BEAT THE BOOKIE

A PREPRINT

Group Name: `Group B`
Department of Computer Science
University College London
London, WC1E 6BT

December 18, 2023

1 Introduction

Predicting English Premier League (EPL) match outcomes has long fascinated fans and experts. Recently, advancements in machine learning (ML) algorithms and the availability of extensive data have enhanced the accuracy and objectivity of these predictions. Our project aims to forecast EPL match results. Despite abundant data, selecting the most relevant information and preparing it for efficient ML algorithm training is challenging. Since football is not an individual sport but a team sport, there are a lot more factors to consider, which makes predicting accurate outcomes very difficult[1]. This task requires meticulous analysis and a strategic approach to achieve not only high-quality but also dependable results.

We made informed decisions at every stage of this project, guided by our expertise and a review of existing literature on the topic. Our thorough data analysis and sophisticated transformation techniques resulted in a high-quality dataset, ensuring relevance. With a refined dataset, various ML algorithms and model selection techniques were employed to find the best model for our data. Ultimately, our best model, a Multilayer Perceptron (MLP) optimized with `Optuna`, achieved an accuracy of 55.5%, demonstrating performance closely comparable to the bookmakers' benchmark of 54% [2].

The paper is structured as follows: We begin with a detailed exploration of our dataset and the data transformation processes employed (Section 2). This is followed by an in-depth discussion of our ML methodologies (Section 3), the training and validation of our models (Section 4), and an analysis of our results (Section 5). We then present our predictions for upcoming EPL matches (Section 6) and conclude with reflections on our research and potential directions for future studies (Section 7).

2 Data Transformation & Exploration

Data transformation and exploration are indispensable for machine learning predictions. These steps contribute to understanding, cleaning, and optimizing the dataset, addressing missing values, and performing feature engineering to enhance model interpretability and overall performance. This section involves steps: Data Preprocessing (Section 2.1), Raw Features Replacement (Section 2.2), Imputation (Section 2.3), Feature Encoding (Section 2.4), and Feature Selection (Section 2.5).

2.1 Data Preprocessing

Data pre-processing is a crucial step in transforming raw data into a format conducive to effective exploration, analysis, and modelling. We interchangeably use the terms "columns" and "features".

Our initial exploration focused on the `epl-training.csv` (original) dataset, which comprises a total of 22 columns. We identified inconsistencies in the values of certain features, such as varied date formats in the `Date` feature (`14/08/23` and `14/08/2023`) and irregularities in the naming convention of the `Referee` feature. For instance, a full name like `Rob Adam Harris` was stored variously as `Rob Harris`, `R Harris`, or `Harris, R. A.`, and sometimes prefixed with a dagger symbol (†) as †`Rob Harris`. Additionally, incorrect names for the same individual, such as `John Harris`,`J Harris`, `Rob Harry`, or `R Harry`, were also noted. To ensure coherence in each feature's values, we converted `Date` values into Pandas `datetime` type and standardized the names in the `Referee` feature to a consistent format, using the first name initial followed by the full last name, such as `R Harris`.

We crawled additional features from the FBref [3] website to enhance the quality of our dataset, saving it as `matches.csv` (crawling). The initial data we collected comprised 28 columns. From these, we eliminated features that were weakly related to predicting the game outcome, retaining 10 relevant features. These include `poss_H`, `dist_H`, and `fk_H`, representing `possession Home`, `distance covered Home`, and `free kicks Home`, respectively.

Before analysing the data, we merged the original and the crawled data into a unified dataset. First, we limited the original data to the past five years, considering that a team's squad changes every season; therefore, we deemed it inappropriate to include matches from more than five years ago. We then performed tasks such as substituting values and renaming entries in the `Referee` column of the crawled data, as well as converting the `Date` column into a Pandas `datetime` type to align it with the original dataset. The merger was executed based on the unique identifiers `Date` and `Referee`, which are unique to each match. This process resulted in a consolidated dataset totaling 32 columns.

Lastly, we pre-processed the `test.csv` (test) data, which contains only three columns: `Date`, `HomeTeam`, and `AwayTeam`. To maintain consistency with the original dataset, we converted the `Date` values into the Pandas datetime format. Additionally, we built a conversion dictionary for `HomeTeam` and `AwayTeam` to facilitate the replacement and standardization of their values.

