Computer Science Extended Essay

Advances in Machine Learning: The Effectiveness of Supervised Learning in Sporting Events

Research Question: To what extent can supervised machine learning algorithms — artificial neural networks and support vector machines — be used to predict football matches in the Premier League?

Session: May 2021

Candidate number: jbt779

Word Count: 3999

Table of Contents

List of Figures

Chapter 1

Introduction

Arguably the most popular sport on the planet, football is a multi-billion industry with revenues only ever increasing. According to Deloitte, the revenues of the "Big Five" European football leagues have increased from €4.8 billion (2001) to €17 billion (2019) [1]. Football's growing popularity has led to a surge of new technologies introduced by statisticians and data scientists to give teams the edge against their opponents [3]. Among these are the use of machine learning (ML) algorithms to predict football match results, which are also used by gamblers [12].

Artificial intelligence (AI) is believed to be the future of technology. Examples include: self-driving cars, user-targeted advertisements, etc. ML, a subset of AI, is becoming progressively more popular. This paper aspires to utilize ML's popularity and accessibility in recent years to answer the question: "**To what extent can supervised ML algorithms — artificial neural networks and support vector machines — be used to predict football matches in the Premier League?**"

The concept of using statistical methods to predict sporting event results is not new, with one of the earliest statistical modelling methods coming from Moroney in 1956 [7]. In more modern times, the use of ML techniques has been widely explored in football. One of the first studies on artificial neural networks (ANNs or NNs) was conducted by Purucker (1996) [10] to forecast NFL games. One limitation found was that the study used limited features to train the NN. Kahn [5] then added to Purucker's studies in 2003 by increasing the number of parameters for the NN and acquired superior outcomes, confirming that NNs may be suitable for predicting sporting events. A more recent study by Hucaljuk in 2011 [2] tested 6 ML methods

to predict football scores, concluding that the NN achieved the best results. This study was particularly helpful in the composition of this paper as the nature of the study — comparing different algorithms on the same problem — is very similar.

As a school representative in football, I am constantly analyzing matches, trying to predict which team is likely to win, and considering factors that may forecast the outcome. For example, if Liverpool played Chelsea, it would be reasonable to consider the current league position of both teams, their form and perhaps some arbitrary metric like the market value of the team and many more. The number of factors are limitless. Some people believe that they are better at predicting matches than others; that how the betting industry came about. This paper aims to go beyond just finding which algorithm is the most accurate, it aims to evaluate ML algorithms in a real world application like gambling.

Compared to past research, this paper is both unique and significant in that it thoroughly compares two of the most suitable ML algorithms, support vector machines (SVMs) and NNs, for this particular application (sporting events), which includes in-depth analysis of the computational complexity of each algorithm, and its implications. Past research has failed to properly address the complexity of ML algorithms in terms of time and space, especially in the context of football. Conflicting opinions also made it rather difficult for other scholars to provide theoretical background on the computational complexities of ML algorithms.

Chapter 2

Theoretical Background

2.1 Machine Learning

An older definition of ML is that it is a "field of study that gives computers the ability to learn without the need of being explicitly programmed" [11]. However, in recent times, ML has developed and , put simply, is seen more as algorithms that can automatically detect patterns in data, be used to predict future data, or perform other kinds of decision-making under conditions of uncertainty [8].

2.2 Regression and Classification

This paper concerns supervised learning, where a labelled set of inputs (x) and desired outputs (y), represented by $T = \{(x_i, y_i)\}_{i=1}^{N}$, where T and N describes the training set and number of examples respectively, are given to ML algorithms to "learn" and predict outputs. Regression and classification are the only to types of supervised learning. Classification algorithms aim to learn how inputs (x) produce outputs (y), where $y \in 1, ..., C$ (number of classes). $C = 2$ describes a binary classification; $C > 2$ is a multiclass classification. Regression is no different from classification other than the output being continuous $y \in \mathbb{R}$. [8].

2.3 Classification Algorithms

There are only three outcomes (W, D, L) that result from a football match, hence $C = 3$, making this a multiclass problem.

2.3.1 Common Algorithms

There are many different classification ML algorithms. Selecting a model is considered an art due to the variety of functions that each algorithm is capable of. Below are some common classification algorithms.

Table 2.1: Summary of common classification algorithms

Algorithm	**Description**
NN	Based on neurons and implements backpropagation.
Naive Bayes	A probabilistic classifier derived from the Bayes theorem, under the assumption that attributes are conditionally independent.
SVM	Uses hyperplanes to predict outcomes.
Decision Trees	Produces a sequence of rules that can be used to classify the data.
k-NN	Classifies objects by a majority vote of the object's neighbours. The object is assigned to the class that is most common among its k nearest neighbour.
Random Forest	Constructs a large number of small decision trees at training time which produces their own prediction. The random forest model combines predictions of the small decision trees to produce predictions.

Out of these algorithms, the two most appropriate for this investigation's specific dataset are SVMs and NNs, due to the following reasons:

- For classification problems in which there is no domain knowledge on the problem, SVMs generally performs better than other algorithms.
- SVMs are effective in high-dimensional spaces. Given that our dataset is 30-dimensional, SVM is an ideal choice.

- NNs are deemed to be the "future" of ML, where their applications range from self-driving cars to disease detection.
- NNs can solve non-linear problems because of activation functions. SVMs can also solve non-linear problems using the rbf kernel.

Although both ML algorithms have their strengths and weaknesses, they are completely different mathematically and conceptually. Most data scientists prefer SVMs because they can mathematically describe the model that has been trained. NNs, however, determine patterns in a way that is unknown to humans.

2.3.2 Support Vector Machines

SVMs are thought to be one of the most accurate off-the-shelf ML algorithms. The objective of SVMs is to find a hyperplane in an N-dimensional space that classifies data points.

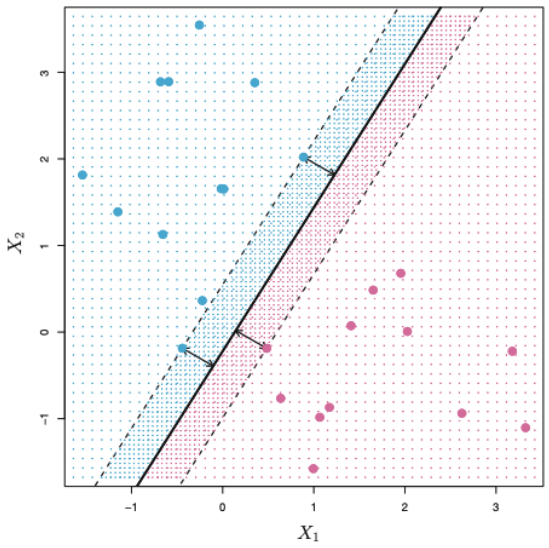

Figure 2.1: 2D linearly separable SVM [4]

The decision boundary of a linear SVM is determined by the maximum margin or hyperplane, which is characterized as the function that has the biggest separation to the closest training data points of any class, since, generally, the bigger the margin the smaller the error [9]. The nearest data points to the hyperplane are called support vectors (SV) because they "support" the algorithm by determining the maximum

margin. In other words, even if the inputs to the left or right of the SV is changed, as far as the two SVs stay the same, the maximum margin will remain the same. In Figure 2.1, there are 3 SVs that are on the margin boundaries, known formally as the Maximum Margin Hyperplane.

Unfortunately, many datasets are not linearly separable (non-linear); thus a linear SVM cannot be used to predict those datasets, which is what kernelized SVMs aim to solve [6]. What was explained in this section is the most basic and simplified version of SVM. Kernelized SVMs will be used in this paper, they utilize a mathematical method called the kernel trick, however, explaining this concept is mathematically challenging and is beyond the scope of computer science and this paper.

