European Soccer Proposal

1. European Soccer Database: <https://www.kaggle.com/hugomathien/soccer>
2. The dataset contains 7 tables: countries, leagues, matches, players, players attributes, teams and teams attributes. The data goes from 2008 to 2016.   
   It is an incredibly large dataset that will be fun to unfold. I’ll describe briefly what each table contains.   
     
   In the countries and leagues tables, we find the 11 European countries and their corresponding soccer leagues, respectively.   
     
   The matches table has a large number of attributes, more than 130, because it also includes several information regarding several bookmakers and their odds for many of the games in the table. An example of odds present in the table, are the ones given for a team to win or draw a specific game. Other attributes in this table include specific data regarding the 10,000+ matches that occurred from 2008 to 2016, such as scores, yellow cards, possessions, etc…   
     
   The player table is a simple table containing basic info of the soccer players in the 11 leagues mentioned. The attributes for this table are the players’ names, their birthdays, their height and weight of the 11,000+ players present in the dataset.   
     
   The players attributes table instead contains a much more detailed representation of each of the players present in the player table. There are more than 40 attributes in this table and they include information such as overall ratings, passing ratings, stamina, strength, and so on.  
     
   The team table is another simple table with only a few attributes (5) including the name of the teams and their abbreviated versions.  
     
   The team attributes table has 25 attributes with the statistics of each team in regards of the games played. These include chance creating by passing, crossing, or shooting, defense pressure or aggression, and so on.
3. The goal of my analysis is to answer the following question: how safe is it to bet on the home team, when the odds are in their favor? Can we have a further indicator on when we should play the home team?  
      
   In other words, is it possible to come out with a positive return if the bets were placed strategically on the home teams when they are in fact the favorited ones in a particular game and certain indicators are in place?

For this analysis I will use three types of classification: LDA, QDA and KNN. The attributes in my final matrix will be the following:

"bet”, "buildUpPlaySpeedClass", "buildUpPlayDribblingClass", "buildUpPlayPassingClass”, "buildUpPlayPositioningClass", "chanceCreationPassingClass", "chanceCreationCrossingClass", "chanceCreationShootingClass", "chanceCreationPositioningClass", "defencePressureClass", "defenceAggressionClass”, "defenceTeamWidthClass", "defenceDefenderLineClass", "B365H" "B365D", "B365A"

We can see there are 12 features strictly related to each of the home team’s characteristics, 3 features that are related to the odds of each game, where B365 is the broker and H stands for “Home” indicating the home team odds of winning. Suffixes “D” and “A” stand for “Draw” and “Away” indicating the corresponding odds. Finally, we have our classification labels that represent “0” in case the bet was not a winning one, and 1 in case the bet was a winning one. A winning bet implies that the condition “home team winning + the odds given by the broker being in favor of the home team” was met, and vice-versa with a non-winning bet.

I will use three classification methods that include LDA, QDA and KNN. I decided to use these methods because they’re simple and usually pretty efficient. I wanted to use three different one, because I did not know ahead of time which one of these would have given me the best model.

This analysis will give us confidence in try to predict and compare the actual results of the test set. Since soccer games do not follow a clear pattern, the models generated will not be as accurate as they would be with a broader set of attributes that can give us information regarding money invested that in players, position in the league at the time of a specific game, and so on. Being my own personal analysis and within the scope of the class, I want to obtain a result that will have at least 55% accuracy on the test and validation sets. I am aware that this is not even close to what a good model is, but for my dataset and my personal goal, it meets my requirements.

To start, I used the match table. First thing I did was to delete all the unnecessary attributes: team\_fifa\_api\_id, country\_id, league\_id, date, all the players ids which is represents 67 columns/attributes, data that is available before a game, such as number of goals, shots on target and off target, fouls commited, red and yellow cards, corsses, corners, and possessions. Moreover, I removed the other nine brokers’ odds and left the Bet365 ones. I created a new attribute to identify if the home team was the winner one. This was done by simply comparing the home\_team\_goals vs. the away\_team\_goals for each game. This attribute was used to create the classification labels in combination with the B365H odds being lower than the B365D and B365A.

At this point I started working on the team\_stats table. This table contains all the other attributes that we need to perform out analysis. The goal is to merge this table with the match table and clean up the extra attributes that I don’t need. I changed the attribute used to identify each team within the team\_stats to match the home\_team\_api\_id used in the match table. This is being used as the common attribute to merge the two tables.

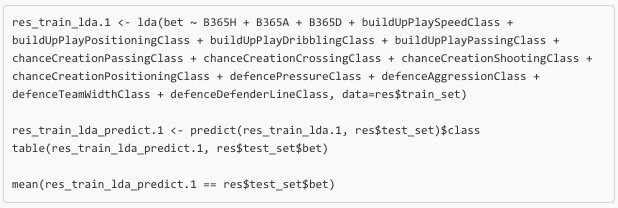
After I merged the two tables in a new table called m, I wanted to two things mainly. One is to remove the extra attributes, and the other is to convert the categorical value to numeric. The latter step is needed in order to normalize the set. The attributes I want to remove are a numerical representation of other class attributes present in the data frame. An example is “buildUpPlaySpeed”, which can be from 0 to 100, but only has 3 categories, which can be seen in the “buildUPlaySpeedClass”. To be more clear, 0-33 is considered Slow, 34-66 is considered Balanced, and 67-100 is considered Fast. Therefore, I deleted the numerical representations of these class attributes. I converted the factor attributes to numeric ones by assigning value between 0 and 1, depending on how many classes there were per each attribute. Using the previous example, if “buildUPlaySpeedClass” had three classes, I assigned Slow to 0, Balanced to 0.5, and Fast to 1. These attributes all have either 2 or 3 classes, making it easy to split between 0 and 1. After this naming conversion, I did the actual value conversion from factor to numeric and stored it back into the appropriate attributes. I also removed the home\_team\_api\_id used previously.

