Beating the Odds: Learning to Bet on Soccer Matches Using Historical Data

Michael Painter, Soroosh Hemmati, Bardia Beigi

SUNet IDs: mp703, shemmati, bardia

Introduction

Soccer prediction is a multi-billion dollar industry played by over 250 million football players in over 200 countries. Soccer is the national sport of many countries in the world and the love of the sport transcends national and international borders. It is perhaps the only sport in the world that does not need any introduction. In this project, we have a multi-class classification problem over a set of input data that is a database containing player, team, match, and league features on 8 seasons from 11 different European soccer leagues. We then use an algorithm such as SVC to classify the result of a new match into one of home win, draw, or away win. In particular, due to the huge amount of data available and the innate differences between different leagues, we opted to try and model the English Premiere League (EPL). Despite this size reduction however, we still have about three times as many features per match (about 1130) as there are matches during a whole season (380). As a result, we narrowed down the main objective of this project to be picking the best features to predict the outcome of a match. We should note that this project is not being used for any other classes.

Related Work

Due to the popularity of soccer, many have attempted to predict the outcome of the beautiful game using a number of different approaches. A fairly common method in predicting the outcome of soccer matches is using collective knowledge [1]. With the availability of online platforms such as twitter, it has become increasingly easier to gain massive amounts of collective knowledge and use them for prediction [1, 2]. Other approaches exist which mostly focus on modeling teams based on their performance in the most recent history of matches [3]. For example, Magel and Melnkov in [3] use the sum of differences in the number of cards and goals score for or against for each team during the last k matches as their features to effectively predict the outcome of new matches. Other methods exist in which the authors tried to find the attributes experts use to rate players and teams and used these features both for match description and prediction [4]. Finally, there are methods in which the focus is to systematically find the most valuable predictors for soccer matches and to logically build upon that data to achieve maximal prediction accuracy [5].
What is certain is that there exists a huge deal of literature based on a vast variety of viewpoints to accurately identify and incorporate suitable features to predict the outcome of soccer matches. What is certain though, is that there is a lot more work to be done in this area to systematically being able to get consistent prediction accuracy across a wide range of leagues and the search, is far from over.

Dataset

Our data is taken from https://www.kaggle.com/hugomathien/soccer. The data is stored in a .sqlite database and contains tables named Country, Player, Team_Attributes, League, Match, Team, and Player_Attributes. The data describes the results and statistics of matches, player, and leagues from 11 European countries. The data is spread over 8 different seasons, of over 11000 players and 26000 matches. There are over 1100 features per match, with 80% pertaining to the players present during that game. The following are examples of schemata and data present in the database:

- **Match:** (id, country_id, league_id, season, stage, date, match_api_id, ..., home_player_X1, etc.).
  - Example: (1000, 1, 1, 20/2013, 1, 2012-07-29 00:00:00, 1223981, ..., 20747, etc.)
- **Team:** (id, team_api_id, team_fifa_api_id, team_long_name, team_short_name).
  - Example: (43042, 8634, 241, FC Barcelona, BAR)
- **Player_Attributes:** (id, player_fifa_api_id, player_api_id, date, overall_rating, potential, preferred_foot, attacking_work_rate, etc.).
  - Example: (6176, 200810, 154238, 2014-09-18 00:00:00, 61, 66, right, medium, etc.)

We put a considerable amount of time into transforming the data into something useful. This included creating a Python script to run the SQL queries to turn the data into a dictionary full of features which would then be turned into a vector of features on which learning took place.
A particularly interesting phenomenon that took place when we were trying to put the data into feature vectors was that our test feature vectors and train feature vectors would end up having different lengths. The reason was that as it turned out, features such as formations would take on different sets of values during different seasons which led to us allocating a different number of slots to a particular feature for two different data sets. As a result, we had to create our feature vectors by first creating a mapping function from an attribute name to its position on the feature vector.

This would mean slightly modifying feature names that had string values to include both the feature name and its value, for example away_formation became away_formation_4-3-2-1, away_formation_5-3-2, or any other concatenation of away_formation and possible formations.
In addition, we are also adding a number of features ourselves which might be helpful. These include team league standing/score and time since the last game. These features were not readily available and had to be extracted using a script.

Methods

The learning algorithms we used include two-layered SVM, SVC, linear SVC, and softmax regression. To allow ourselves to work more on the high level problem rather than getting stuck in the details, we mainly used scikit and numpy libraries on python. Here is a semi-detailed explanation of these algorithms.