2.3.3 Neural Networks

Loosely based on the human brain (Figure 2.2), NNs refer to non-linear models that involve combinations of matrix multiplications and non-linear operations.

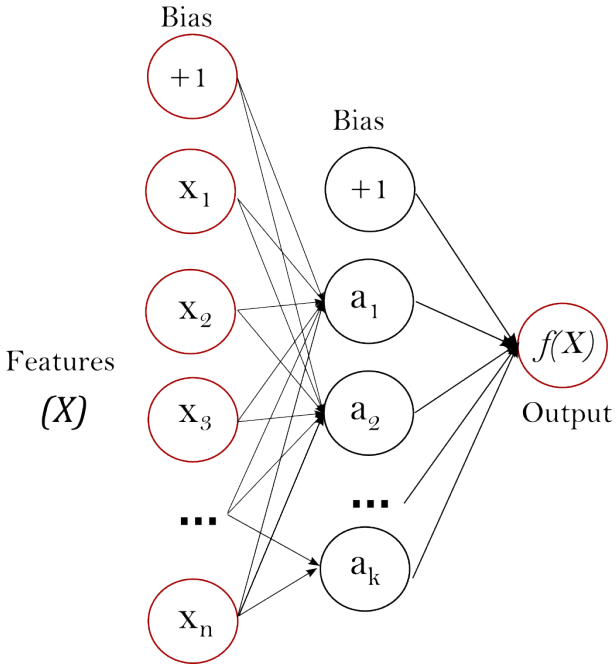

Figure 2.2: Neural network diagram [9]

NNs are comprised of a few layers of interconnected nodes that transmits information from an input layer to an output layer. The entire model is depicted by a sequence of matrix functions (2.1) where data is:

1. Added together to create a sum
2. Multiplied by a weight
3. Added to a bias number
4. Mapped into the activation function

This cycle is known as the feedforward cycle and permits NNs to perform logical decisions like XOR or AND logic gates. Since the nodes in each layer of the NN can be learned, NNs can discover patterns to classify data.

$$\overbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_i \end{bmatrix}}^{output} = \varphi \left(\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_j \end{bmatrix}}_{input} \times \underbrace{\begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix}}_{weights} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \end{bmatrix}}_{biases} \right) \tag{2.1}$$

NNs, depending on the model's complexity, can take longer to train compared to other techniques, due to their learning technique of back propagation.

Extensive knowledge of football and football rules are not required to understand the contents of this paper and will therefore not be mentioned.

Chapter 3

Methodology

Before conducting the investigation, many procedures were carried out to achieve high predictive accuracy. This section discusses the practical steps that were implemented.

3.1 Procedure

1. Prepare dataset by means of data preprocessing (Appendix A).
2. Intuitively select suitable features.
3. Perform GridSearch using sklearn library to find the best hyperparameters.
4. Construct SVM using sklearn library; construct NN using tensorflow and keras library.
5. Perform Cross-Validation.
6. Calculate and plot evaluation metrics (All plots from this point on was generated with the matplotlib python library).

3.2 Dataset

The dataset used was taken from (http://www.football-data.co.uk), which contains match statistics for the Premier League since the 1993/1994 season. For this experiment, only data collected from the 2000-01 season to the 2018-2019 season was used,

because older statistics contained missing data and were in different formats. The dataset contains a total of 65 columns with the following column headings:

Command	**Description**
Date	Match Date (dd/mm/yy)
HomeTeam	Home Team
AwayTeam	Away Team
FTHG/FTAG	Full Time Home/Away Team Goals
FTR	Full Time Result (H=Home Win, D=Draw, A=Away Win)
HTHG/HTAG	Half Time Home/Away Team Goals
HTR	Half Time Result (H=Home Win, D=Draw, A=Away Win)
Referee	Match Referee
HS/AS	Home/Away Team Shots
HST/AST	Home/Away Team Shots on Target
HF/AF	Home/Away Team Fouls Committed
HC/AC	Home/Away Team Corners
HY/AY	Home/Away Team Yellow Cards
HR/AR	Home/Away Team Red Cards

The other 41 columns consists of betting odds from various betting websites

A sample of the dataset:

Table 3.1: Dataset for 2015-16 season. Contains 380 matches and 65 columns

Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR
E0	08/08/15	Bournemouth	Aston Villa	0	1	A
E0	08/08/15	Chelsea	Swansea	2	2	D
E0	08/08/15	Everton	Watford	2	2	D

3.3 Data Preprocessing

Dealing with 65 columns of data is not ideal, especially when some data, like referee names, are irrelevant. That is why data preprocessing is paramount. After using intuition and experimenting with different parameters, the final dataset was narrowed down to 30 columns with the following headings:

Command	Description
HTP/ATP	Home/Away Team Points
HTGD/ATGD	Home/Away Team Goal Difference
DF	Difference In Points
DLP	Difference In Last Year's Position

The other 25 columns consist of the results of the last three matches for both teams.

After processing the data, a statistical technique called t-SNE was used to reduce the dimensionality of the dataset from 30 dimensions into something that can be visualized (2D or 3D). The t-SNE Python library function was used to plot the graph below:

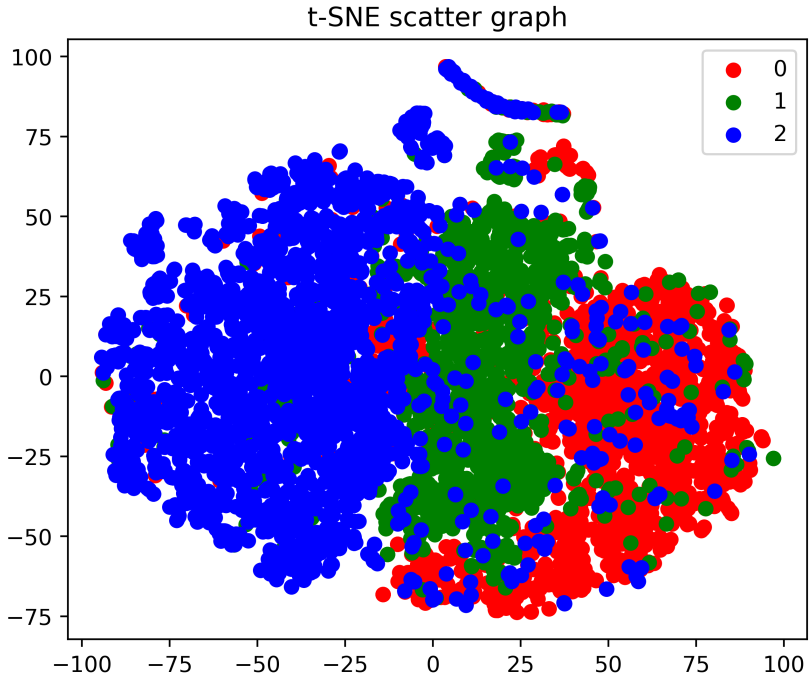

Figure 3.1: t-SNE graph reducing 30 dimensions into 2 dimensions

The t-SNE diagram suggests a non-linear relationship between the variables, which is what SVMs and NNs are suited for. The graph also visualizes the three classes and shows distinguishability, however there are overlapping data points, which may be a concern during prediction.

3.4 Controlled Variables

All experiments require controlled variables that do not change throughout the course of testing, below is a table that shows these variables.

