Predicting the Outcome of Soccer Matches

Description of the problem

Soccer is one of the most unpredictable sport. In facts, there are so many variables that can be taken into account that you can never be sure of the outcome of the match. Our goal is to take up this challenge and to find the best Machine Learning model to predict the outcome of football matches. In football matches, the home team wins about 46% of the time so our model needs to do better than this.

Description of the dataset

To help us in this task, we have a dataset from Kaggle. This dataset contains a lot of information such as the following:

* +25,000 matches
* +10,000 players
* 11 European Countries with their lead championship
* Seasons 2008 to 2016
* Players and Teams' attributes\* sourced from EA Sports' FIFA video game series, including the weekly updates
* Team line up with squad formation (X, Y coordinates)
* Betting odds from up to 10 providers
* Detailed match events (goal types, possession, corner, cross, fouls, cards etc…) for +10,000 matches

A large part of the available data won’t be used as we will mainly focus on the match table. I also decided to focus on a single football league that will be the England Premier League.

Data exploration

So as we said, we will only use the match table to realize our task. The match table contains a lot of information and its dimension is 115. Of course we will not use everything from this table, so here are the attributes that I found that would be useful:

* season
* date
* home\_team\_api\_id
* home\_team\_name
* away\_team\_api\_id
* away\_team\_name
* home\_team\_goal
* away\_team\_goal
* shoton
* shotoff
* foulcommit
* card
* cross
* corner
* possession
* (betting odds from different bookmakers)

Data preprocessing

First, I started by averaging all the betting odds into three columns (home win – draw – away win). Doing this I would have a global vision of what the bookmakers are thinking about the outcome of the match.

Then, I needed to extract the data from the XML string we have for each statistic in the match (shoton, shotoff, corner, possession…). This was by far on of the most complicated task to do as I didn’t know how to extract the data from an XML format in Python. Moreover, I didn’t only have to extract a single information but several in several variables. Take the following as an example from a corner xml string:

Text

Description automatically generated

Here we can see that for each value we have different information such as the number of corners (which is always 1, 1 value = 1 corner), the time elapsed in the match at the time of the corner etc.. And in red we can see the team who actually got the corner. So for each cell in the corner columns, I had to check each value in the XML string to check which team actually got the corner and then store the total value for each team. And this for every feature concerned.

After obtaining the concerned value for each team, I subtract them to obtain a single value which will be easier to use in the future.

This took me a lot of time and it was sometimes painful not understanding why I got NaN or wrong values back from the function as I never manipulated XML in Python before.

Fortunately, once I did this, I had done the most of preprocessing the data. I added a result column containing three possible values:

* “Win” if home\_goals > away\_goals
* “Draw” if home\_goals = away\_goals
* “Lose” if home\_goals < away\_goals

This column will be our future target for our model.

Finally, I removed any rows containing any N/A value to avoid any problem when building the model. I decided to do this because there was only 6 missing values (out of 3040) so I was actually not losing a lot of data.

Analysing Data

Now that we have a clean dataframe to work with, we can analyse it. Here is the description of the actual data:

Graphical user interface, text

Description automatically generated

Finding the best model

To find the best model for this dataset, I decided to use KFold and Cross Validation in order to get an idea on how each model would perform. Here are the model that I tested in the first place:

* Logistic Regression
* Linear Discriminant Analysis
* Decision Tree
* Gaussian Naive Bayes
* Support Vector Machine
* Random Forest

And as I figured it out, several of those models were not adapted for this task.

I decided to choose Linear Discriminant Analysis and Support Vector Machine as they seemed to be the best models for the data I was using. I tuned both models to obtain the best parameters to work with.

Support Vector Machine (SVM)

Using SVM, I made several versions by changing the number of data or the features. Every version was trained with a 80/20% split of the data.

Using the whole dataset

For this first iteration, I took the data as I prepared it before without any changes. The result was already quite good with an accuracy score of nearly 58% on the test set but I knew I could do better.

(CONFUSION MATRIX)

Using the best number of matches

One of the thing than can be improved is the number of matches that we are taking into account to predict the outcome. That’s why I made a function searching for the best number of past matches to take into account. This function was actually really useful and I found that for this model (SVM), the best number of matches was 1010 (out of 3034). Retraining the model with this new split of the data gave me insane result with an accuracy score of 67.33% on the test set. At this moment, I told myself that this result was a clearly higher than what I expected and I started to search where I could have did something wrong.

(CONFUSION MATRIX)

Removing the betting odds features

After having such a good result, I decided to remove those features than comes from bookmaker. Those numbers are not directly depending on the match and I was afraid that it would have a negative impact on the model. In this third iteration, I trained the model only with statistical data coming from matches (corners, shot on target etc..) and using the best number of matches. This time, the accuracy score was about 59.9% and I was pretty happy with that.

(CONFUSION MATRIX)

Linear Discriminant Analysis