- **Two-layered SVM:** In two-layered SVM, we first trained a model based on whether the results of the training matches were home wins or not. We also trained a second model on whether the result of a match was home loss or draw. Prediction on test set was done first using the first model to determine if a given test match would end in a home win or not. If not, we would then predict the outcome of the match using the second model to determine whether the result was a draw or home loss.
  The function we used for this was sklearn.SVC. Given $x_i \in \mathbb{R}^p$, $i = 1, ...n$ and $y_i \in \{1, -1\}$, SVC solves the following problem:
  $$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$
  subject to $y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$, $\zeta_i \geq 0, i = 1, ..., n$, where C is the regularization parameter in this problem. We should note that this problem is equivalent to solving the more familiar $min_\alpha \frac{1}{2}\alpha^T Q\alpha - \mathbf{1}^T\alpha$ subject to $y^T\alpha = 0$ for $0 \leq \alpha_i \leq C$, where $Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$.

- **SVC:** The only difference between this algorithm and the one described above is that this time we used this function for multi-class classification. To implement such functionality, SVC uses the one-against-one approach for multi-class classification in which for every pair of classes (in our case home win, draw, away win), a model is created and upon testing, the new test vector is checked against all the models and the winning class gets a +1 score. Finally, the class with the highest score becomes the prediction for that test case. A wellknown downfall of this algorithm is when all classes get the same score.

- **linear SVC:** Linear SVC is similar to SVC but it uses the multi-class SVM formulated by Crammer and Singer. This algorithm solves the following problem:
  $$\min_{\mathbf{w}_m \in H, \xi \in \mathbb{R}^l} \frac{1}{2} \sum_{m=1}^{k} \mathbf{w}_m^T \mathbf{w}_m + C \sum_{i=1}^{l} \xi_i$$
  subject to $w_{y_i}^T \phi(\mathbf{x}_i) - w_t^T \phi(\mathbf{x}_i) \geq 1 - \delta_{y_{i,t}} - \xi_i$, where $i = 1, ...l$ and $t \in \{1, ..., k\}$. Note that $\delta_{i,t} = 1\{i = t\}$. Note that k is the number of classes (3 in our case) and l is the number of examples.
  To describe this algorithm a little bit, note that the expression to be minimized is the standard SVM expression. However, the constraint looks a little bit more complicated which it actually is not. We have for
  - $y_i = t$ that $0 \geq 1 - 1 - \xi_i \Rightarrow 0 \leq \xi_i$, and for
  - $y_i \neq t$ that $w_{y_i}^T \phi(\mathbf{x}_i) - w_t^T \phi(\mathbf{x}_i) \geq 1 - \delta_{y_{i,t}} - \xi_i$ or $w_{y_i}^T \phi(\mathbf{x}_i) - w_t^T \phi(\mathbf{x}_i) \geq 1 - \xi_i$.

  The two conditions above ensure that ξ's are positive and that we would like the winning prediction class for $\mathbf{x}_i$ to be at least $1 - \xi_i$ higher than the score for any other possible class. It turns out that this algorithm is more computationally expensive than the two previously described but do not suffer the downfall of one-versus-one selection algorithm.

- **Softmax regression:** Softmax regression is extremely similar to logistic regresssion with the difference that the classification problem now involves more than two classes. The associated cost function is
  $$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\} \log \frac{\exp(\theta^{(k)^T} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)^T} x^{(i)})}\right]$$

$$= -\left[\sum_{i=1}^{m} \log \frac{\exp(\theta^{(y^{(i)})^T} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)^T} x^{(i)})}\right] = \sum_{i=1}^{m} \left(-\theta^{(y^{(i)})^T} x^{(i)} + \log \sum_{j=1}^{K} \exp(\theta^{(j)^T} x^{(i)})\right)$$

Minimization of the cost cannot be done analytically and will be carried out using an iterative approach such as gradient descent.

- **Feature selection:** In this project, the number of features grew to be more than the number of training examples, so we suspected only a subset of the features was relevant and necessary for learning. To obtain the subset of useful features and to avoid overfitting, we ran the feature selection algorithm on the training data. In particular, we used forward search to maintain a current subset of features that minimizes the cross validation error. We added features one by one to the list, and in every iteration the optimal feature that minimizes the validation error was added to the list.

