COSE362-2019F: Final Project Paper
Soccer Match Result Prediction and its Application

HyeongKyu Choi   HyunHo Choi   JaeHyun Lee   JiHwan Park   JuHyeon Shin

1. Introduction

The sports betting market has been steadily growing for the past ten years, and the trend is still on-going. Currently, the international sports betting market is presumed to have a market capitalization of $250 billion [1]. However, negative views on such betting markets do exist due to its gambling nature, and gambling should definitely not be generally encouraged. Here, our motivation of this project is to moderate the gambling nature of such betting conducts through providing data-driven analysis based on ML methods.
The main objective of our project is to build machine learning models from scratch, and demonstrate the optimized performance of each model. The specific task would be to predict the outcome of a specific soccer match – home win, away win, or draw. We have considered regression models for classification as well, by first predicting the final score with regression, and then determining which team has won, judging from the predicted scores.
The performance of each model is evaluated with respect to two criteria – predictive capacity and profitability. The predictive capacity is quantified in terms of the MSE, MAE, F1 metrics, and the profitability is measured through Return on Investment (ROI), that is, Net Profit divided by the total investment into the bet. The major challenge of this task is to prevent overfitting due to noisy data. To elaborate, a sports match is generally unpredictable, and results tend to be irregular. Thus, a model might overfit on such irregularities, leading to a poorly generalized model. We took subtle measures to improve each model, which will be specified in detail in section 4, Models and Optimization.
In section 2 Related Works, we provide academic backgrounds along with the baseline models we have took into consideration. In section 3, we discuss the data source and data preprocessing steps. In section 4, our from-scratch models are dealt with and optimization details are provided. In section 5, we provide experimental results of our models, along with some comparisons with the baseline models and benchmark settings. Finally, in section 6 and 7, we state our conclusions, discuss future works and summarize our individual roles.

[1] www.sportsbettingdime.com/guides/finance/global-sports-betting-market/

2. Related Works

Sports match result prediction is a frequently-dealt topic of research. Bunker proposed an ML-based method to predict the outcome of a sports match in general using historical data, player performance indicators and opposition information in his paper in 2017 [1]. Moreover, Kumar proposed a general approach in predicting the outcome of a soccer match in his thesis, mainly focusing on feature engineering [2]. In a more casual context, Carter introduced prediction results on soccer games and provided results in terms of both accuracy and profitability in his blog post [3].
We set Kumar's work as our baseline to evaluate the expected level of performance of each model. His SVM model had 45% of accuracy, and MLP classifier had 59% accuracy. Given the accuracy scores of two of the most intricate models, we may roughly assume that our from-scratch models will perform around such level at best.

3. Data

3.1. source

We use the raw data of English Premier League match result provided in football-data webpage[2]. Carter has provided in his github[3] an aggregated dataset from the football-data webpage from year 2008 to 2018. For convenience, we resolve to take advantage of the dataset.

3.2. preprocessing

Each feature vector in the data provided contains mainly three types of attributes : match result attributes, match record attributes and bet attributes. Match result attributes include the full-time and half-time scores of home and away, and the final outcome whether the result was home win, away win, or draw. We may utilize these attributes as are target variable. The match record attributes contain information on the match record itself, such as number of shots, number of fouls, *et cetera*. Finally, the bet attributes contain information on the odds for home, away, and draw of each betting sites. The higher the odds, the more return

[2] www.football-data.co.uk/englandm.php
[3] www.github.com/AndrewCarterUK/football-predictor

you get for correct predictions.
Unfortunately, the match record attributes cannot be used directly because it contains the 'after-match' information of the match we want to predict the outcome for. That is, data leakage problem will occur if we use the attributes without preprocessing. Therefore, we renew the match record data by taking the average of the most recent n matches from that point of time. For instance, we take the average number of shots of the home team in its previous five games and name the new variable 'home_shots'. The specific attributes newly generated are listed in Appendix A.
Furthermore, the bet attributes can be averaged out for all betting sites to reduce data dimensions. Hence, the average odds for 'Home Win', 'Draw', 'Away Win' are added into the input vector.

