Football Predictions Capstone Project

Predicting football games in Spanish La Liga

Paul Salaberria
October 31st, 2017

I. Definition

Project Overview

During the last decades, football has become the most popular sport in Europe. The British Premier League, Italian Calcio, German Bundesliga, French League 1 and the Spanish La Liga are some of the most important leagues. There are millions of football watchers across the globe, including markets like China or India, which leads to numerous business opportunities such as TV rights, merchandising, betting, and many more.

I am especially interested in the bettings. Multiple betting companies operate both online and offline, enticing their users with an opportunity to become rich. For the latter to happen, however, there will also be users who lose all of their betting money. I would love to beat the betting systems by predicting the results of the upcoming games. For this project I am choosing the Spanish league because it is, in my opinion, the best league in the world. I could not find any existing research on Spanish league predictions, which makes my research even more interesting.

Nowadays, we have access to very rich information about football teams, players, and several other sensors. Humans are not capable of processing all this information without the help of computers. This is where machine learning comes into play. Once we have collected data from many inputs, we can train a model to generate predictions. Classification algorithms and neural networks are two of the options in this regard.

In this project I will first explain the problem I am trying to solve. The different type of metrics for evaluating the solution will also be discussed.

The analysis of the dataset will be performed in the *Analysis* section. In this section I will explore the data, decide which algorithms and techniques will be used, and define the benchmark model for measuring the performance of my results. “European Soccer Database” (Mathien, 2016) is the name of the dataset I will use for this project..

In the *Methodology* section I will talk about data preprocessing, implementation, and refinement.

In the last two sections I will discuss the results, and finalize the project with my conclusion.

Problem Statement

Many different predictions can be made for a football match. The number of goals per team, the top scorer of the game, who will receive a yellow card, et cetera. However, I would like to start with a simple prediction, which will predict the winner of the match. It will be a win/lose/draw classification problem. In addition, in order to know if it is a clear win, an obvious draw, or an evident lose, I want to get the probability of an instance being in each of the classes.

There are many multi-class classifiers in scikit-learn (“12. Multiclass and multilabel algorithms”, n.d.) that return exactly what I am looking for. Naive Bayes, Logistic Regression, or SVMs are some of these algorithms. Scikit-learn provides the functions *predict* and *predict_proba* for the mentioned algorithms. In the final dataset, all features will be numerical, and therefore supported by the classifiers.

Metrics

The metrics used for evaluation will be the ones available in the classification report of sklearn models (precision, recall and F1-score). The benchmark model will be generated based on the classification report of BWIN betting house predictions. This will be compared then to the results of my model. We are not using the accuracy for evaluating the models due to the classes being highly imbalanced.

We will use the following performance measures for gaining more insights about the model (“8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset”, 2016).

- Confusion Matrix: A breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned). This will help us visualize how good our model is at predicting different labels.
- Precision: A measure of a classifiers exactness. tp/(tp+fp) where tp = true positives and fp = false positives. “The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.” (“Sklearn.metrics.precision_score”, n.d.). If the

precision of our model is high, it means there will not be many false positives. It’s important for our model to have a good precision in order to be able to make safe bettings.

- Recall: A measure of a classifiers completeness. tp/(tp+fn) where tp = true positives and fn = false negatives. “The recall is intuitively the ability of the classifier to find all the positive samples.” (“Sklearn.metrics.recall_score”, n.d.). The recall is equally important for betting. We do not want our model to give us false negatives.
- F1 Score (or F-score): A weighted average of precision and recall. This is the best metric for measuring the performance of the model in my case. Both precision and recall are equally important when predicting football results.

II. Analysis

Data Exploration

The “European Soccer Database” (Mathien, 2016) is used in this project as the only datasource.

The chosen dataset was originally created by Hugo Mathien, a Kaggle user. He used a crawler (https://github.com/hugomathien/football-data-collection/tree/master/footballData) to collect data about teams, players and matches using an external API provided by EA Sports FIFA.