Results

First, whilst running forward search, we interestingly picked out '`awayTeamLeaguePosition`' and '`homeTeamGameHistoryScore`' frequently, which intuitively are features representing how well a team is doing. Due to the large number of features that we had, we also tried restricting our features to a smaller set, removing all features related to players, and running feature selection on this set of features, which led to better results. We think that the poorer test errors that can be seen in figure 2 compared to figure 1 is due to the inclusion of player attributes as features, which seem to be far too specific. In particular, linear SVC managed to pick out particularly bad player attributes (such as out field player's goal keeping abilities) that happened to have a good correlation with the validation set and gave them a high weighting, which we think caused the very spiky training and test errors.

Because we had a large data set to draw examples from, we found hold-out cross validation to be sufficient, and for this cross validation we split our overall training set in a ratio of 70/30 to give a validation set. From this phase of our implementation we concluded that softmax regression performed best, with a set of 30 features, `[awayLeaguePosition, homeGameHistoryScore, homeFormation-4-2-3-1, homeTeamName-Manchester United, away_team_-chanceCreationPassingClass-Normal, ...]`. This model gave a 48% error on our training set.

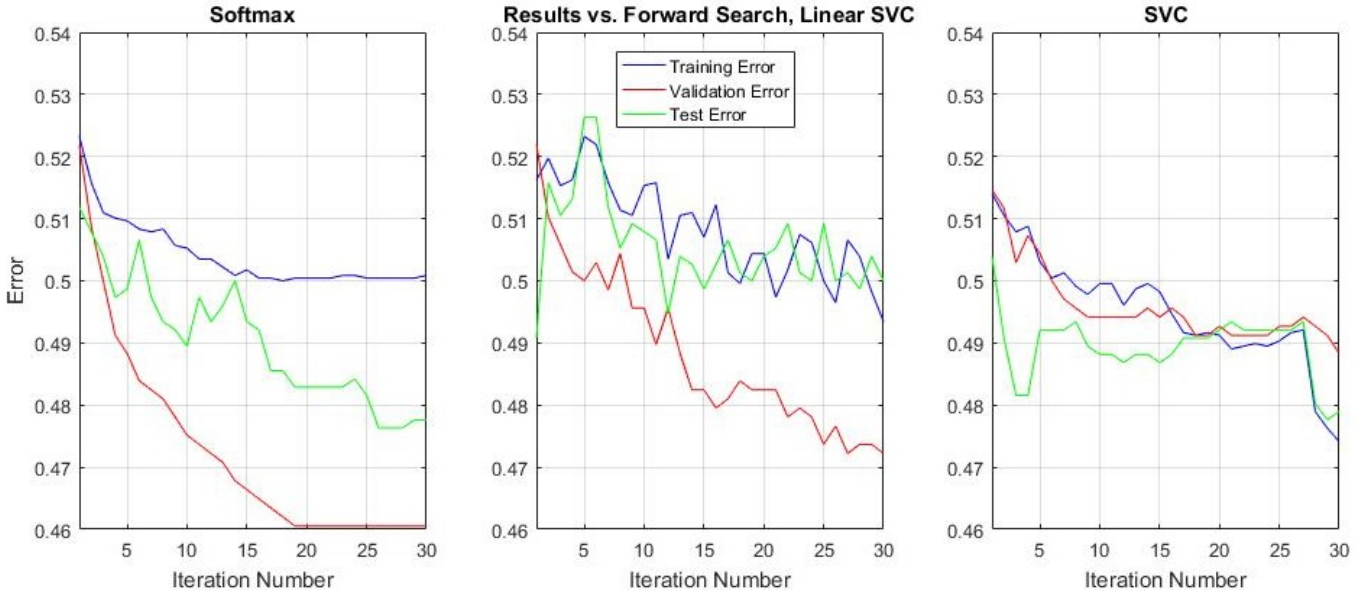

Figure 1: The training, validation and test errors for each model with respect to the number of features selected, when features about specific players were *not* included.