At this point, I wanted to make sure to omit any row with empty data so I ran an na.omit() on the m table.

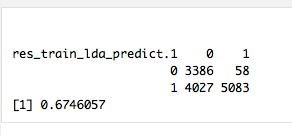
The next step is to divide the data into training, testing, and validation sets. In order to be more consistent in case something went wrong, which it did many times, I set the seed to 1 and perform the random dataset division. I assigned 60% to training, 30% to testing, and 10% to validation. I stored the labels in new vectors for each of the sets I just created.

At this point I wanted to normalize the training data. I performed this only on the training data because we do not know if the data given in the future will be normalized. This way we train the model more accurately, but we test it on data closer to a real scenario. In order to normalize the training set, I removed temporarily the labels, I performed the normalization using the function scale() and I placed the labels back into place. The reason why I did this is because I wanted the labels to still represent 0’s and 1’s.

The set was ready to be used for the linear discriminant analysis. This analysis is quite simple and only requires a few lines of code:

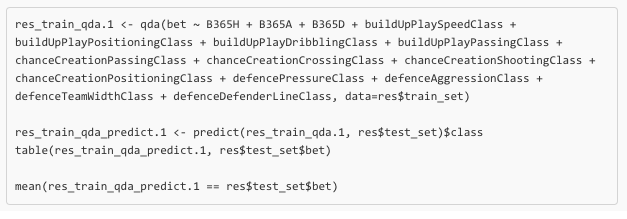


The first parameter is my labels from my training set. The ~ includes all the groups of interest within the dataset, which for us is the odds and all the classes we discussed earlier, finally we need to specify the dataset where the data is pulled from. Once I trained my model I used the predict() function to see how it would perform on my test set. Here we just need to provide the trained model and the new dataset on which we want to perform the classification. I then displayed the resulting table and the accuracy:

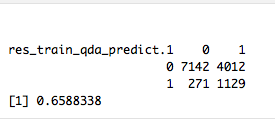


With this this model I obtained a 67.5% accuracy. As I mentioned before, this is not a great result, but it meets and exceeds my expectations.

Next I performed a Quantitative Descriptive Analysis. It follows the same steps as the LDA the code snippet looks like this:

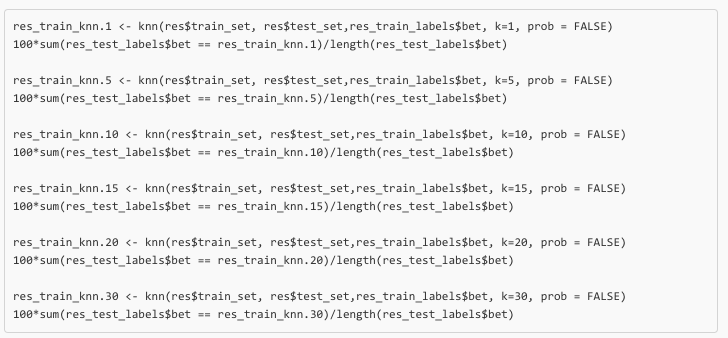


As it’s visible the steps and parameters are the same as previously. The table and accuracy displayed looked like this:



With this model we obtained an accuracy of 65.9%, which is not as good as the one obtained from the LDA model.

Lastly, I performed a K-nearest-neighbor analysis. For this analysis I took out the labels from within the data set as knn requires them to be in their separate vector. I will use the vectors I created earlier for this purpose. I tried several k values to see which one would give me a better estimation rate. Here is a code snippet of the analysis:



In KNN the parameters used are the training set, the test set, the labels for the training set, the number of neighbors considered (k), and if we want to return a new attribute “prob” with a portion of the votes for the winning class, which I do not want. Then we can easily output the ratio of correct predictions as showed above. The results were respectively:

K=1

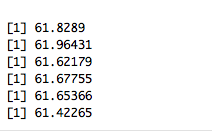
K=5

K=10

K=15

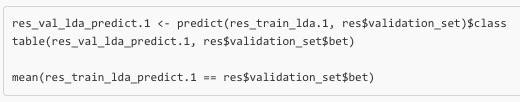
K=20

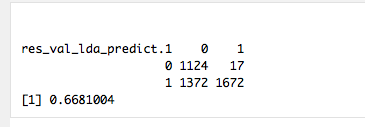
K=30



KNN did not give us better results than either LDA or QDA. The best outcome was with k=5 with a 61.96% accuracy.

Now that I know for sure, which one of the 3 models has the best outcome on the testing data, I can try to run the LDA model on the validation set. I followed the same steps as shown earlier:

The output was the following:



The results indicate a 66.8% on the ‘unseen’ data, which is fairly close to what the results on the testing data looked like (67.4%).

In conclusion of my results, I can state that LDA was the most efficient out of the three methods used. The results on the ‘unseen’ data were above my expectations with a 66.8% compared to the 55% I was aiming for. Strategically, this type of model can be used to predict ideally 2/3 of the games in Europe by using the limited number of attributes proposed, which will result in a guaranteed return into the pockets of the person betting on the games.