2.2 Raw Features Replacement

To make predictions with ML models, we need to have the same feature set for both the training and test sets. However, when forecasting the outcome of a future match, we only have the date and the two competing teams, without any other feature values that are present in the training dataset. To address this issue, we opted to use the average of previous values for both the training and test datasets, instead of relying directly on the raw data.

We've introduced three types of average-based features: HomeTeam Average (`H_avg`), AwayTeam Average (`A_avg`), and Face-to-Face Average (`avg_face`). The computation of `HomeTeam` and `AwayTeam` feature averages, such as `FTHG_avg`, `FTAG_avg`, `HS_Avg`, and `AS_Avg`, involves calculating the mean of two mean values for each respective feature. These values are derived from the last five matches preceding a given match date for both the `HomeTeam` and `AwayTeam`. In contrast, calculating the Face-to-Face Average values involves determining the mean of face-to-face match results between the same `HomeTeam` and `AwayTeam` from the two matches prior to a given match date. To reduce uncertainty, we set the mean values for the first match to `NaN` instead of zero, as no previous matches exist at that point. After these implementations, we have 58 features.

For the `Date` column, which is in Pandas `datetime` format and incompatible with ML models, we addressed this issue by using the Fourier Transform method, as explained by Andrei [4]. This approach allowed us to replace the `Date` column with two float type columns: `cos_time` and `sin_time`. The Fourier Transform decomposes any periodic signal into a sum of sine and cosine components, effectively representing large periodic seasonal features with a minimal number of components. While one-hot encoding or breaking down the dates into year, month, and day could have been used for simpler, shorter periods, we opted for the Fourier Transform. This method is better suited for handling complex waveforms or extended periods, as it can represent these patterns with fewer features. After applying this Fourier Transform replacement, our feature count increased to 60.

2.3 Imputation

ML models are unable to comprehend `NaN` missing values within the data. We identified 20 missing values each in the `HomeTeam` and `AwayTeam` average features, along with 664 missing values each in the face-to-face average features, accounting for 34% of our dataset. Discarding these missing values is not a viable option. Instead, we opted for an iterative imputer to estimate the missing values by modelling each feature as a function of another element. The `IterativeImputer` we used is implemented by Scikit-Learn which is an experimental feature. This experimental feature is based on Multivariate Imputation by Chained Equations in R (`MICE`)[5] which will predict the data that the missing entry would normally exhibit while preserving the relationship among the data. It's important to note that the `IterativeImputer` produces a single imputation, disregarding the uncertainty of the model that calculates the missing value [6]. Despite this, we chose to overlook the uncertainty based on the documentation's indication [7] that using single imputation or multiple imputation does not significantly impact predictions when uncertainty is not a key consideration.

2.4 Feature Encoding

We divided our data into an 80:20 ratio for the training and test sets to train the models. Specifically for MLP models, we further split the test set into two halves to create a validation set and a separate test set. During the hyperparameter tuning process, `RandomizedSearchCV` was used to perform k-fold cross-validation.

For the target variable (FTR), we applied both Label Encoding and One-Hot Encoding. This dual approach allows the data to be compatible with both regression and classification models. Regarding the features, only two non-numeric features were present: `HomeTeam` and `AwayTeam`. We applied One-Hot Encoding to these features, resulting in a total of 110 features.

Scaling is critical in ML to ensure consistency in magnitude across different units or scales. It not only facilitates improved convergence but also enhances model performance and ensures compatibility with certain algorithms, like principal component analysis (PCA). To achieve this, we used the `StandardScaler` to transform the 52 Gaussian features we identified. This step ensures that gradient descent converges more smoothly and quickly during the training phase.

2.5 Feature Selection

Feature selection is a critical component of the process to reduce the number of features to mitigate computational complexity, enhance model performance, and improve accuracy. Three distinct methodologies, namely the wrapper method, filter method, and embedded method were used to identify the optimal 10 features to decide our final feature set and reduce the 110 features into a smaller feature set.

Wrapper method is a technique that assesses subsets of features based on the performance of a machine-learning model. The iterative process of a wrapper method involves selecting and evaluating subsets, training the model, and utilising performance metrics to ascertain optimal feature combinations. We used Recursive Feature Elimination (RFE) to iteratively repeat the `LogisticRegression` model by removing 10 features at a time.