Variable	**Description**	**Specification**
Computer and operating system	A MacBook Pro laptop will be used to run the program	Version: 10.15.4 Processor: 2.7 GHz Dual-Core i5 Memory: 8 GB 1867 MHz DDR3
Programming language	The Python programming language will be used	Version: 3.7
Integrated development environment	The Spyder scientific Python IDE will be used	Version: 4.1.3
Dataset	Dataset for both algorithms will be the same	Features: 30 Number sets: 7220

3.5 GridSearch

Other than parameters that are fed into a model, some ML algorithms also require hyperparameters that are predetermined by the user. This is true for NNs and SVMs. By using the scikit-learn library, a GridSearch algorithm was implemented to find the combination of hyperparameters that produced the best results. Below is a table of the values that were tested with GridSearch.

Neural Network	
Batch Size	10, 20, 30, 32, 40
Epochs	10, 25, 50, 100, 150
First layer neuron	10, 25, 50, 100, 150
Second layer neuron	10, 25, 50, 100, 150
Support Vector Machine	
Kernel	rbf
Gamma	1, 0.1, 0.01, 0.001, 0.0001
C	0.1, 1, 10, 100, 1000

The NN GridSearch ran for 5 hours and 20 minutes. The GridSearch algorithm tested each possible case to determine the highest accuracy, which means that the NN model ran $5^4 = 625$ times. The SVM GridSearch, however, is only limited to 3 hyperparameters. The dataset is non-linear, hence only the rbf kernel is tested, leaving only $5^2 = 25$ combinations.

Neural Network				
Accuracy (%)	Batch Size	Epochs	First Layer Neuron	Second Layer Neuron
91.33	30	25	50	100
91.29	10	10	50	100
91.27	40	25	25	25
91.25	32	10	25	50
91.23	40	100	25	25
Support Vector Machine				
Accuracy (%)	Kernel	C	Gamma	
91.14	rbf	100	0.01	
91.11	rbf	1	0.1	
90.96	rbf	10	0.01	
90.90	rbf	1000	0.001	
90.90	rbf	1	0.01	

The above table shows the top 5 accuracy scores of the GridSearch. Looking at the

results of the GridSearch, it would be plausible to simply select hyperparameters from the highest accuracy. However, a better method would be to select the most frequent hyperparameter in the top 5; this will ensure that the selected hyperparameter is consistently accurate.

3.6 Algorithm Architecture

Using the results from the GridSearch, below is a table highlighting the hyperparameters and the values that was used in this investigation.

Neural Network	
Batch Size	40
Epochs	25
First Layer Neuron	25
Second Layer Neuron	25
Support Vector Machine	
Kernel	rbf
Gamma	0.01
C	100

The layers in the NN GridSearch refer to the hidden layer; the input and output layers remain the same. This results in a NN with 83 neurons and 1,503 weights (including bias nodes).

Chapter 4

Data Presentation

4.1 Cross-Validation Scores

Cross-validation is a statistical method commonly used by data scientists to test how well a model is able to be trained by a given dataset and then predict data that it has not seen. This method of evaluation reduces the chance of over-fitting. In this experiment, a K-Fold cross-validation, where $K = 5$, was used. A common practice is to have $K = 10$, however, to keep the program computationally inexpensive, a smaller number was chosen. A K value of 5 means that 20% of the dataset would be used for testing while the other 80% is used for training. The cross_val_score function from the sklearn library ensures that the dataset is split in the same way, allowing comparisons between the NN and SVM.

Cross-Validation	**Neural Network (%)**	**Support Vector Machine (%)**
1	87.95	88.09
2	88.57	88.50
3	89.20	89.34
4	94.11	94.39
5	95.29	95.22
Mean	91.02	91.11

The above table displays the 5 results that came from the cross-validation. Oddly, the accuracy seems to follow a positive linear relationship with the cross-validation, suggesting that the model became better at predicting over time. However, it is

known that cross-validation creates a new model every time, which is discarded after testing is done, hence the increase in accuracy is likely a coincidence. Variations in the accuracy could be due to some test subsets being harder to predict due to a high number of outliers.

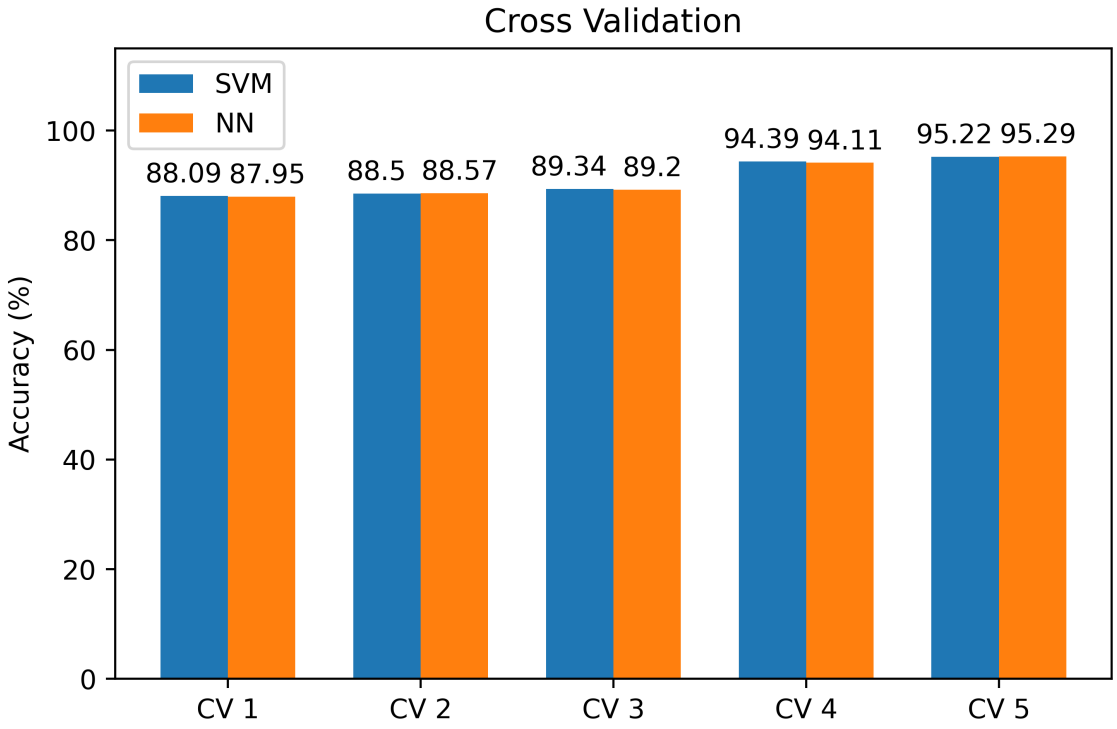

Figure 4.1: Cross-validation bar chart

Looking at the bar chart, the NN performed extremely similarly to the SVM. The mean accuracy for the SVM, however, is 0.09% higher than that of the NN, which is neglectable. What is surprising is how high the accuracy of both models is. 91% accuracy for almost any ML model is remarkable.

4.2 Performance Graphs

Performance graphs in ML are used to provide more information about the model. For NNs, it is typical to plot epoch against accuracy or loss to determine whether a model is over-fitting or under-fitting. For SVMs, models are not trained by epochs and the process of learning is not obvious; however, by splitting the dataset, we can

graph the accuracy and derive a learning curve that will show whether too much data is used and if the model is over-fitting.