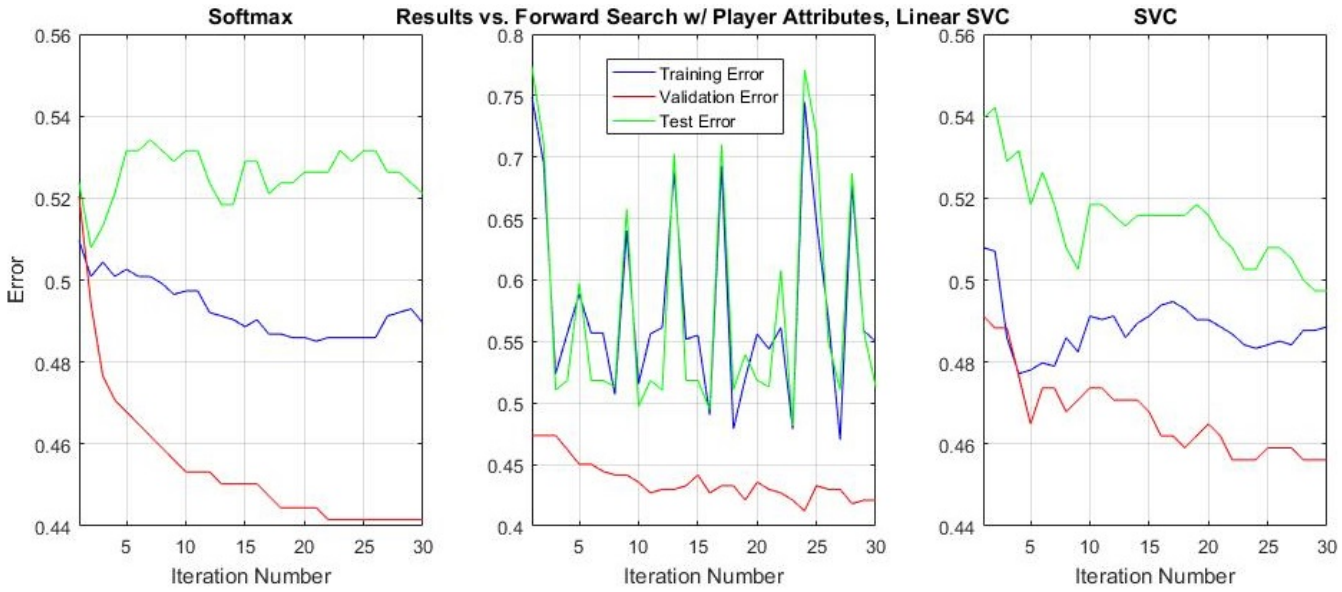

Figure 2: The training, validation and test errors for each model with respect to the number of features selected, when features about specific players *were* included.

During training we also used an L2 regularization term, and so we had one hyper-parameter 'C' that we were able to tune. As we only had one hyper-parameter we used 'grid-search' (or just 'line-search' really) to find a good value of C. We found that a value a little below 0.1 worked best with the softmax regresson model we found as can be seen in figure 3. This squeezed an additional 1% of performance out, giving an error of 47.6%, as seen in figure 1.

Finally, the confusion matrix is provided in table 1, and the precision and recalls for each class of our final model is provided in table 2.