4. Models and Optimization

4.1. Linear Regression

Linear model makes a prediction by simply computing a sum of the weighted input features and bias term : $\hat{y} = \theta \cdot \mathbf{x}$. We choose Mean Square Error(MSE) for the loss function.

$$MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Our goal is to find the parameters that minimize the cost function. We choose Gradient Descent method to optimize the MSE cost function. This method is a generally well suited for cases where there are a large number of features. Gradient descent method first measures the local gradient of the cost function with respect to the parameter vector θ, and updates the parameters in the direction of descending gradient. When the gradient is zero, the parameters reach optimal solution.
The learning rate α is initialized to 0.01. Since it is hard to find the point when the gradient is exactly zero, we make our algorithm stop when the difference between past MSE and current MSE is under $1e-7$.

$$\theta^{(\text{next step})} = \theta - \alpha \nabla_\theta MSE(\theta)$$

The gradient vector, noted $\nabla_\theta MSE(\theta)$, contains all the partial derivatives of the cost function.

$$\nabla_\theta MSE(\theta) = \frac{2}{m}X^T(X\theta - \mathbf{y})$$

After predicting the home team score and away team score by regression, we classify it into three classes H, D and A.

4.1.1. Ridge Regression

Ridge regression is a regularized version of Linear Regression. The regularization term, which is added to the cost function, is $\beta\sum_{i=1}^{n}\theta_i^2$. This term prevents over-fitting by keeping the model weights as small as possible. Below equation is the cost function of ridge regression.

$$J(\theta) = MSE(\theta) + \beta\frac{1}{2}\sum_{i=1}^{n}\theta_i^2$$

Hyperparameter β, which controls how much we regularized the model, is added to the model. It is initialized as 0.01.

4.2. Logistic Regression

Logistic Regression has the cost function as follows, and is commonly referred to as Negative Log Likelihood (NLL).

$-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1-y^{(i)}) log(1-h_\theta(x^{(i)}))$

For optimization, we update the weights by subtracting the partial derivatives of the cost function by each weight parameters with the learning rate of 0.01. This is commonly known as Gradient Descent. Since Logistic Regression is a type of convex optimization, other optimization measures such as Momentum Gradient Descent or the Nesterov Accelerated Gradient Descent were not considered in depth.
In our specific task, we have 3 different classes : H, D, A. Given that the Logistic Regression is a binary classifier, we need to build a one-vs-rest classifier system for multi-class classification. To elaborate, we designate a target class as class '1', and the rest as class '0'. In this one-vs-all setting, the logistic regression model is fit to compute the probability value for each data instance with the hypothesis function. This is carried out for each class. After three iterations for each class, the class with the largest probability value is assigned to each data instance. The pseudo code is provided in Appendix B.

4.3. K-Nearest Neighbors : Classification & Regression

Both the KNN Classification and the Regression model are implemented. Since KNN is a non-parametric model, there is no need for the model to be trained beforehand. We have set the initial hyperparameter k to a rather large value of 11, so as to reduce variance in the prediction.

4.4. Kernel Regression

The kernel regression is a non-linear model which makes predictions within the context of data. The idea is similar to that of the KNN regression model, but we may expect the kernel regression to be smoother since the model adopts the interpolation method for unseen data. For the kernel function, we used the radial basis function, that is, the gaussian kernel as it may output an appropriate size of weight with respect to the similarity of the two inputs. The L2 norm is used for the distance metric.

$$y^* = \frac{\sum_{i=1}^{m} K(x^*, x_i) y_i}{\sum_{i=1}^{m} K(x^*, x_i)}$$

$$K(x^*, x_i) = exp(-\frac{||x_i - x^*||^2}{\sigma^2})$$

The predicted y is a weighted average of past data using the weights computed through the gaussian kernel. The best standard deviation, which determines the centrality of context, was selected through an empirical process. It was found that a value of 0.5 was optimal. As the standard deviation was shifted upward or downward from the value, the model started to predict only for one class. As the standard deviation increases, model variance increases, and as it decreases, model bias increases.