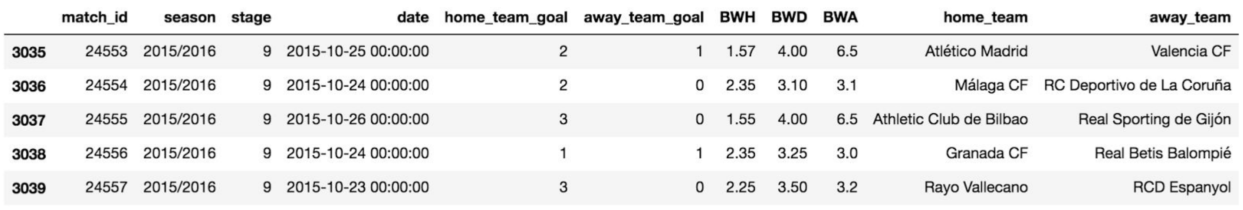
The dataset contains the following:

- +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

The dataset is a combination of tables (*Player_Attributes, Player, Match, League, Country, Team, Team_Attributes*) included in a sqlite database. However, we will focus on the *Match* table because it contains the most valuable information for our problem. In addition, we will only keep the matches belonging to the Spanish La Liga, which sums up to 3040 matches.

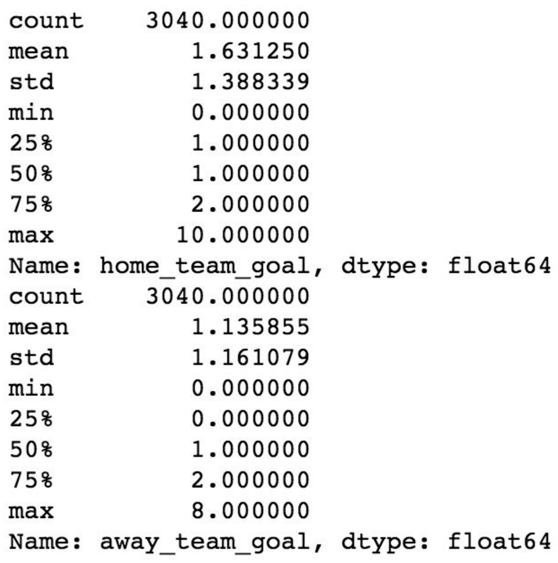
Some columns, such as betting odds from multiple bookkeepers and player stats, have been deleted in order to make the project more simple. I am only interested in the previous results between teams. See sample rows from the aforementioned table after the cleansing in Figure 1.



	match_id	season	stage	date	home_team_goal	away_team_goal	BWH	BWD	BWA	home_team	away_team
3035	24553	2015/2016	9	2015-10-25 00:00:00	2	1	1.57	4.00	6.5	Atlético Madrid	Valencia CF
3036	24554	2015/2016	9	2015-10-24 00:00:00	2	0	2.35	3.10	3.1	Málaga CF	RC Deportivo de La Coruña
3037	24555	2015/2016	9	2015-10-26 00:00:00	3	0	1.55	4.00	6.5	Athletic Club de Bilbao	Real Sporting de Gijón
3038	24556	2015/2016	9	2015-10-24 00:00:00	1	1	2.35	3.25	3.0	Granada CF	Real Betis Balompié
3039	24557	2015/2016	9	2015-10-23 00:00:00	3	0	2.25	3.50	3.2	Rayo Vallecano	RCD Espanyol

Figure 1. Sample rows from Match table after deleting unnecessary columns.