Filter method is divided into two main categories: Univariate Filter Methods and Multivariate Filter Methods. Univariate methods evaluate and rank individual features independently based on criteria like Fisher score, mutual information, or feature variance. However, these methods may neglect inter-feature relationships, leading to the selection of potentially redundant features. In contrast, Multivariate Filter Methods take into account the mutual relationships between features, allowing for the identification and removal of redundant and correlated features from the dataset.

Embedded method is a technique that integrates the process of selecting relevant features into the training of a machine learning model. Unlike filter methods that evaluate features independently or wrapper methods that use model performance for selection, embedded methods incorporate feature selection as an inherent aspect of model training. During the training process, these methods automatically learn which features are most informative for the task. We used two techniques: Elastic Net and Random Forest. Elastic Net Regression combines features of Lasso and Ridge Regression. Additionally, Random Forest builds multiple decision trees and combines their predictions, inherently capturing feature importance during the training process.

	Wrapper Method	**Filter Method**	**Embedded Method**	**Overall**
1	AS_avg_face	FTAG_avg	poss_H_avg_face	FTHG_avg
2	HTAG_avg	HST_avg	poss_A_avg_face	HS_avg
3	AY_avg_face	AS_avg	dist_A_avg_face	poss_H_avg_face
4	FTHG_avg	poss_H_avg_face	HS_avg	poss_A_avg_face
5	HC_avg	FTHG_avg	HC_avg_face	HC_avg_face
6	poss_A_avg_face	poss_A_avg	AS_avg_face	HST_avg
7	HS_avg	poss_H_avg	HST_avg	AS_avg
8	pk_A_avg_face	AC_avg	AS_avg	poss_A_avg
9	HC_avg_face	HC_avg	HTAG_avg_face	poss_H_avg
10	pk_H_avg	AST_avg	FTHG_avg	AC_avg

Table 1: Top 10 most important features

As shown in Table 1 we computed the rankings of the top 10 most significant features, with the highest-ranked feature listed first and the 10th most important feature at the bottom. To validate the reasonability of our selected features, we used a Random Forest Classification model to compare the impact of feature removal. Initially, our dataset comprised 110 features, yielding an accuracy of 52.7%. Following the implementation of our feature selection process, the feature set was refined to 48 features, resulting in an increased accuracy of 53%, representing a 0.3% small improvement. It's noteworthy that this modest improvement is considered negligible and has minimal impact on the decision to drop the selected features which we decided to drop them for our final feature set.

3 Methodology Overview

When presented with the task, our intuitive response was to analyse as many classifier models as possible to compare their results and draw conclusions from different approaches. This was motivated by the diversification of our approaches, which would lead to a more robust result and ideally more accurate prediction. Considering initial research done in previous related works [1][8][9][10], the use of simple ML algorithms like Linear Regression (LR), Support Vector Machines (SVM) and Naive Bayes are deemed to perform well for the multi-label classification task, hence we decided to study these models under their forms of Logistic Regression with softmax, Support Vector Classifier (SVC) and Gaussian Naive Bayes `GaussianNB` respectively. These base models are often considered as benchmarks, yet we've decided to consider further models.

To improve our models, we considered taking a step forward in LR and opted to study a Ridge classifier with built-in cross-validation `RidgeClassifierCV`, this selection provided a model that accounted for a Regularization while automatically tuning in a Cross-Validation for its hyper-parameters. To diversify our model repertory further, guided by our previous research, we considered a Tree based models starting with Decision Tree Classifier as a base for this class and the model Random Forests. Similarly, we expanded to another class Ensembles, which consists of grouping the prediction of distinct models to seek an improved performance[11]; implemented by the Gradient Boosting Classifier and Histogram-based Gradient Boosting Classification Tree. These two implementations were then combined with the trained models of LR, Random Forest, and SVC to make a `VottingClassifier` called Ensemble aimed to use the individual learnt knowledge of each model and merge them into an ideally more accurate model. While these models perform well theoretically for classification tasks the research focus of ML has shifted towards Neural Networks (NN).

The use of NN in ML is widely promoted due to their self-adaptive nature and because they are universal functional approximators, leading to being applied to several classification problems [12]. The research [13] motivated us to study a Multi-layer Perceptron (MLP) Classifier, which allowed multi-class classification. All of the aboved mentioned models were implemented using the Sci-kit Learn library. Considering the accuracy of the MLP, we implemented two own versions of an MLP NN to explore further the difference the number of nodes in the NN would generate on the accuracy of the model. The first version used `Optuna` while the second was manually optimized.

