project has been a huge success, from work completed to the success criteria being far exceeded. I'm very happy with the results found throughout the dissertation that together clearly demonstrates a gradual improvement through research, use of professional techniques and industry standard models, and iterative adaptations. In this chapter I speak about the models, process, results and the project overall in more detail, evaluating the results and effectiveness of approaches used.

### 4.1 Model Evaluations in Spiral Process

Here I start by evaluating the models used in the Spiral Process. In the Implementation Chapter, I outlined the copious amount of models built, and gave their prediction accuracy on the development set which was used to fine-tune and choose models. In this section I looked into the results a little further, and outlined potential next steps.

#### 4.1.1 First Spiral Evaluation

The initial set of models quickly surpassed my project goal of a 40% prediction accuracy. Beyond this, various models gave a wide range of results.

Looking more into the predictions themselves, it was clear from early on that draws were particularly difficult to predict, with the majority of the models correct predictions coming from correctly predicting a Home Team Win. For example, a few lines of code show us the number of correct predictions made by our best model so far (the KNN Classifier), per class, below in Figure 4.1. All predictions were similarly spread out - the models could not predict draws correctly. This is mainly due to the fact that models didn't ever predict draws as highlighted in Figure 4.2, because of the Home Team bias of this dataset (and the underlying problem).

Not being able to predict draws correctly (or ever) left me with a difficult decision to make: Do we continue this approach and try to optimise it, or do we force the models to predict draws more often and try to optimise that? Whilst no approach is necessarily better, since our own goal is prediction accuracy overall, it seemed more sensible to both and see what works.

Furthermore, as detailed in sections 3.4.2, 3.4.3 and 3.4.4, I decided to force the Neural Networks to take into account class weights, attempting to solve the problem outlined

```
In [24]: correct_pred_dict = {0:0,1:0,2:0}
for i in range(len(y_pred)):
    pred = y_pred[i]
    real = y_test[i]
    if pred == real:
        correct_pred_dict[pred] += 1
correct_pred_dict

Out[24]: {0: 180, 1: 6, 2: 932}
```

Figure 4.1: Code Sample and Output showing that the KNN Classifier correctly predicted 180 Away Wins, 932 Home Wins but only 6 Draws

```
In [30]: from collections import Counter
Counter(list(y_pred))

Out[30]: Counter({2: 1899, 0: 368, 1: 13})
```

Figure 4.2: Code Sample and Output showing the counts of each class in the predictions of the test set for the KNN Classifier

above. This decreased the overall accuracy as expected (we know predicting Home Win often works well and may still be the best strategy if optimised), but at least the model began to predict draws sometimes. This is shown in Figure 4.3.

```
In [13]: predictions = model.predict_classes(X_test_norm)
pd.Series(list(predictions)).value_counts()

Out[13]: 2    979
         0    933
         1    368
         dtype: int64
```

Figure 4.3: Code Sample and Output showing the **new** counts of each class in the predictions of the test set for the Neural Network (V3) after considering class weights

Whilst this may seem a step in the right direction, I still needed the accuracy to increase. I also ran into overfitting issues in this phase whilst trying to tune models to improve prediction accuracy. This could not be easily handled at this stage, implying I either needed more features or I needed to work with simpler models.

Therefore, the main conclusion from this stage was to massively increase the feature set, in order to tackle these issues surrounding predicting draws and considering class weights.

### 4.1.2 Second Spiral Evaluation

This section involved adding an abundance of new features to the model, from multiple sources. Overall, these features were a huge part of the success of this project, and I'm happy I spent the time researching them and implementing them into my datasets. Also changed range of dataset because of what data we had, and this seemed to be a worthwhile trade-off based on improved results.