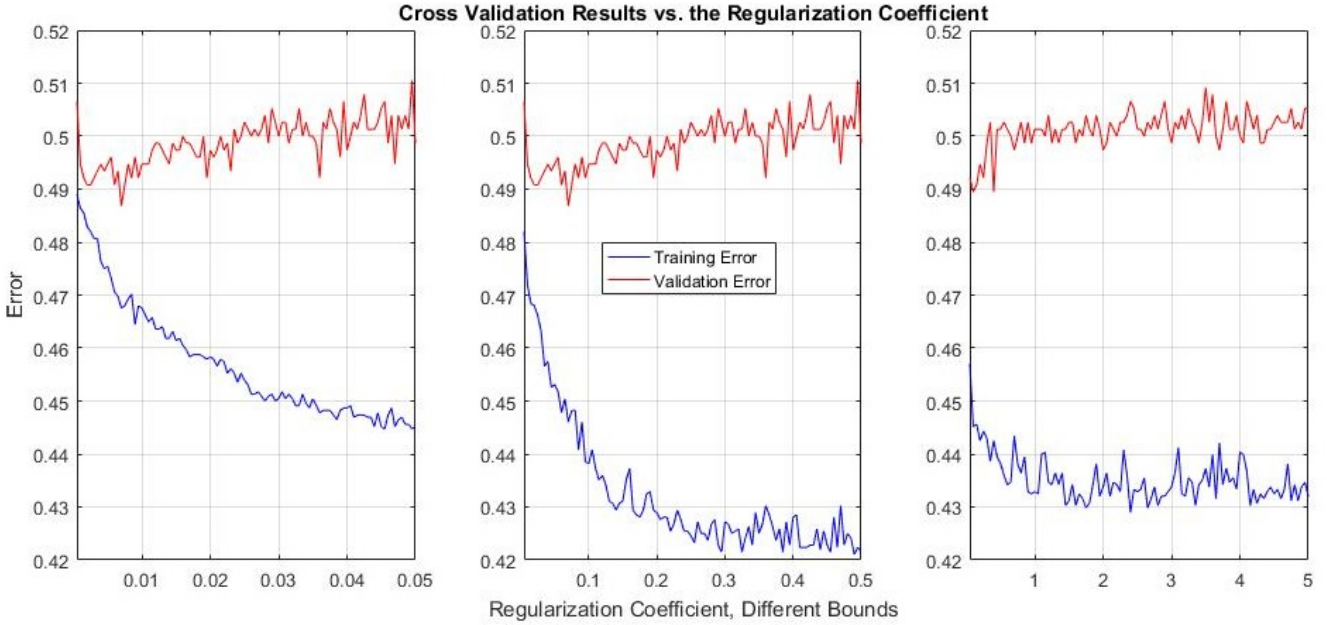

Figure 3: The training and validation error of our softmax regression model, as we tuned the value of the hyper-parameter C.

		Actual		
		Win	Draw	Loss
Predicted	Home Win	166	70	49
	Draw	3	6	1
	Away Win	23	16	30

Table 1: The confusion matrix of the final model.

Class	Precision	Recall
Home Win	$\frac{166}{166+70+49} = 0.582$	$\frac{166}{166+3+23} = 0.862$
Draw	$\frac{6}{3+6+1} = 0.6$	$\frac{6}{70+6+16} = 0.066$
Away Win	$\frac{30}{23+16+30} = 0.431$	$\frac{30}{49+1+30} = 0.434$

Table 2: The precision and recall values of each class.

We made a few observations during the implementation of our project. We found that if we shuffled the match data, in an attempt to make the model agnostic to the date on which the matches were played, the performance of the models were marginally worse. So all of the above training was completed using an ordered training set and test set, which is realistic of how a model such as this may want to be used anyway.

Another observation that we made was that if we increased the size of the training set too much, then the performance of the model tended to be worse, as can be seen in figure 4. Because of this, we restricted the size of the training set throughout our model selection above.

Figure 4: Example of training and test errors of one of our models with respect to the size of the training set.

Conclusion

The main goal of this project was to predict the outcome of soccer matches. This was broken down into two main branches, finding the best features to be used for prediction and establishing the most appropriate algorithm for it. To find the best features, we ran forward search and found around 20 to 30 features to be optimal in terms of validation error. We ran forward selection using a number of algorithms, among which Softmax proved to be the most precise algorithm, leading to about 47% error. Although being only slightly better than random, this algorithm bodes really well compared to other algorithms used, and competes with a fair amount of literature values. However, there are still a number of ways to improve upon this result as outlined in the following section.

Future Work

There are a number of fronts we could explore given more time and computational power which are as follows:

- Applying other machine learning algorithms to the data set, particularly neural networks.
- Use features used in successful literature reviews to get better accuracy levels.
- Try a larger range of training sets and find the optimal time to start prediction during a given season.
- Produce betting odds and find the expectation of money won using the optimal strategy.

References

[1] Schumaker, R. P., Jarmoszko, A. T., & Labedz, C. S. (2016). Predicting wins and spread in the Premier League using a sentiment analysis of twitter. Decision Support Systems.

[2] Godin, F., Zuallaert, J., Vandersmissen, B., De Neve, W., & Van de Walle, R. (2014). Beating the Bookmakers: Leveraging Statistics and Twitter Microposts for Predicting Soccer Results. In KDD Workshop on Large-Scale Sports Analytics.

[3] Magel, R., & Melnykov, Y. (2014). Examining Influential Factors and Predicting Outcomes in European Soccer Games. International Journal of Sports Science, 4(3), 91-96.

[4] Kumar, G. (2013). Machine Learning for Soccer Analytics.

[5] Heuer, A., & Rubner, O. (2012). Towards the perfect prediction of soccer matches. arXiv preprint arXiv:1207.4561.