4.5. Support Vector Machine Classification

A soft margin linear SVM is implemented in this project. To extend SVM to cases in which the data are not linearly separable, we introduce the hinge loss function.

$$max(0, 1 - y_i(w \cdot x_i + b))$$

$$subject\ to\ \ (wx_i + b)y_i \geq 1 - \xi_i$$

Note that y_i is the i-th target and $w \cdot x_i + b$ is the current predicted output.
This function is zero if the constraint is satisfied, that is, if x_i lies on the correct side of the margin. For data on the wrong side, the function's value is proportional to the distance from the margin. Then, the objective function is

$$[\frac{1}{n}\sum_{i}^{n} max(0, 1 - y_i(w \cdot x_i + b))] + \lambda \|w\|^2$$

where parameter λ determines the trade-off between increasing the margin size and ensuring that the x_i lie on the correct side of the margin. Hence, for sufficiently small values of λ , the second term in the loss function will become negligible.
In the above equation, we have set the allowable margin as 1. However, we may generalize this term as δ and rewrite the equation as follows.

$$min\|w\|^2 + C\sum_{j} max(0, \delta - (w \cdot x_j + b)y_j)$$

We now have three hyperparameters : C, α, δ. C controls the affect of regularization, δ the allowable margin, and α the learning rate. In our experiment, we used 0.01 for C, 1 for δ, and 0.005 for the learning rate α.
In a multi-label setting, the gradient of the hinge loss function for data point j is as follows.

$$\nabla_{w_{y_j}} L_j = -(\sum_{i \neq y_j} \mathbb{1}({w_i}^T \cdot x_j - {w_{y_j}}^T \cdot x_i + \Delta > 0))x_i$$

$$\nabla_{w_i} L_j = \mathbb{1}({w_i}^T \cdot x_j - {w_{y_j}}^T \cdot x_i + \delta > 0)x_j$$

$$\nabla_w \|w\|^2 = 2\|w\|$$

With the computed gradient $\frac{\partial}{\partial \theta} J(\Theta)$, gradient descent is conducted. The pseudo code is provided in Appendix B.

4.6. Multi-Layer Perceptron Classification

The MLP Classifier with a single hidden layer of size 64 is implemented. Since the input dimension is 41, and the number of classes to predict is 3, layer 1's weight $W^{(1)}$ has dimension (41, 64), and the output layer has dimension (65, 3). Note that the bias node is added for the hidden layer.
Frequently, the MLP classifier is affected by initialization of weight parameters. In addition to a simple random sampling from the uniform distribution, an initialization measure called Xavier uniform initialization is implemented. Using this method, each weight value w_{ij} is sampled from a uniform distribution with bound $(-\sqrt{\frac{6}{d_{in}+d_{out}}}, \sqrt{\frac{6}{d_{in}+d_{out}}})$. Here, d_{in} and d_{out} signifies the dimension of a neural network layer. Furthermore, in an attempt to better find the global optimum, the momentum method for optimization was considered. More specifically, the Nesterov Accelerated Gradient Descent is implemented. Lastly, so as to deal with the frequent overfitting problem with the MLP classifier, a method called dropout was added. This method stochastically 'drops out' a certain portion of neurons during an epoch while training in order to prevent overfitting.
Apart from the model optimization logic, the prediction mechanism is tuned as well. Instead of naively using the predicted output of the model, we set a decision threshold of 0.45 so that we use the model's prediction only when the predicted label's probability is higher than the threshold. Otherwise, the model will automatically choose the label with the lowest odds, that is, the label which most people have predicted.
After experimentation, we have found out that the simple neural net without Xavier initialization and NAG performs better. Therefore, our final experiment was carried out using a simple neural network only with dropout and a decision threshold. The learning rate for the model was set to 3e-4, and the dropout probability was set to 0.3.