The best indication of this improvement was from model V1, where I used the same hyperparameters as the previous best model, and managed to get an accuracy of 50.79%, an increase in almost 2% which is huge for this type of problem. Additionally, this model

was working with class weights, so despite this alongside a smaller training set to work with, it was able to increase accuracy and improve the spread of predictions, outlined by Figure 3.6 from earlier alongside Figure 4.4 just below here.

```
In [15]: predictions = model.predict_classes(X_test_norm)
         preds = list(predictions)
         pd.Series(preds).value_counts()

Out[15]: 2    733
         0    555
         1    232
         dtype: int64
```

Figure 4.4: Code Sample and Output for spread of classes in validation set predictions with best NN model from the first spiral

Comparing these two figures you see that the model is predicting draws far more often, and is still able to improve on the accuracy of when it barely predicted draws. This was a promising result for future spirals. On top of this success, fine tuning the NN model even further (to end up with 2 hidden layers, each with 12 neurons, and training it for 50 epochs) allowed me to improve the best model to achieve a 51.78% prediction accuracy. This was the first time in the project where I achieved my ambitious secondary project object outlines in the Original Aims section - getting a prediction accuracy close to that of professional expert human pundits (who get around a 52% accuracy in this problem). This huge success was also a key indicator that my dedication to focusing my efforts on improving the data rather than the models was a great decision.

This best model also predicted draws a little more often as class weights were being applied, however predictions were still less spread out than the model above, as shown in Figure 4.5.

```
In [12]: predictions = model.predict_classes(X_test_norm)
         pd.Series(list(predictions)).value_counts()

Out[12]: 2    739
         0    646
         1    135
         dtype: int64
```

Figure 4.5: Code Sample and Output for spread of classes in validation set predictions with best NN model from the second spiral

This improvement in prediction accuracy whilst predictions skewed further towards predicting Home Wins more often implies that the best model is likely to have a bias towards Home Wins - which makes sense because the underlying distribution does too.

### 4.1.3 Third Spiral Evaluation

At this point in the project I had a great predictor, a great dataset and a great understanding of the problem. The only things left to do fell into two categories: advanced industry standard techniques for fine-tuning and improving models, or exhaustive searches for improvements.

## Feature Importance

The first bit of work I did in this Spiral was of course on the data, as defined by the Spiral Process. Here I chose to work with a technique called Permutation Importance, described in more detail in section **3.6.1 Feature Importance**. An interesting discovery was that in each iteration, I'd run Permutation Importance, it would highlight around 4 features that are pretty much entirely responsible for the predictions, I'd remove these features and then the models would still perform nearly as well.

Furthermore, if you refer back to Figure 3.7, you see that 4 features are highlighted as doing all the work for the prediction accuracy to be 51.91%: aPosLS, hPosLS, hGD5 and aSpent. Given all other features had a weight of 0, you'd assume removing these 4 important features would drastically effect prediction accuracy. But after removing these four features, the model was able to predict at a 51.12% accuracy. Running Permutation Importance again on this dataset and model gave the following output, shown in Figure 4.6:

Weight	Feature
0.0558 ± 0.0139	aPtsPG
0.0270 ± 0.0055	hSPG
0.0086 ± 0.0049	hGCPG
0 ± 0.0000	aNet
0 ± 0.0000	aIncome
0 ± 0.0000	hCSDC

Figure 4.6: Feature Importance's on New Dataset where aPosLS, hPosLS, hGD5 and aSpent were removed

So now I have a new set of 3 features, which are doing all the work in predicting matches to a 51% accuracy (very high), yet these features had 0 feature importance before the previous four features were removed? Whilst this seems very counter-intuitive, it's actually fairly simple - it's because the best features are overpowering the rest of the features in the models. Now this process continued through 5 iterations of this process, leaving me with a set of 19 features that after excluding, the prediction accuracy had shot down to 47.7%. Additionally, the weights had all began to get quite small, rarely exceeding 0.01. Now that the feature set is smaller, models should train much faster and pick up patterns easier, allowing me to work with more complex models.

## The Models

Here I'll detail the results in a list, and analyse results further afterwards:

- **Decision Tree Classifier**(*criterion='entropy', max\_depth = 3, min\_samples\_leaf = 5*) - Reached a 52.03% accuracy.
- **Keras NN V1**(*Two Hidden Layers, 12 Neurons each, 50 epochs training*) - Reached a 51.6% accuracy - Note: This is the best prediction accuracy of all Neural Networks built so far, implying the Permutation Importance step has simplified the data enough to allow the NNs to learn patterns in the data faster and better.



- **GaussianNB**(*Defaults*) - 51.84% accuracy.
- **K-Nearest Neighbours Classifier**( $K=109$ ) - An astonishing 53.36% accuracy, strongly outperforming expert pundits.
- **Support Vector Machines**(*kernel='linear'*) - 53.09% accuracy, the 2nd best model so far.