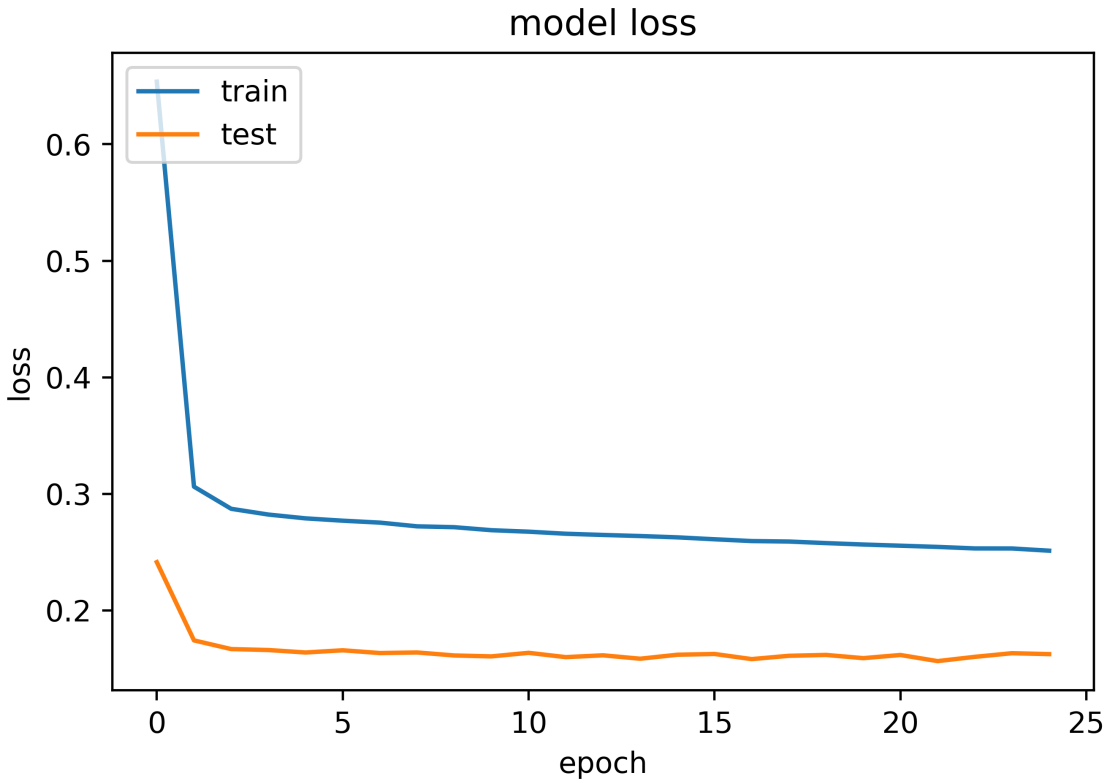

Figure 4.2: Neural Network model loss graph

The model loss graph can be a "diagnostic" to evaluate the performance of NNs. The graph above shows a steep fall in the first two epochs followed by a plateau all the way to the 25th epoch. Another important observation is that test loss is significantly lower than that of train loss. This could be an indication that the test set is easier to predict than the training set.

In terms of fit, the graph does not display indications of underfitting (decrease in both the test and training set). The graph also does not suggest an overfit, as the test set does not increase in loss over time, which leaves the model to be categorized as a good fit. Unfortunately, the model does not possess all the features of a good fit, but it is still acceptable.

The model accuracy curve draws similar conclusions to the model loss curve. The gap between the test and train dataset suggests an "easy" test set. This could be

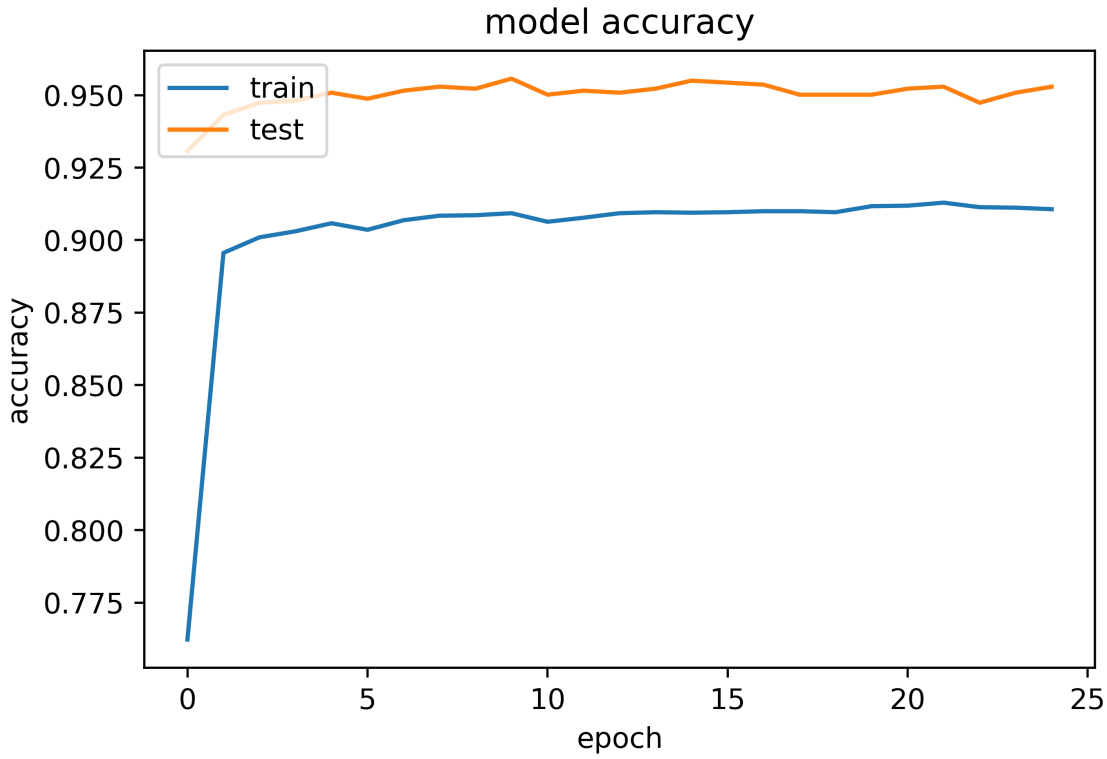

Figure 4.3: Neural Network model accuracy graph

solved by using a cross-validation. However, since the purpose of the diagnosis is to determine whether the NN has a good fit, using cross-validation (which is also computationally heavy) is not necessary.

The learning curve for the SVM is very different from the NN curve since it plots training set size as opposed to epoch. Fundamentally, the graphs display different information. SVMs do not learn from epochs, therefore it is difficult to visualise the learning that occurs during a single training. What can be done, however, is to run the SVM using different dataset sizes to analyze how it affects accuracy.

The graph above shows a steep increase with a smaller dataset, but quickly stabilizes after a few hundred datapoints. Both the training score and the cross-validation score gradually increase as the dataset gets larger. The conclusion can be made that there is sufficient data for the SVM to make relatively good predictions. Additionally, the gradual increase indicates that there is not too much data for the model to handle.