These models were then subject to training and model selection. We used a Cross Validation technique over our models except for `RidgeClassifierCV`, this decision was made due to the size of our data set. Then the evaluation phase took all the models with their best parameter fitted during the model selection and predicted the values to then compare to our test data and calculate accuracy. Finally, our prediction was taken from the most accurate model. This process is explained in more depth over the sections 4,5 and 6.

4 Model Training & Validation

In our quest for superior performance, we embarked on a comprehensive evaluation of a diverse array of 12 models. This included 9 standalone models, readily available in the scikit-learn library. To augment this collection, we crafted an ensemble model, meticulously curated from the top 5 performers among the individual models. The final two models in our repertoire were Multi-Layer Perceptron (MLP) models, one manually tuned and the other fine-tuned using `Optuna`.

The models we evaluated encompassed a wide range of techniques:

- Support Vector Classifier (SVC)
- Random Forest Classifier
- Decision Tree Classifier
- Gaussian Naive Bayes
- Gradient Boosting Classifier
- Ridge Classifier
- MLP Classifier
- Histogram Gradient Boosting Classifier
- Logistic Regression
- An Ensemble model (excluding Ridge)
- Two custom MLPs

Our testing procedures were designed with precision, aiming to thoroughly compare and assess the performance of these models, each unique in its approach.

4.1 Model Training

All models were trained on our dataset, covering the data from the last five seasons. Past articles indicated that this amount of data would be sufficient for our task.[14]

We optimized a diverse set of classifiers using `RandomizedSearchCV`, a robust technique for hyperparameter tuning.[15] The initial step involved defining a list named `base_classifiers`, which paired classifier instances (e.g., `Support Vector Classifier (SVC)`, `RandomForestClassifier`, and others) with dictionaries specifying hyperparameters to be tuned.

We invoked the built-in `fit` method on each `RandomizedSearchCV` instance, leveraging our training data. This step activated the hyperparameter search through cross-validation, and we recorded the trained models in the `models` list.

In constructing the ensemble model, we selected the top 5 out of 9 individual models based on their performance. We employed the 'soft' voting method for the ensemble and used instances of `RandomizedSearchCV` directly for training.

Our Multi-Layer Perceptron (MLP) models utilized the Rectified Linear Unit (ReLU) as the activation function and the softmax activation function at the final layer for classification into three categories: H, D, and A. Both models employed `CrossEntropyLoss` for the loss function, a common choice for multi-class classification problems, as it combines with the softmax activation function. We used the Adam optimizer in both models. A crucial difference between the models lies in their selection criteria during training. While both models were trained by minimizing the loss function, one model was selected based on the highest accuracy on the validation set, whereas the MLP model using Optuna was chosen based on the lowest loss.

4.2 Hyperparameter Optimization

While `GridSearchCV` and `RandomizedSearchCV` have traditionally been the go-to methods for hyperparameter tuning, recent advancements in machine learning research have brought their limitations to the forefront, deeming them outdated, ineffective, and inefficient compared to newer alternatives.[16] Cutting-edge packages like `Hyperopt` showcase significant improvements, offering a diverse range of methods for exploring the hyperparameter space, including modified tree Parzen estimators.[17]

In our pursuit of optimal hyperparameter configurations, we developed two MLP models. One leverages the capabilities of `Optuna`, benefiting from its sophisticated hyperparameter search toolbox and universal compatibility.[18] This approach proved crucial, especially considering the diverse range of ML models we tested for hyperparameter tuning. The second model involved a manual adjustment of hyperparameters.

In the realm of hyperparameter tuning for scikit-learn models, `RandomizedSearchCV` emerges as the most suitable, efficient, and widely applicable method.[19] This is attributed to the manageable number of hyperparameters in classifiers provided by scikit-learn, making `RandomizedSearchCV` a sufficiently powerful tool. For models like `MLPClassifier` and gradient boosting, which entail a more extensive set of hyperparameters, our experiments revealed that increasing parameters such as iterations or node numbers in gradient boosting models may not necessarily lead to improved results. Consequently, we advocate the advantages of employing `RandomizedSearchCV` to discover an optimal set of hyperparameters for scikit-learn models.