The first thing to obviously mention is the outstanding results achieved by the KNN and SVM models. They both massively outperformed what I thought was possible at the beginning of this project. This is a very exciting discovery and is all thanks to the extensive focus on the data rather than the models.

It's also worth mentioning that again, the best models rarely predicted a draw. For example with the KNN model, out of 811 correct predictions in the validation set, only 5 of them were from predicting draws, despite draws representing about 25% of the outcomes. This really does raise the idea that trying to predict for draws may potentially be a mistake. However, it does also raise potential for some improvements if I can figure out a reliable way to detect draws.

```
In [11]: import time
from sklearn import svm
from sklearn.metrics import accuracy_score

t0 = time.time()
#Create a svm Classifier
clf = svm.SVC(kernel='rbf',gamma='scale') # RBF Kernel

#Train the model using the training sets
clf.fit(X_train_subset, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test_subset)

print("Accuracy: {}".format(accuracy_score(y_test, y_pred)))
t1 = time.time()
print("Time: {}".format(t1-t0))

Accuracy: 0.530921052631579
Time: 1.363900899887085
```

Figure 4.7: Code Sample and Output demonstrating the Prediction Accuracy and Time Taken for the SVM with an RBF Kernel

#### 4.1.4 Training Speeds

All models were trained almost instantly (except of course the Neural Networks), so this was something not worth exploring too much. This is mainly due to the lack of training examples we were able to supply to the models. The only exception to that almost-instant training times was when I trained the RBF Kernel SVM, but still it took under 1.5 seconds as shown in Figure 4.7.

The V2 Neural Network from section **3.5.3 Models and Evaluation** (the best NN model of this project) took 17.7 seconds to train, as shown in Figure 4.8. This is fully expected from a Deep Learning Neural Network, as is one of the main disadvantages of NNs especially when working with small training set sizes, as you don't get a benefit of improved prediction accuracy in return for the extra training times.

```

In [10]: import time
t0 = time.time()
model.fit(X_train_norm, y_train.values, epochs=50, validation_data=(X_test_norm,y_test), class_weight=class_weights)
t1 = time.time()

4180/4180 [=====] - 0s 68us/step - loss: 1.0169 - acc: 0.4758 - val_loss: 0.9976 - val_
acc: 0.5164
Epoch 45/50
4180/4180 [=====] - 0s 75us/step - loss: 1.0162 - acc: 0.4811 - val_loss: 0.9911 - val_
acc: 0.5184
Epoch 46/50
4180/4180 [=====] - 0s 65us/step - loss: 1.0155 - acc: 0.4746 - val_loss: 0.9984 - val_
acc: 0.5276
Epoch 47/50
4180/4180 [=====] - 0s 65us/step - loss: 1.0162 - acc: 0.4778 - val_loss: 0.9953 - val_
acc: 0.5138
Epoch 48/50
4180/4180 [=====] - 0s 64us/step - loss: 1.0150 - acc: 0.4868 - val_loss: 0.9910 - val_
acc: 0.5184
Epoch 49/50
4180/4180 [=====] - 0s 71us/step - loss: 1.0145 - acc: 0.4818 - val_loss: 1.0046 - val_
acc: 0.5026
Epoch 50/50
4180/4180 [=====] - 0s 63us/step - loss: 1.0147 - acc: 0.4763 - val_loss: 0.9884 - val_
acc: 0.5257

In [11]: print("Time: {}".format(t1-t0))
Time: 17.69850492477417

In [12]: val_loss, val_acc = model.evaluate(X_test_norm, y_test)
print(val_loss, val_acc)
1520/1520 [=====] - 0s 50us/step
0.9883934422543175 0.5256578947368421

```

Figure 4.8: Code Samples and Outputs demonstrating the Training of a NN, the time taken to train, as well as the validation set loss and accuracy

## 4.2 Final Model Evaluation

Now since the goal of the project is to be able to predict **future** football matches using only **past** information/ results, I chose to modify the final model slightly for use in future predicting scenarios. This gave me the adaptive KNN model outlined in section **3.7.1 Model Chosen**.