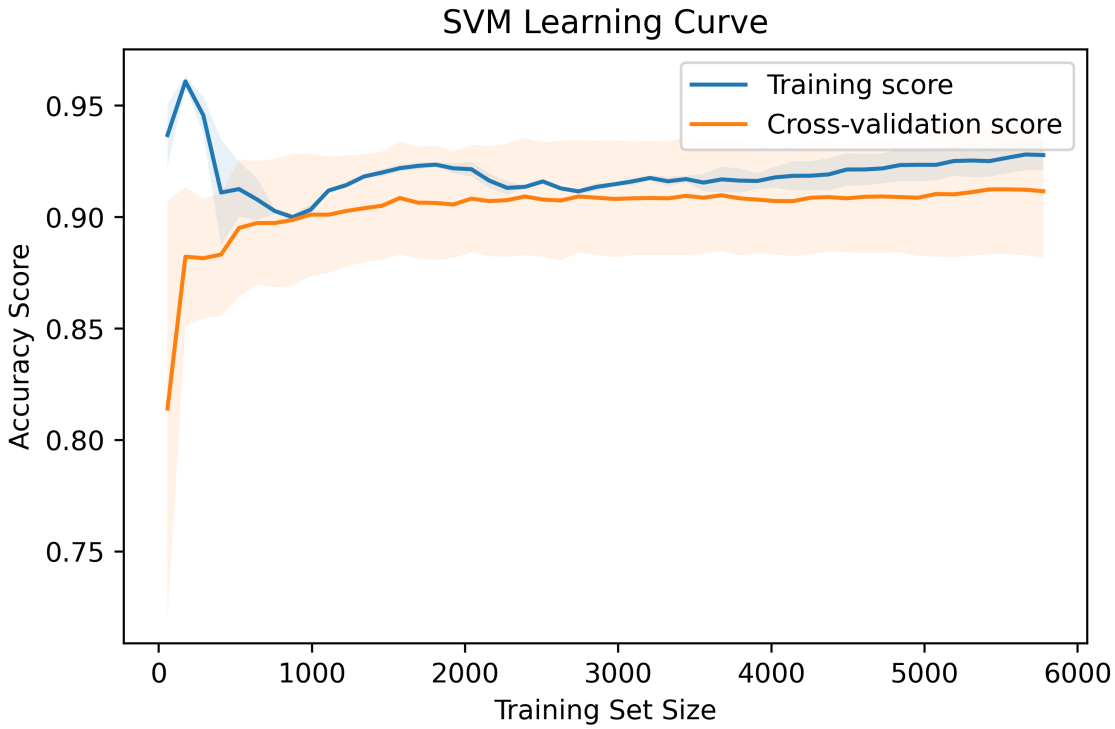

Figure 4.4: Support Vector Machine learning curve

4.3 Confusion Matrix

After analysing the learning curves, the actual predictions of the model can be investigated, as the accuracy score alone is usually not enough to distinguish the two ML algorithms. However, in this investigation, since the accuracies are so high, the confusion matrix will only be lightly analysed. Below are the two matrices:

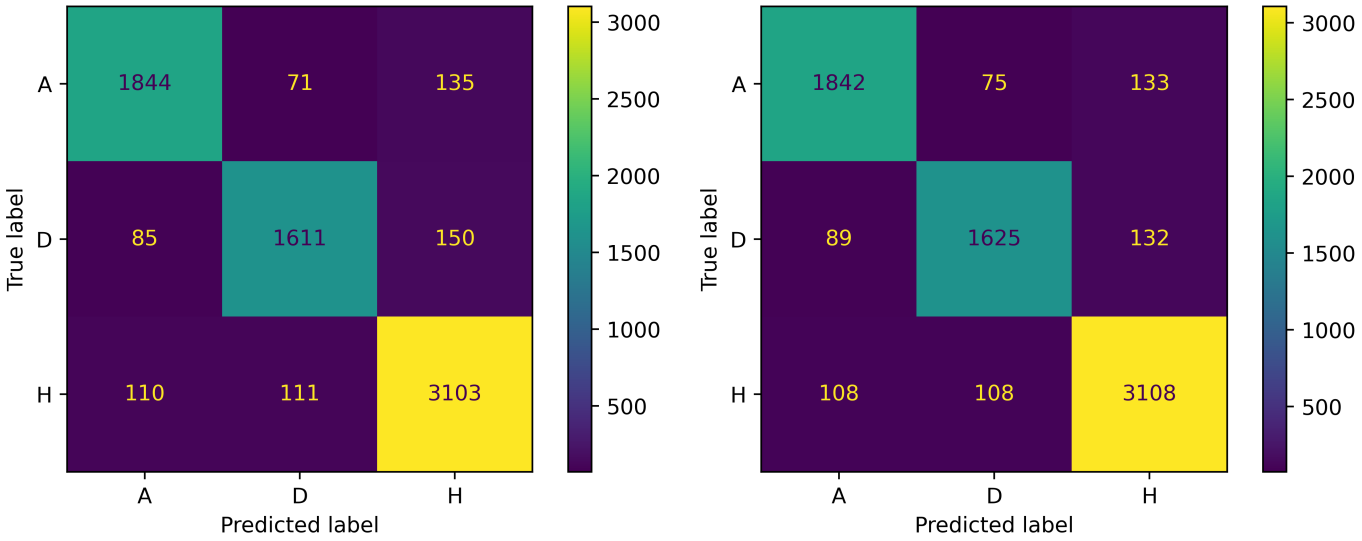

Figure 4.5: Confusion matrix for Support Vector Machine (Right) and Neural Network (Left)

The above matrices includes the predictions for all the 5 K-Fold cross-validation sets (total number of predictions is 7220). The diagonal sections highlighted in green and yellow represent the values that were predicted correctly. Thus, a model should aim to have a high percentage of predictions fall into those diagonal sections of the confusion matrix. As expected, the differences between the two are very low.

The main purpose of a confusion matrix is to visualise the results of a model, but since the two models performed similarly, the confusion matrix alone can't draw any meaningful conclusions. Additionally, since the accuracies are so similar, analysis on the classification report and receiver operating characteristic (RCO) would also be meaningless as they certainly would bear similar results.

4.4 Computational Complexity

In computer science, big o notation describes the performance and complexity of an algorithm by depicting the worst-case scenario of an algorithm and is used to represent execution time or space, in terms of memory.

4.4.1 Time Complexity

Other than accuracy, it is also paramount to analyse the efficiency of any algorithm. For example, it wouldn't be very helpful to staticians if their 100% accurate model took a few weeks to train or predict. A real-life application of ML algorithms are live score predictions, which rely heavily on time efficiency to help betters win money and predict outcomes in a matter of seconds. Any delay could be costly. A simple metric for analysing efficiency would be to programmatically compute the total time taken for the two models to train and predict the dataset. On Python, this is done with the use of the "time" library and the programming concept of sequencing.

Although plausible, there are multiple issues with this approach. Firstly, the calculated time would only reveal the more efficient algorithm for one particular case. A more impartial approach would be to find the time efficiency of different dataset sizes. Secondly, the nature of how NN trains its dataset (forward and backward propagation) requires its user to determine the number of epochs. Theoretically, the higher the number of epochs, the better the NN can train its data. However, too many epochs can cause over-fitting. SVMs, however, do not contain many hyper-parameters and therefore the training method is unchanged. Therefore, based on the number of epochs the user defines, training time will change accordingly. This in turn creates a trade-off between accuracy and efficiency. In a real life situation, trying to balance efficiency and accuracy would be a difficult task. Nevertheless, in this investigation, accuracy is the priority, hence the GridSearch determined an epoch of 25.

To work around the epoch issue, the time complexity of the ML algorithms can give a good depiction of efficiency without having to worry about the number of epochs or the batch size. Time complexity for a NN depends on the architecture of the algorithm. For an untrained Multi layer Perceptron (MLP), the structure used in this investigation, evaluating a single pattern requires the model to process all the weights and neurons. Given that each neuron is associated with at least one weight, we can ignore the number of neurons, leaving us with the time complexity of $O(w)$,

where w is the number of weights. Back propagation has the same complexity as forward evaluation. This leaves us with the time complexity $O(w \cdot e \cdot m \cdot)$.