We get numeric data representing the number of goals for each team. The maximum number of goals for the home team is 10, and 8 for the away team. Minimum is 0 for both away and home. The mean is 1.63 for the home team, whereas for the away team the mean is 1.13 goals. See statistics about home/away team goals provided by pandas describe() function in Figure 2.



```
count    3040.000000
mean        1.631250
std         1.388339
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max        10.000000
Name: home_team_goal, dtype: float64
count    3040.000000
mean        1.135855
std         1.161079
min         0.000000
25%         0.000000
50%         1.000000
75%         2.000000
max         8.000000
Name: away_team_goal, dtype: float64
```

Figure 2. Statistics for home and away team goals in the dataset

The stage of the season is also numeric, and probably relevant for my machine learning algorithm. Some football teams have very good starts of seasons, whereas others improve their performance as the season gets to the end.

The teams are represented by strings. I will need to convert them into binary features, and add all existing teams to the columns list.

BWH, BWD, and BWA are betting odds from BWIN betting house. I will use these for generating a target column (bwin_target) containing the predicted result in terms of win/draw/lose. I will be able to benchmark my model against it afterwards.

My target variable will be 'home_team_result' in the format of win/draw/lose. The target column has been populated with a function that compares home team goals against away team goals. See the distribution of the target variable below (Figure 3).

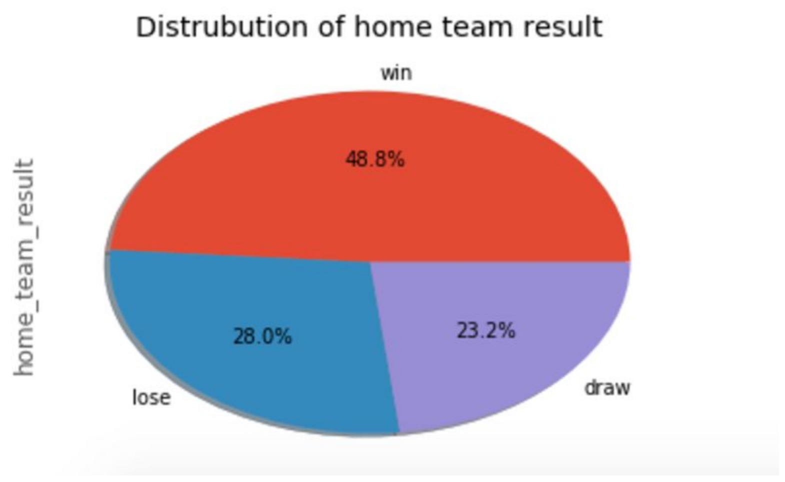


Figure 3. Distribution of home team result.

For each datapoint or match in the dataset I computed and selected 89 features. See the list of features used for training the model in Table 1.

Feature name	Description
stage	Stage in season. Integer from 1-38.
<team-name>_h	Binary feature for a specific football team. Equals to 1 if the team-name = home team, 0 otherwise.
<team-name>_a	Binary feature for a specific football team. Equals to 1 if

	the team-name = away team, 0 otherwise.
<home/away>_team_w_til_date	Number of victories for the home and away teams. Positive integer.
<home/away>_team_d_til_date	Number of draws for the home and away teams. Positive integer.
<home/away>_team_l_til_date	Number of losses for the home and away teams. Positive integer.
<home/away>_team_strike_w_til_date	Number of consecutive wins for the home and away teams. Positive integer.
<home/away>_team_strike_d_til_date	Number of consecutive draws for the home and away teams. Positive integer.
<home/away>_team_points	Number of points for the home and away teams. Positive integer.
<home/away>_team_pos	The position in the league for the home and away teams. Range from 1 to 20.
<home/away>_team_goals_favor_til_date	Number of goals scored for the home and away teams. Positive integer.
<home/away>_team_goals_against_til_date	Number of goals received for the home and away teams. Positive integer.
<home/away>_team_avg_goals_favor_til_date	Number of goals scored on average for the home and away teams. Positive integer.
<home/away>_avg_goals_against_til_date	Number of goals received on average for the home and away teams. Positive integer.

Table 1. Final features for training the model.

It is worth noting that the newly promoted teams can become outliers since there is no data from previous matches about them. If we are going to bet for newly promoted teams we should do it carefully.