In order to evaluate this model, I chose to build it around predicting the results of the 2017/18 Premier League season. Therefore, using the final version of the dataset to test out this model, I set apart the final year of data (2018/19 season) for predictions at the end, I then separated out the four most recent seasons left into a development set of 1520 entries (4 seasons, 380 matches each season). The rest of the data was used as Training Data. Then, I normalised the values in the datasets, before building the KNN Classifier 150 times for the values of K between 1 and 150, testing it against the development set each time. This clearly showed an optimal K value of 84, which gave an accuracy of 53.36% on the development set.

Now we have an optimised model for the problem, I ran it on the 2018/19 season to predict the outcomes, which ended up having a 51.84% accuracy (very good accuracy for unseen recent data). As shown by Figure 4.9, these predictions include only 4 Draw predictions, 95 Away Win predictions and 281 Home Win Predictions.

```

In [16]: from collections import Counter
Counter(preds)

Out[16]: Counter({2: 281, 0: 95, 1: 4})

```

Figure 4.9: Code Sample and Output showing class prediction spreads for final adaptive model

## 4.3 Project Evaluation

The project was a huge success, and I'm very happy with the work completed and the outcomes. However, I feel there were plenty of things I could have done better, and there's still plenty of options for what to do next.

### 4.3.1 What went well

First and foremost the project was a success formally as it passed the success criteria comfortably, and also passed the secondary more ambitious success criteria I set for the project of rivalling the prediction accuracy of professional expert pundits. Additionally, the Spiral Model I implemented worked perfectly for this project, as it successfully allowed me to improve the data and models in iterations (Spirals), keeping things organised whilst keeping me focused on the next improvement to make.

The project was also extremely interesting, allowing me to learn a lot about Machine Learning techniques, implementing them in Python, working with Jupyter Notebooks, designing effective evaluation techniques and producing a project. The choice of Football was a perfect balance of difficult, unique but doable, with enough similar research done to keep ideas thriving, but leaving enough to figure out myself to make the project fun to work through.

### 4.3.2 What I'd do differently

At around the half way stage, I had decided that I'd spent too long working with data. From all the data curation steps described throughout this project, it always felt like I was doing a lot of work with no rewards. However, once I got to model building, the benefit of this approach was clearly shown, with drastic increases in prediction accuracy made throughout the project.

However next time, I would have done the data analysis work sooner. Too late into the project did I learn many useful statistics and patterns that I didn't get time to work with and outline here. For example, I began to look into what happens if we split the dataset up by how far apart the teams are in the current league table, to see if one type of match was far easier to predict than others. Due to lack of time, I was unable to finish this analysis, but it seemed like a route with potential. Similarly, I wanted to dive more into the issue surrounding the difficulty of predicting draws but modelling the problem as a binary "Predict Draw or Not Draw" problem, but this seemed to not work at first go.

I also would've asked for more advice earlier on. It's very easy to get stuck into the project, complete lots of work and feel like you've made progress, before realising a lot of it was unstructured and unnecessary. This was why I chose to implement a Spiral Model, which worked exactly as intended once I began to follow it.

### 4.3.3 Ethical Considerations

When predicting something that has a whole betting market behind it, it's worth considering the potentially unethical side to using Machine Learning for improving your chances. However, considering that the betting companies obviously know this themselves and have

their own models to counter it, and many companies exist to achieve exactly this goal, I'm certain it's not a major issue.

Another potential issue is the use of the data sources for this project, however combining knowledge of the robots.txt files that detail what we're allowed to scrape with the fact that this project has had no commercial benefits, all is good regarding data usage too.

#### 4.3.4 What next?

I certainly plan to continue working on this project in my own time, especially due to the high level of success achieved in this project.

Some things I'd like to try/ work on next:

- A pipeline for automatically collecting the necessary data for each new match every week, allowing me to build an automatic predictor system
- Exploring the problem of predicting draws
- Using other leagues and competitions so the dataset can grow significantly, to build models that are more generalised to the sport of Football as a whole, not just specific to the Premier League
- Turn all unused features/ data collected into further features. This was certainly a frustrating point of the project, as a lot of work went into cleaning features that were never used.
- Incorporate individual player data to the model, such as referencing players on form or highest goalscorer stats etc.