4.3 Model Validation & Evaluation

Effective model evaluation requires testing the model's accuracy on previously unseen data to ensure an authentic gauge of its performance.[20] To safeguard against potential bias in the evaluation process, we employed a sophisticated approach known as `Stratified k-fold cross-validation`. This technique ensures a balanced representation of class distributions in each fold, thereby fostering a comprehensive evaluation of model performance.[21] In our model assessments, all models fine-tuned through `RandomizedSearchCV` underwent an 8:2 split for training and testing. `Stratified k-fold cross-validation` with a fold count of 5 (cv=5) was chosen as the preferred technique. For custom MLP models, we partitioned the dataset into distinct sets, encompassing training, validation, and testing subsets (8:1:1). This tailored approach afforded us the ability to assess the performance of our custom MLP architectures. Through this nuanced and diverse methodology, we aimed to leverage the strengths of custom-tailored strategies for specific models, while also ensuring a broad and reliable assessment framework for the ensemble as a whole.

5 Results

5.1 Benchmarks

The performance of various models was evaluated based on accuracy, recall, precision rates, and the F1 score. To contextualize these results, we established two benchmarks: a lower bound and an ideal value. The lower bound, or the no information rate, is the accuracy achievable by guessing the same outcome for all matches. This rate is exemplified in Table 2, where guessing that the home team wins every match results in an accuracy of 43%. The ideal value is set as the de facto gold standard, which is the bookmakers' accuracy rate, documented at 54% [2]. Therefore, the goal for our models is to achieve an accuracy exceeding 43%, while striving to approach or surpass the 54% mark.

	Home win	**Away win**	**Draw**
Counts	173	125	102
Ratio (3 s.f.)	0.433	0.313	0.255

Table 2: No information rate

5.2 Model Comparison

As indicated in Table 3, it is noteworthy that the highest accuracy achieved is 55.5% with the MLP using `Optuna`. This result demonstrates that our best-performing model not only surpassed the gold standard but also gained an additional stochastic advantage. However, considering the margin of error, which we observed to be approximately $\pm 1\%$, models like Random Forest, Gradient Boosting, Logistic Regression, and Ensemble models exhibit similar levels of performance in terms of accuracy and F1 score. Therefore, while it's challenging to definitively assert the superiority of the MLP model, its success suggests significant potential for further improvement. This aspect, along with other insights, will be further discussed in Section 7.

	Accuracy	**F1 Score**	**Recall**	**Precision**
Kernel SVM	0.51	0.441	0.51	0.548
Random Forest	0.53	0.452	0.53	0.395
Decision Tree	0.405	0.401	0.405	0.398
Gaussian Naive Bayes	0.475	0.442	0.475	0.435
Gradient Boosting	0.528	0.458	0.528	0.52
Ridge Classifier	0.512	0.45	0.512	0.47
MLP Classifier	0.42	0.411	0.42	0.409
Histogram Gradient Boosting	0.532	0.468	0.532	0.486
Logistic Regression	0.525	0.446	0.525	0.389
Ensemble	0.522	0.395	0.461	0.346
MLP Manual	0.55	0.48	0.55	0.427
MLP with `Optuna`	0.555	0.485	0.555	0.432

Table 3: Accuracy, F1 score, recall, precision performance of models.

5.3 Analysis

It is worth discussing the models' prediction with the two top-performing models: MLP with `Optuna` and Histogram Gradient Boosting. As illustrated in Figure 1, it is likely for models to predict the results as H (Home Team Win) since the statistics of match history imply the home advantage exists as shown in Figure 2. On the other hand, it was rare for a model to predict the Full Time Result (FTR) as Draw (D). Outcomes that might have been accurately labelled as Draw often ended up being classified as either H or Away Team Win (A). This is primarily because non-Draw options constitute the majority of outcomes, and predicting a Draw accurately requires discerning subtle differences that our models currently struggle with. Enhancing the models' ability to correctly predict Draw outcomes could significantly improve the overall performance metrics.