5. Experiments

In our experiment, we designated data from year 2017 and 2018 as our test set, and the other years are used for training. Our performance report will be based on the test set.
The following Table 1 lists the experiment result along with the two baseline models from Kumar's paper [1]. Note that MSE and MAE were only computed for regression models, and F1-score is the macro f1 score which averages out the f1 scores of each label H, D, and A. And the ROI is the Return on Investments, which is the total profit of the bets divided by the total amount of money invested in the bet. It

is computed as follows.

$$ROI = \frac{\sum_i^N \mathbb{1}(\hat{y}_i == y_i) * odds_i * bet}{bet * N} - 1$$

The experiment shows a rather perplexing result. The F1 score and the ROI seems to have a negative correlation. That is, a model that had a higher prediction error earned more in years 2017 and 2018. Considering that a soccer match is full of anomalies and unexpected outcomes, a random classifier may have an edge in such match anomalies. That is, if one predicts an unexpected match correctly, the odds for the correct prediction is extremely high, and its contribution to the ROI will be significant. In other words, the models that have classified 'at random' could have had some chance to 'predict' an anomaly correctly, and thus have a higher return. However, such predictions are innately unstable, and will encroach the total ROI in the long run. Conversely, relatively simple models performed well in terms of the F1 score. We believe this prediction pattern is analogous to that of a human being. It predicts well on obvious matches with low odds, but fail to predict for unexpected outcomes, thus losing the chance of winning for high odds.
Overall, the ridge linear regression model has performed well. Its F1 score ranks second place, and the ROI is also quite high. In order to better evaluate the model, we carried out a backtest for the years 2017 and 2018. Here, we set two benchmarks. One is a random classifier that predicts the outcome of a match randomly among H, D, and A. The other is called the 'copycat' which always predicts the class with the lowest odds. That is, the model follow suits what people in general have selected. The ROI is computed for every soccer match in sequence, and is plotted to see the trend [Figure 1]. From the plot, we may see that the Ridge Regression Model outperforms its counterparts by a significant scale.

	MSE	MAE	Accuracy	F1	ROI
BL-SVM	-	-	0.4500	0.28	-
BL-MLP	-	-	0.5947	0.69	-
LINEARreg	1.47	0.92	0.5383	**0.52**	1.20
RIDGEreg	1.50	0.92	0.5306	**0.50**	1.27
KERNELreg	1.53	0.93	0.5230	0.46	1.26
KNNreg	1.65	0.97	0.4949	0.44	1.15
KNNclass	-	-	0.5255	0.42	**1.28**
LOGITreg	-	-	0.5536	0.44	1.17
LIN SVM	-	-	0.5510	0.39	**1.30**
MLPclass	-	-	0.5638	0.47	1.18

Table 1. Experiment Result

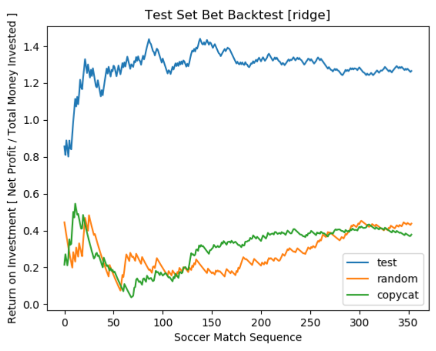

Figure 1. Ridge Regression Backtesting Result

6. Conclusion

We have addressed the question whether a soccer match result can be predicted through a machine learning approach. Our main concern was to prevent the model from overfitting on noises. Various models were implemented – from linear regression to the MLP classifier. The source codes are publicly available on github[4]. Our models were evaluated based on their predictive performance metrics, and were further assessed through profitability measures and backtesting. The backtesting results signify that our models significantly outperform the random classifier and 'copycat' model. Also, results show that in terms of predictive capacity, simpler models perform relatively superior to complex models. This pattern seems to be caused from the innate noise in the dataset itself; high-capacity models tend to overfit on such features, notwithstanding our efforts to prevent them, while a model with simpler hypotheses performs better.
In our project, we have not dealt with soccer player data. Normally, the specific players in the match may greatly influence the outcome. Future work that incorporates such information is called for.
Considering that the neural network is generally preferred in many domains these days, it was an unexpectancy that rather simple models outperformed it. It seems that the quote definitely holds. *There is no free lunch.*