# Chapter 5

## Conclusion

This project has been a success. I surpassed a 40% prediction accuracy for predicting Home Win, Draw or Away Win. My final model managed to reach a prediction accuracy over 52% on test sets and 53% on validation sets, which rivals what the professional expert human pundits can do. The modified version of the Spiral Model I used was certainly the correct choice for this type of project, however some minor changes would've helped even more, such as allowing for a way to explore new avenues through the project. One other conclusion is that Neural Nets appear to work less well than their hype suggests, and simpler models worked better (see section 4.1.3).

If I were to do the project again, I would do a couple things differently such as requesting more advice and guidance earlier on. I will continue to work on this project in my own time because this problem fascinates me, and I feel there is plenty left to explore.



# Bibliography

- [1] Norman E Fenton Anito Joseph and Martin Neil. Predicting football results using bayesian nets and other machine learning techniques. 2006.
- [2] Burak Galip Aslan and Mustafa Murat Inceoglu. A comparative study on neural network based soccer result prediction. 2007.
- [3] Barry Boehm. *A Spiral Model of Software Development and Enhancement*. ACM SIGSOFT Software Engineering Notes, 1986.
- [4] Josip Hucaljuk and Alen Rakipovi. Predicting football scores using machine learning techniques. 2011.
- [5] Daniel Pettersson and Robert Nyquist. Football match prediction using deep learning recurrent neural network applications. 2017.
- [6] RudrakshTuwani. Football data analysis and prediction.





# Appendix A

## Project Proposal

Computer Science Tripos – Part II – Project Proposal

Predicting the outcomes of English Premier League  
matches

jm2129

12 October 2018

**Project Originator:** jm2129

**Project Supervisor:** Helena Andres-Terre

**Director of Studies:** Prof Alan Mycroft

**Project Overseers:** Prof A. M. Pitts & Dr R. Mantuik

### Introduction

*The largest betting market in the world is football, and the most popular league in the world is the English Premier League (EPL). I will be investigating 25+ years of results, and trying to predict their outcomes at an accuracy that rivals experts, and other literature on this subject.*

A lot of literature uses post-match data, but those that do not tend to investigate potential features in little depth. I will be using only pre-match data, and will be investigating as many potential features as I can. The aim is to end up with a small subset of these features that can help my system predict well.

Data will be generated through a mix of web scraping (allowing me to compare and validate data from multiple sources), and manual collection where more appropriate. Each season has around 400 games, therefore my data will contain around 11000+ rows. In terms of features I expect to collect no more than 30 features worth of data, and expect my final data set to contain no more than 20.

I will be experimenting with multiple common classification techniques such as Logistic Regression, Support Vector Machines (SVMs), K-Nearest Neighbour (KNN), and

exploring the use of Deep Learning. A strong focus will be on selecting the most important features, which includes testing out every feature I can collect data on. Through looking at the literature and pairing it with my own domain knowledge, I know this is something that can be improved on. I will use multiple evaluation techniques such as comparing accuracy and error rate to literature and experts.

## Starting point

Literature can predict the winner of a match at a very high accuracy<sup>1</sup> given live match data such as the number of corners in a match at a certain time. *However* I wish to only use pre-match data, not live match data, as this would produce more interesting and useful results.

There are various data sets for a mix of pre-match and post-match data at football-data<sup>2</sup>. I may use some data from this, however all other data I collect will be via web scraping from multiple websites I decide upon.

I will be using Python to web scrape, clean data, engineer features, apply machine learning techniques, build Neural Networks and evaluate the systems. I have experience with cleaning data and minimal experience applying some basic ML techniques in Python. Everything else will require learning through books, tutorials and documentation.

I have some experience writing in L<sup>A</sup>T<sub>E</sub>X, which is how I will write my dissertation.

I have been a keen follower of the EPL my entire life, and therefore have knowledge of the domain and what features should help predict the outcome.

## Resources required

For this project I will be using my personal laptop: ASUS ROG GL552VW. This has a quad-core i7, with a 256GB SSD and a 1T Hard Drive. The laptop is dual boot Windows 10 and Ubuntu 16.0.4. For larger programs I will be using Ubuntu, with Python 3 installed and using the VSCode IDE, but most of my code will likely be produced on Windows using Jupyter Notebooks with the latest Anaconda distribution. Most packages I will use are in the Anaconda distribution, however I may use further external Python 3 packages.