The time complexity can be tested using the football dataset and plotting training set size against time taken. The dataset would have to be split into gradually increasing lengths. The split dataset would then be used to train the model and the time taken would be recorded. The results of 100 datasets with varying sizes are shown below:

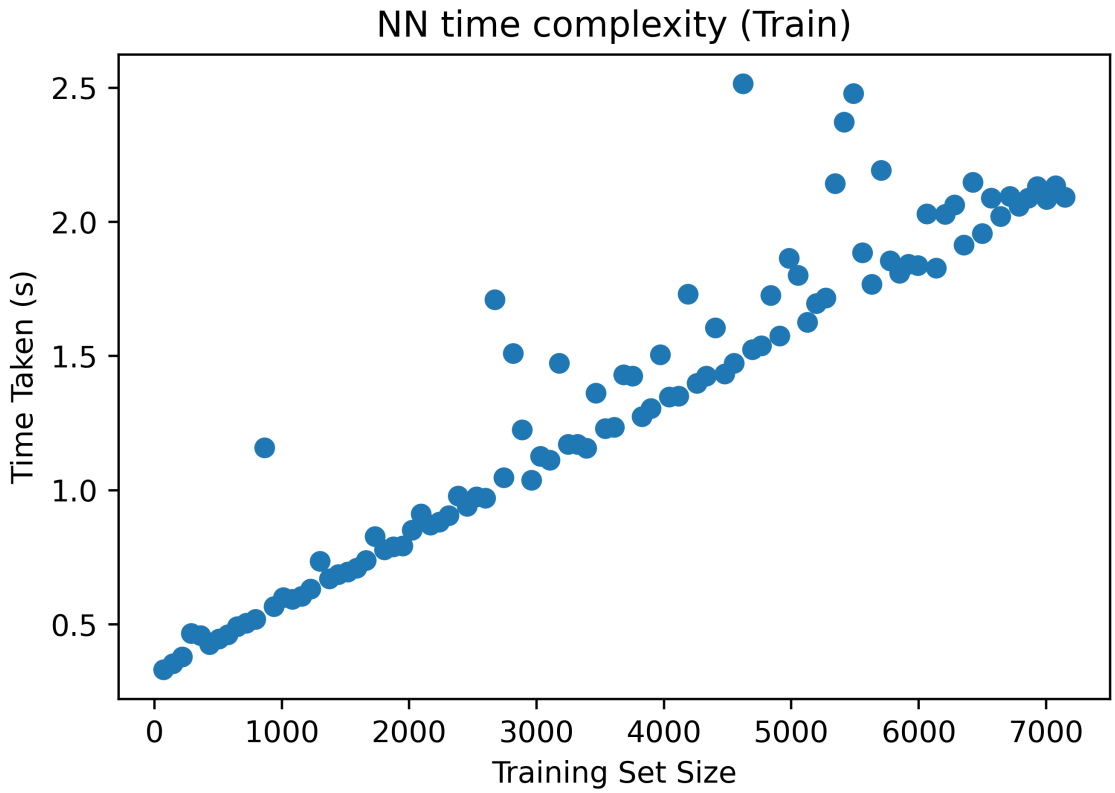

Figure 4.6: Time complexity curve for Neural Network (Train)

Although there are some anomalies, the relationship between the training set size and the time taken for training is positively linear where $R^2 = 0.88$, suggesting a strong correlation. Plotting a trend line seems to intercept the origin, suggesting that training set size is directly proportionate to time taken.

Time complexity for SVMs is highly debated, however, it is thought that it is between $O(n^2)$ and $O(n^3)$. Generally, most kernel methods have a time complexity of $O(n^2)$. After evaluating the time complexity for the NN, the same method can be used to find the relationship for the SVM.

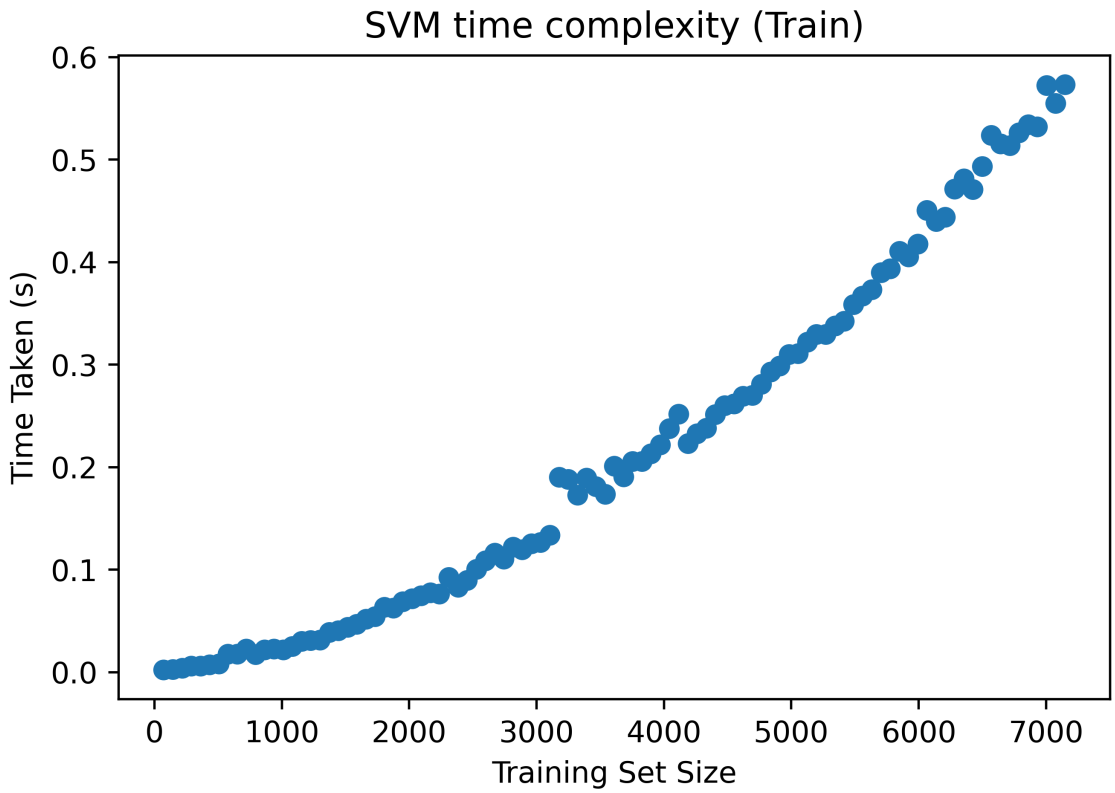

Figure 4.7: Time complexity curve for Support Vector Machine (train)

Unlike the NN, the graph above suggests that for the SVM, the train set size has a quadratic relationship with the time taken. The results are in line with the big O notation previously mentioned.

With both time complexities for training evaluated, comparisons can be made to determine which model is more efficient. Keeping in mind that the actual dataset size is around 7000, purely looking at the training time for the whole dataset would suggest that the SVM is more time efficient, since the SVM took 0.6 seconds, whereas the NN took over 2 seconds. However, from the big O notation, it is known that the time complexity for SVMs is $O(n^2)$ while the NN has a linear relationship. This would mean that for datasets that are larger, the SVM would eventually surpass the NN in time taken, making it less efficient.

4.4.2 Space Complexity

As with time complexity, some theories behind space complexity is still debated by computer scientists. This is my take: In a NN, the only variables stored are the weights of each layer. Essentially, all the NN is doing is changing the weights ever so slightly. Hence, if weights are the only variables stored, the space complexity is $O(w)$, where w is weights.