7. Individual Roles

2013120315 HyunHo Choi	Kernel Regression & KNN
2015120300 HyeongKyu Choi	MLP Classifier
2015170740 JaeHyun Lee	SVM Classifier
2016160030 JuHyeon Shin	Linear Regression & Ridge
2017320151 JiHwan Park	Logistic Regression

[4]www.github.com/imhgchoi/soccer-match-predict

References

[1] RP Bunker. F Thabtah. A machine learning framework for sport result prediction. *Applied Computing and Informatics*, 2017.
[2] G Kumar. Machine Learning for Soccer Analytics. (master's thesis), 2013
[3] A Carter. "A Beginners Guide to Beating the Bookmakers with TensorFlow", Feb 2018, andrew.carterlunn.co.uk/programming/2018/02/20/beating-the-bookmakers-with-tensorflow.html

Appendix A

Feature Specification

· Hodds, Dodds, Aodds :
Average odds of betting sites for the specific match. Corresponds to odds for Home, Draw, Away
· home_goals, away_goals :
Average goals scored in the most recent N matches of the Home(Away) team
· home_oppos_goals, away_oppos_goals :
Average goals scored in the recent N matches between the Home and Away
· home_oppos_wins,home_oppos_draws,home_oppos_losses:
The ratio of wins, draws, losses of the Home team in the recent N matches against the Away team. 1 minus the value naturally signifies the ratio of wins, draws, losses of the Away team in the recent N matches against the Home team.
· home_wins, away_wins :
The win ratio in the most recent N matches of the Home(Away) team
· home_losses, away_losses :
The loss ratio in the most recent N matches of the Home(Away) team
· home_draws, away_draws :
The draw ratio in the most recent N matches of the Home(Away) team
· home_shots, away_shots :
Average number of shootings in the most recent N matches of the Home(Away) team
· home_oppos_shots, away_oppos_shots :
Average number of shootings in the recent N matches played against the same opponent
· home_shotontarget, away_shotontarget :
Average number of shootings on target in the most recent N matches of the Home(Away) team
· home_oppos_shotontarget, away_oppos_shotontarget:
Average number of shots on target in the recent N matches played against the same opponent
· home_cornerkicks, away_cornerkicks :
Average number of cornerkicks in the most recent N matches of the Home(Away) team
· home_oppos_cornerkicks, away_oppos_cornerkicks :
Average number of cornerkicks in the recent N matches played against the same opponent
· home_fouls, away_fouls :
Average number of fouls committed in the most recent N matches of the Home(Away) team
· home_oppos_fouls, away_oppos_fouls :
Average number of fouls committed in the recent N matches played against the same opponent
· home_yellowcards, away_yellowcards :
Average number of yellowcards received in the most recent N matches of the Home(Away) team
·home_oppos_yellowcards, away_oppos_yellowcards :
Average number of yellocards received in the recent N matches played against the same opponent
·home_redcards, away_redcards :
Average number of redcards received in the most recent N matches of the Home(Away) team
·home_oppos_redcards, away_oppos_redcards :
Average number of redcards received in the recent N matches played against the same opponent

Appendix B

Algorithm 1. Logistic Regression

```
 1: classes = [H, D, A]
 2: for c in classes do
 3:     repeat
 4:         $\Theta_c = \Theta_c - \alpha \cdot \frac{\partial}{\partial\Theta} J(\Theta_c)$
 5:     until $loss$ doesn't change
 6:
 7: # Prediction for X
 8: for c in classes do
 9:     probabilities[c] = $\Theta_c(X)$
10: return argmax(probabilities)
```

Algorithm 2. Multi-class Support Vector Machine

```
 1: classes = [H, D, A]
 2: for c in classes do
 3:     repeat
 4:         $\Theta_c = \Theta_c - \alpha \cdot \frac{\partial}{\partial\Theta} J(\Theta_c)$
 5:     until $loss$ doesn't change
 6:
 7: # Prediction for X
 8: for c in classes do
 9:     scores[c] = $\Theta_c(X)$
10: return argmax(scores)
```