I will be using git for version control, and will have a public Github repository for code. Data and code will be backed up weekly to my external 1T Seagate Hard Drive. Should my laptop become unusable for any reason, I have an older version of the same laptop spare and should be able to get back to working on the project within a week if the situation arises. I require no other special resources.

## Further Project Substance

This project will be treated as a three-way classification problem. The models will predict one of the following three options per row: Home Win (H), Away Win (A), Draw (D). Each row will represent one English Premier League match. The data set will cover the

---

<sup>1</sup><http://publications.lib.chalmers.se/records/fulltext/250411/250411.pdf>

<sup>2</sup><http://www.football-data.co.uk/englandm.php>

seasons from 1992/93 to 2017/18 inclusive. Each season has 20 teams, each of which play every other team twice: one at home and one away. This equates to 380 matches per season, giving the data set almost 10000 rows.

Some potential individual features and sets of features include: Date, Home Team, Away Team, Result (target variable), Referee, Manager, Pitch Size, Form data (such as last 3 game win%), League Stats in current season (position in league of each team, win%, upset%, goals per game etc.), Injury Data, Last Season League Position, Transfer Window Spending, Player Data (such as total PL games experience of starting eleven, average win% of players, etc) and more yet to be decided upon.

Of course each bit of information used as a feature would need to be represented in numerical form in order to be useful to the model. I will investigate multiple ways of representing certain features. For example, say we are trying to represent the Referee column as numerical data. It could be represented as 'Referee upset%', i.e. % of games where this referee is present, and the underdog team wins, or it could be represented as 'Referee Home team win%', i.e % of games where this referee is present, and the specific current home team wins. These alongside other representations can be investigated.

I will start by collecting a minimal data set containing around 5 features but still using all 26 years of data, and then I will use some of the ML techniques above to evaluate how well they perform on it.

I first plan to try out Logistic Regression with this data set, as it is one of the simpler models to use. This should allow me to get a good grasp on building and evaluating the model, as well as giving me insight into effective features and test-train splits. A test-train split refers to the ratio used to split the data set into 2 smaller data sets; one used for training the model, and the other used for testing it.

I hope to get this first data set ready for testing before mid-November, as outlined in my Timetable. This should allow me to have a model built and evaluated by the end of Michaelmas term. My initial test-train split will be allocating around 6 years of data to testing (around 20% of the data), based on typical splits used in literature. However, this is a flexible ratio depending on further research and evaluation. Since my data is chronological I cannot use cross-validation to improve the reliability of my test results, which is something I must keep in mind. I will leave the current season data (2018/19) for further evaluation/testing and validation nearer the end of the project.

After this, I will investigate adding additional data columns/features to my data, and re-evaluate the algorithms (or move on to new algorithms), eliminating less-useful columns along the way. This will be an iterative process, allowing me to keep improving my data and models.

Some features will be pre-computed in order to take into account the dynamic nature of the data set (as all data is in chronological order), and the prediction model itself will be static. Furthermore, example dynamic features (also known as derived features) include any sort of 'Form' or 'History' feature. It may be the average win% over the last 7 games for the home team, or the win% at a certain stadium over the whole data set **so far** for the home team. This will allow my model to use past results and data to improve prediction accuracy, whilst also allowing me to investigate a wider range of potential features. Thus, a large portion of the work required for this project will be feature engineering.

## Work to be done

The project breaks down into the following sub-projects:

1. Research literature and discover some useful sources of data. Investigate techniques to be used.
2. Collect data from numerous sources and clean/merge data sets for analysis and for inputs to any system.
3. Analyse features using data analysis and simple ML models, to generate an improved data set.
4. Build a more advanced prediction system. This is a project where I wish to experiment with multiple techniques and systems, and therefore by design I am uncertain of exactly which ones I will be applying at any given time. This extra flexibility is a positive aspect. This allows me to learn and improve my predictor as the project goes on, without being tied down to an overly-specific plan.
5. Evaluate prediction system(s).
6. Iterate the ‘feature engineer – build model – evaluate – improve’ cycle.

## Success criteria

This is a notoriously difficult problem for algorithms to solve, and humans have always performed better. Famous expert pundits, Mark Lawrenson and Paul Merson, have demonstrated a 52%<sup>3</sup> accuracy over previous years in this exact problem: predicting the outcome (H,A,D) of each match. Therefore achieving a prediction accuracy rivalling this would be an extremely successful project. In light of this, my success criterion is to exceed a 40% prediction accuracy in this three-way classification problem.