Similar to NNs, the only variables that need to be stored are the support vectors, because they are the points that determine the hyper-plane, which evaluates to $O(k)$, where k is the number of support vectors.

Both models are thought to have a linear increase. Space complexities will not be tested in this investigation as there is a lack of documentation and resources concerning the best method to measure and track memory usage for a particular function.

Chapter 5

Limitations of the Investigation

Although this investigation achieved its objectives, there are many limitations of the investigation, which include the following:

1. The training dataset was relatively small (7440 matches) for a NN, as they require large datasets. However, this may increase the training time for both models, especially for the SVM, where $O(n^2)$.

2. For simplicity's sake, this investigation looked at classification algorithms for predicting matches. A more comprehensive model would be a regression model to predict the numerical score. This would certainly lead to lower accuracy.

3. Only two algorithms were investigated due to the belief that they were the most suitable. However, in a larger scaled project more algorithms should be tested.

4. Although hyperparameter optimization was conducted using GridSearch, feature analysis was not made to search for the best features to use; instead, intuition was used to approximate features that may have an impact. In a larger project, softwares such as OpenMOLE could be used to determine features with the most impact.

Chapter 6

Conclusion

This investigation sought to find the **extent to which supervised ML algorithms can predict the outcome of Premier League football matches**. After a rigorous process of experimentation and analysis, it can be concluded that SVMs and NNs are both very capable of predicting football results at 91% accuracy, albeit only able to predict multiclass outcomes.

Comparing the two models is slightly more complicated and answering question "Which is better?" has no definitive answer. To begin with, both had very similar accuracy scores with the SVM coming ahead at just 0.08%. In terms of predictability, the two models are almost identical. Unfortunately, that is not the full picture, as NNs are known for performing better with larger datasets. This investigation is only limited to 7220 match statistics and therefore cannot test this hypothesis.

Other than accuracy, computational complexity was also evaluated for both the models. Simply looking at training time is not a good picture of time complexity as NNs train in batches and epochs that can be tuned. Big O notation was used instead to conclude that, for training, time taken for the NNs increased linearly: $O(w * m * e)$, while the SVMs increased quadratically: $O(n^2)$. Therefore, depending on the dataset set size, one of the models would be more efficient than the other: SVMs for smaller datasets and NNs for larger datasets. Analysis of space complexity could not be properly tested, however, it is believed that both models have a linear space complexity. All in all, neither one is better than the other as they each have their unique strengths and weaknesses for different problems.

The use of ML to predict sporting events results is likely to become over-saturated in the coming years. An expansion of ML in sports for the future could be the use of convoluted neural networks to predict scoring chances from a video. An even more robust and ambitious use of ML is computer vision powered referees and coaches, where ML algorithms can predict which strategy is best and which lineup the opponent might use. Its applications are boundless.

References

[1] Deloitte, *Annual review of football finance 2020*, 2020. [Online]. Available: https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance.html (visited on 21st Feb. 2020) (cit. on p. 1).

[2] J. Hucaljuk and A. Rakipovic, 'Predicting football scores using machine learning techniques.', Jan. 2011. (visited on 2nd Oct. 2020) (cit. on p. 1).

[3] A. insight, 'How ai and machine learning are developing smarter football coaching', 2017. [Online]. Available: https://www.analyticsinsight.net/how-ai-and-machine-learning-are-developing-smarter-football-coaching (visited on 29th May 2020) (cit. on p. 1).

[4] G. James, D. Witten, T. Hastie and R. Tibshirani, *An introduction to statistical learning: with applications in R*. Springer, 2017. (visited on 17th Aug. 2020) (cit. on p. 5).

[5] J. Kahn, 'Neural network prediction of nfl football games', *World Wide Web Electronic Publication*, Jan. 2003. (visited on 22nd Aug. 2020) (cit. on p. 1).

[6] E. Kirill, d. Hadelin, T. SuperDataScience and S. SuperDataScience, *Machine learning a-z^{TM}: Hands-on python r in data science*, 2020. [Online]. Available: https://www.udemy.com/course/machinelearning/ (visited on 22nd Mar. 2020) (cit. on p. 6).

[7] M. Moroney, *Facts from Figures*, ser. A Pelican original. Penguin Books, 1956. [Online]. Available: https://books.google.com.gi/books?id=%5C_0ErswEACAAJ (visited on 17th Jul. 2020) (cit. on p. 1).

[8] K. P. Murphy, *MACHINE LEARNING: a probabilistic perspective*. MIT Press, 2013. (visited on 20th Feb. 2020) (cit. on p. 3).

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. (visited on 12th Apr. 2020) (cit. on pp. 5, 7).

[10] M. Purucker, 'Neural network quarterbacking', *IEEE Potentials*, vol. 15, pp. 9–15, 1996. (visited on 2nd Mar. 2020) (cit. on p. 1).

[11] A. L. Samuel, 'Some studies in machine learning using the game of checkers', *IBM Journal of Research and Development*, vol. 44, pp. 206–226, 1959. (visited on 16th Sep. 2020) (cit. on p. 3).

[12] J. Vincent, 'This startup is building ai to bet on soccer games', 2017. [Online]. Available: `https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance.html` (visited on 19th May 2020) (cit. on p. 1).