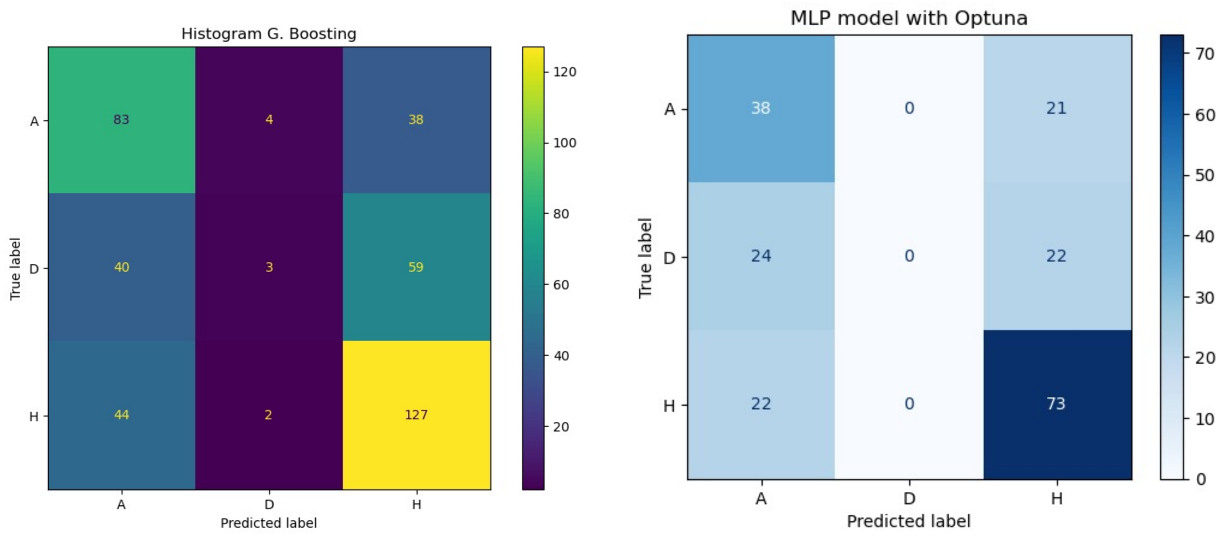

Figure 1: Confusion matrices for Histogram Gradient Boosting and MLP model with Optuna

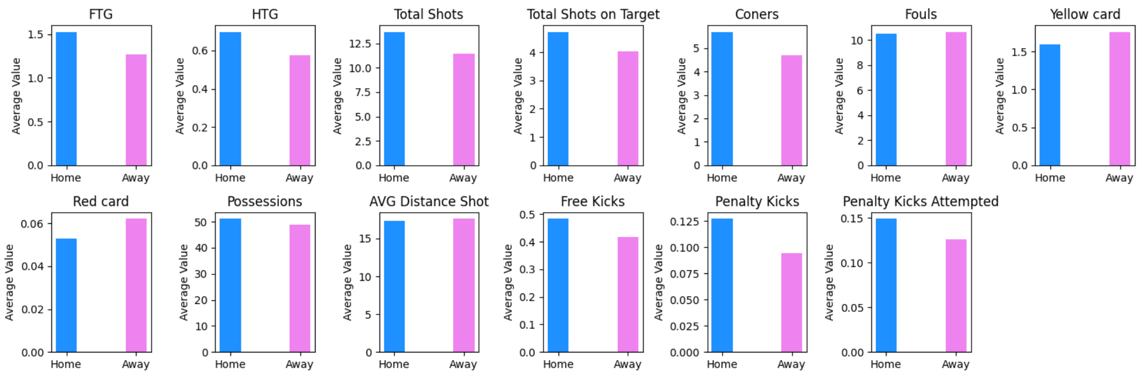

Figure 2: Existence of home advantage: comparing the average of each feature for the home and away teams exhibits a clear advantage for the home team.

6 Final Predictions on Test Set

We applied the same feature engineering and data pre-processing on the Test set as we did on the training set. We used our trained MLP with `Optuna` model which showed the best accuracy to make our predictions. Our predictions are shown in Table 4.

7 Conclusion and Future Research

In this project, we aimed to produce a high-quality dataset and apply appropriate ML techniques to predict the outcomes of EPL matches. Leveraging our understanding of the domain, comprehensive data analysis, and meticulous feature selection and engineering, we developed a robust dataset spanning 2018 to 2023. We evaluated a wide range of advanced ML models with k-fold cross-validation and ultimately selected the top-performing model, MLP with `Optuna`, which surpassed the bookmakers' gold standard of predictive accuracy.