## Possible extensions

One key extension would be to compare my prediction system against more metrics and evaluation techniques, that give a further insight into the effectiveness of the system. Some examples include:

- Evaluate the accuracy of the model by simulating the process of placing a bet on every prediction the final system makes for the current (2018/2019) English Premier League season, and determining if a net profit was made.
- Compare my predictor to a statistics-based predictor. By this I mean using a probabilistic predictor that makes predictions based on a specific probability distribution that was determined from the data. For example, we could generate some values such as ‘% chance of home team win’, and do the same for away win and draw,

---

<sup>3</sup><http://eightyfivepoints.blogspot.com/2017/07/how-are-lawrenson-and-merson-beating.html>

then draw predictions from this distribution. This would create a very insightful and detailed set of evaluations, as a particularly accurate statistical predictor would highlight an effective feature.

Some other extensions include:

- Could split the data into two or more groups based on the difference in teams playing, applying the prediction models to each separately. For example, I could split the data set into two sets: The first covering all games where the difference between the league position of the home team and the league position of the away team is less than 7, and the second containing the games where this difference is 7 or greater. This could potentially help my model predict more accurately, as it may learn different weights in each scenario. For example, in the set where the difference is large the model will learn to nearly always predict the team with the better league position, whereas in the other set it will most likely be more complex.
- Have a Deep Learning model built and evaluated. This is something not certainly in my plan, but will be something I do if it becomes clear it is the best option. Therefore it is a useful extension to note.
- Extend the system to predict cup games between Premier League games; these notoriously have a higher percentage chance of upset, and this would be a fascinating phenomenon to explore.
- Extend the prediction system to other leagues, such as the English Championship (the second division). This may cause extra complexity if my original data sources for the EPL don't have the same data for these other leagues.

## Timetable

Planned starting date is 18/10/2018.

1. **Michaelmas weeks 2–3: Oct 18 - Oct 24** Setup git code repository, and USB data backup. Research literature on sports prediction systems (both specific to football and the problem I am tackling, but also looking at more general problems and sports). Also research Python packages to use/ how to use them.
2. **Michaelmas weeks 3–4: Oct 25 - Oct 31** Determine good sources of data and collect this data via web scraping or whatever technique is most appropriate.
3. **Michaelmas weeks 4–6: Nov 01 - Nov 14** Clean and merge data, validating data from multiple sources along the way. Design initial data set in correct form to input into a predictor. Design first predictor (logistic regression). Begin potential feature research.
4. **Michaelmas weeks 6–8: Nov 15 - Nov 28** Train, test and evaluate the first ML system. Run a few evaluation techniques. Generate and collect data for some more features to add to data set, and decide upon next model to use. Begin building this next model.

5. **Christmas Vacation Part I: Nov 29 - Dec 20:** Continue building improved models and adding features in an iterative manner.
6. **Christmas Vacation Part II: Dec 20 - Jan 16:** Continue collecting more features, and experimenting with new techniques. By this point either have experimented with 3+ techniques, or have begun on a decided-upon larger system. Experiment with Deep Learning if time permits and I have not already.
7. **Lent weeks 0–2: Jan 17 - Jan 30** Narrow down on best system so far, for an in-depth evaluation. Run evaluation and write up a progress report.
8. **Lent weeks 2–4: Jan 31 - Feb 13** Work on next prediction system and data set depending on results and research.
9. **Lent weeks 4–6: Feb 14 - Feb 27** Continue working on this system and data set. Improve based on further feature analysis, evaluation results and research.
10. **Lent weeks 6–8: Feb 28 - Mar 13** Finish current version of system, run on latest data set and complete a full evaluation report. Begin writing dissertation.
11. **Easter Vacation Part II: Mar 14 - Mar 31** Experiment with improving system and/or looking at potential extensions, but mainly focus on dissertation.
12. **Easter Vacation Part II: Apr 01 - Apr 24** Complete draft dissertation, pass it to supervisor and DofS and tie up work.
13. **Easter term 0–2: Apr 25 - May 8** Finish up dissertation changes and additions.
14. **Easter term 3: May 9 Onwards** Proof reading and then an early submission so as to concentrate on examination revision.