Appendices

6.1 Appendix A

```
import numpy as np
import sklearn
import time
import random
from sklearn import metrics
from sklearn import svm
from sklearn.neural_network import MLPClassifier
import pandas as pd

# Read data and drop redundant column.
data = pd.read_csv('EPLfinal_dataset.csv')
data = data.filter(['FTR','HTP','ATP','HM1','HM2','HM3','AM1','AM2',
    'AM3','HTGD','ATGD','DiffFormPts','DiffLP'], axis=1)

# Preview data.

#Full Time Result (H=Home Win, D=Draw, A=Away Win)
#HTGD - Home team goal difference
#ATGD - away team goal difference
#HTP - Home team points
#ATP - Away team points
#DiffFormPts Diff in points
#DiffLP - Differnece in last years prediction

# Separate into feature set and target variable
#FTR = Full Time Result (H=Home Win, D=Draw, A=Away Win)
X_all = data.drop(['FTR'],1)
```

```
y_all = data['FTR']

# Standardising the data.
from sklearn.preprocessing import scale

#Center to the mean and component wise scale to unit variance.
cols = [['HTGD','ATGD','HTP','ATP','DiffLP']]
for col in cols:
    X_all[col] = scale(X_all[col])

#last 3 wins for both sides
X_all.HM1 = X_all.HM1.astype('str')
X_all.HM2 = X_all.HM2.astype('str')
X_all.HM3 = X_all.HM3.astype('str')
X_all.AM1 = X_all.AM1.astype('str')
X_all.AM2 = X_all.AM2.astype('str')
X_all.AM3 = X_all.AM3.astype('str')

#we want continous vars that are integers for our input data, so lets remove any categorical vars
def preprocess_features(X):
    ''' Preprocesses the football data and converts catagorical variables into dummy variables. '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output
```

```
63
64 X_all = preprocess_features(X_all)
65 print("Processed feature columns ({} total features):\n{}".format(
       len(X_all.columns), list(X_all.columns)))
66
67 ######################################################################
68
69 def SVM_classifier(train_X,train_Y,test_X,test_Y):
70     clf = svm.SVC()
71     clf.fit(train_X, train_Y)
72     trainAccuracy = clf.score(train_X,train_Y)
73     y_pred = clf.predict(test_X)
74     accuracy = metrics.accuracy_score(test_Y,y_pred)
75     return accuracy,trainAccuracy,clf
76
77 def NeuralNets_classifier(train_X,train_Y,test_X,test_Y):
78     clf = MLPClassifier(alpha=1, learning_rate_init=0.0001,max_iter
       =100,)
79     clf.fit(train_X, train_Y)
80     trainAccuracy = clf.score(train_X,train_Y)
81     y_pred = clf.predict(test_X)
82     accuracy = metrics.accuracy_score(test_Y,y_pred)
83     return accuracy,trainAccuracy,clf
84
85
86 # Cross Validation # 80% 20%
87 from sklearn.model_selection import train_test_split
88
89 from sklearn.preprocessing import LabelEncoder
90 le = LabelEncoder()
91 y_all = le.fit_transform(y_all)
92
93 # train_X, test_X,train_Y, test_Y = train_test_split(X_all,y_all,
       test_size=0.2)
94
95
96
97 import tensorflow as tf
```

```
import matplotlib.pyplot as matplot


nn_keras_model = tf.keras.models.Sequential()
nn_keras_model.add(tf.keras.layers.Dense(units=100, input_dim = 30,
    activation='relu'))
nn_keras_model.add(tf.keras.layers.Dense(units=100, activation='relu
    '))
nn_keras_model.add(tf.keras.layers.Dense(units=1, activation='
    sigmoid'))
nn_keras_model.compile(optimizer = 'adam', loss = '
    binary_crossentropy', metrics = ['accuracy'])

nn_history = nn_keras_model.fit(X_all, y_all, validation_split=0.20,
     batch_size = 32, epochs = 100)

# Make a list of all the data in history
print(nn_history.history.keys())
# Summarize the history for accuracy
matplot.plot(nn_history.history['accuracy'])
matplot.plot(nn_history.history['val_accuracy'])
matplot.title('model accuracy')
matplot.xlabel('epoch')
matplot.ylabel('accuracy')
matplot.legend(['train', 'test'], loc='upper left')
matplot.show()
# Summary of loss history
matplot.plot(nn_history.history['loss'])
matplot.title('model loss')
matplot.plot(nn_history.history['val_loss'])
matplot.xlabel('epoch')
matplot.ylabel('loss')
matplot.legend(['train', 'test'], loc='upper left')
matplot.show()


train_X, test_X,train_Y, test_Y = train_test_split(X_all,y_all,
    test_size=0.2)
```

```
130
131 y_pred = nn_keras_model.predict_classes(test_X)
132
133
134 import matplotlib.pyplot as matplot
135 from sklearn.metrics import confusion_matrix, accuracy_score,
    plot_confusion_matrix
136 from sklearn.metrics import ConfusionMatrixDisplay
137
138 cm = confusion_matrix(test_Y, y_pred)
139 print(cm)
140
141 cm_display = ConfusionMatrixDisplay(cm, display_labels=["H","NH"]).
    plot()
142
143 score = accuracy_score(test_Y, y_pred)
144 print(score)
145
146
147
148
149 accuracy_SVM = []
150 train_SVM = []
151 for i in range(3):
152     testAccuracy4, trainAccuracy4, clf = SVM_classifier(train_X,
    train_Y,test_X,test_Y)
153     accuracy_SVM.append(testAccuracy4)
154     train_SVM.append(trainAccuracy4)
155
156 print("Train for svm:",np.mean(train_SVM))
157 print("Accuracy for svm:",np.mean(accuracy_SVM))
158
159 from sklearn.model_selection import cross_val_score, StratifiedKFold
160 accuracies = cross_val_score(estimator = clf, X = train_X, y =
    train_Y, cv = 10)
161
162 def annK():
163     nn_keras_model = tf.keras.models.Sequential()
```

```
164     nn_keras_model.add(tf.keras.layers.Dense(units=100, input_dim =
        30, activation='relu'))
165     nn_keras_model.add(tf.keras.layers.Dense(units=100, activation='
        relu'))
166     nn_keras_model.add(tf.keras.layers.Dense(units=1, activation='
        sigmoid'))
167     nn_keras_model.compile(optimizer = 'adam', loss = '
        binary_crossentropy', metrics = ['accuracy'])
168     return nn_keras_model
169
170 from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
171 estimator= KerasClassifier(annK, epochs=100, batch_size=32, verbose
        =1)
172 estimator._estimator_type = "classifier"
173
174 results = cross_val_score(estimator = estimator, X = train_X, y =
        train_Y, cv = 10)
175
176
177 print(accuracies.mean())
178 print(results.mean())
179
180
181
182 start_time = time.time()
183 estimator.fit(train_X, train_Y)
184 nn_pred = estimator.predict(test_X)
185 time_1 = time.time()
186 print("NN took " + str(time_1 - start_time) + "seconds")
187
188
189
190 time_2 = time.time()
191 clf.fit(train_X, train_Y)
192 svm_pred = clf.predict(test_X)
193 time_3 = time.time()
194 print("SVM took " + str(time_3 - time_2) + "seconds")
195
```

```
196 from sklearn.metrics import classification_report
197 from sklearn.metrics import plot_roc_curve
198
199 print(classification_report(test_Y, nn_pred))
200 print(classification_report(test_Y, svm_pred))
201
202
203 disp = plot_confusion_matrix(clf, test_X, test_Y, display_labels=["H
    ","NH"])
204 disp = plot_confusion_matrix(estimator, test_X, test_Y,
    display_labels=["H","NH"])
205
206 nn_cm = confusion_matrix(test_Y, nn_pred)
207 svm_cm = confusion_matrix(test_Y, svm_pred)
208 cm_display = ConfusionMatrixDisplay(nn_cm, display_labels=["H","NH"
    ]).plot()
209
210
211 print(nn_cm)
212 print(svm_cm)
213
214
215 svc_disp = plot_roc_curve(estimator, test_X, test_Y)
216 svc_disp = plot_roc_curve(clf, test_X, test_Y)
217
218
219 from sklearn.metrics import accuracy_score
220
221 nn_score = accuracy_score(test_Y, nn_pred)
222 svm_score = accuracy_score(test_Y, svm_pred)
223 print(nn_score)
224 print(svm_score)
225
226
227
228
229 labels = ["NN", "SVM"]
230 usage = [nn_score*100, svm_score*100]
```

```
231
232 # Generating the y positions. Later, we'll use them to replace them
        with labels.
233 y_positions = range(len(labels))
234
235 # Creating our bar plot
236 matplot.bar(y_positions, usage)
237 matplot.xticks(y_positions, labels)
238 matplot.ylabel("Accuracy (%)")
239 matplot.title("Machine Learning algorithm")
240 matplot.show()
241
242 import numpy as np
243 import matplotlib.pyplot as matplot
244
245 # set width of bar
246 bar_width = 0.25
247
248 # set height of bar
249 bars1 = [results.mean()*100, accuracies.mean()*100]
250 bars2 = [nn_score*100, svm_score*100]
251
252
253 # Set position of bar on X axis
254 row_1 = np.arange(len(bars1))
255 row_2 = [x + bar_width for x in row_1]
256
257
258 # Plot the points
259 matplot.bar(row_1, bars1, color='#7f6d5f', width=bar_width, label='
        CV')
260 matplot.bar(row_2, bars2, color='#557f2d', width=bar_width, label='
        Predict')
261
262 # Insert ticks on group bars
263 matplot.xlabel('group', fontweight='bold')
264 matplot.xticks([r + bar_width for r in range(len(bars1))], ['NN', '
        SVM'])
```

```
265
266 # Generate the legend and display the graphics
267 matplot.legend()
268 matplot.show()
```

Listing 6.1: Python example