Our work has also uncovered potential directions for future research. For instance, considering external factors like weather could enhance predictions, as we found that match dates influenced accuracy. Additionally, RNNs can be employed for their ability to process sequences of data and their 'internal memory' of previous events, which is crucial for understanding the context of a game within a season or series of matches [22]. Furthermore, RNNs have evolved into long short-term memory (LSTMs) and gated recurrent units (GRUs) to overcome the vanishing gradient issue RNNs have[23].

Date	Home Team	Away Team	FTR
2024-02-03	Bournemouth	Nott'm Forest	H
2024-02-03	Arsenal	Liverpool	H
2024-02-03	Brentford	Man City	A
2024-02-03	Brighton	Crystal Palace	H
2024-02-03	Burnley	Fulham	A
2024-02-03	Chelsea	Wolves	H
2024-02-03	Everton	Tottenham	H
2024-02-03	Man United	West Ham	H
2024-02-03	Newcastle	Luton	H
2024-02-03	Sheffield United	Aston Villa	A

Table 4: Final Prediction

References

[1] Tomislav Horvat and Josip Job. The use of machine learning in sport outcome prediction: A review. *WIREs Data Mining and Knowledge Discovery*, 10(5):e1380, 2020.

[2] Ryan Beal, Timothy J. Norman, and Sarvapali D. Ramchurn. Artificial intelligence for team sports: a survey. *Knowledge Engineering Review*, 34, 2019.

[3] FBref. Premier league stats. `https://fbref.com/en/comps/9/stats/Premier-League-Stats`, 2018–2023. Accessed: 2023-12-16.

[4] Florin Andrei. Modeling variable seasonal features with the fourier transform: Improve time series forecast performance with a technique from signal processing. 2012.

[5] Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011.

[6] *Statistical Analysis with Missing Data*, volume 793 of *Wiley series in probability and statistics*. Wiley, Newark, third edition. edition, 2019.

[7] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[8] Johannes Stübinger, Benedikt Mangold, and Julian Knoll. Machine learning in football betting: Prediction of match results based on player characteristics. *Applied Sciences*, 10(1), 2020.

[9] Johannes Stübinger Julian Knoll. Machine-learning-based statistical arbitrage football betting. *KI - Künstliche Intelligenz (2020)*, 34:69–80, 2019.

[10] Nilay Zaveri; Shubham Tiwari; Pramila Shinde; Utkarsh Shah; Lalit Kumar Teli. Prediction of football match score and decision making process. *International Journal on Recent and Innovation Trends in Computing and Communication*, 6(2):162–165, 2018.

[11] Jason Brownlee. A gentle introduction to ensemble learning algorithms. `https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms`, 2021–2023. Accessed: 2023-12-16.

[12] Peter Zhang. Neural networks for classification: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30:451 – 462, 12 2000.

[13] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science Technology*, 9(2):14–14, 02 2020.

[14] Fadi Thabtah Rory P. Bunker a. A machine learning framework for sport result prediction. *sciencedirect*, 2019.

[15] Satyam Kumar. 7 hyperparameter optimization techniques every data scientist should know. *medium*, 2021.

[16] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications, 2020.

[17] James Bergstra, Dan Yamins, and David Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. pages 13–19, 01 2013.

[18] Umar Zaib. A deep dive into optuna vs. hyperopt for hyperparameter optimization excellence. *medium*, 2023.

[19] Georgios Tourloukis. Scikit-learn pipeline randomizedsearchcv | ml model selection | churn modeling dataset. *medium*, 2021.

[20] Everton Gomede. The significance of train-validation-test split in machine learning. *medium*.

[21] Sujit Kumar Dash Sashikanta Prusty, Srikanta Patnaik. Skcv: Stratified k-fold cross-validation on ml classifiers for predicting cervical cancer. *frontiers*.

[22] Georgi Boshnakov, Tarak Kharrat, and Ian G. McHale. A bivariate weibull count model for forecasting association football scores. *International Journal of Forecasting*, 33(2):458–466, 2017.

[23] Niklas Donges. A guide to RNN: Understanding recurrent neural networks and LSTM networks. `https://builtin.com/data-science/recurrent-neural-networks-and-lstm`, 2021. Accessed: 2023-12-16.