Algorithms and Techniques

We can define the problem as a multi-class classification problem with 3 possible outputs: Win, draw or lose for the home team.

As previously mentioned in the *Problem Statement* section, there are many multi-class classifiers in *scikit-learn* (“12. Multiclass and multilabel algorithms”, n.d.). I will try a few of these, and choose the best performing model for the final benchmark. In order to find the best model, I will apply GridSearch with different algorithms and parameters.

See the list of algorithms I am planning to use below.

- MultinomialNB
- BernoulliNB
- LogisticRegression (multinomial)
- Support Vector Machine (SVC)

MultinomialNB and BernoulliNB are two popular Naive Bayes techniques which are often used as machine learning classifiers. They are normally used for text classification, and they assume features to be independent. The “unpredictability” of football results suggests that these classifiers may not be the best option for the given dataset. I will try to run the algorithms anyway just to get an idea of their performance, but I have very low expectations.

Logistic regression is another algorithm which is used as a classifier. It works by separating the inputs into two regions, and therefore it is suitable for binary classification problems. The core of the model is a logistic function in which each feature gets assigned a coefficient. The coefficients of the logistic regression algorithm are estimated from the training data. The technique used for finding the best values is called maximum-likelihood estimation. It tries to find the values for the coefficients that minimize the error in the probabilities of the predictions. As I mentioned previously, logistic regression works for binary classification. In this project we will use multinomial logistic regression, which generalizes logistic regression to multiclass problems. One of the reasons for choosing logistic regression is the training efficiency. Another interesting feature is that it returns the probabilities for a datapoint belonging to each class. A weakness for logistic regression is that it does not perform well when the feature space is too large. Having multiple categorical features can also be problematic. Our dataset is not too large, and we do not have many categorical features, therefore it seems like a good fit for our problem.

Support vector machines are a set of methods used for classification, regression and outliers detection in supervised learning. A support vector machine creates a hyper-plane or set of hyper-planes in a high dimensional space which are later used for the aforementioned tasks. The hyper-plane with the largest functional margin, the distance to the nearest training data points of any class, is preferred in order to avoid generalization errors (“1.4. Support Vector Machines”, n.d.). See Figure 4 for a simple linear solution. However, it is often the case where the separation between classes is not linear. SVM solves this problem by using different kernel functions such as polynomial, rbf, sigmoid or custom kernels (Figure 5).

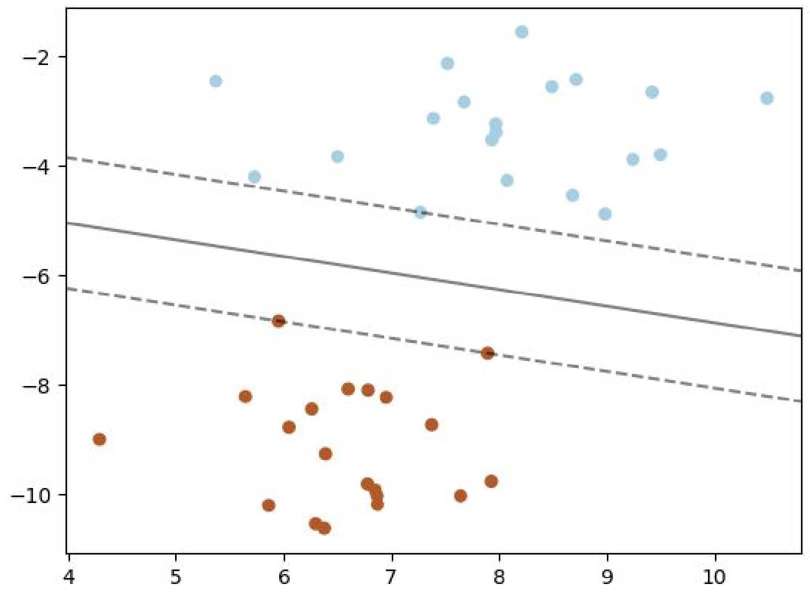


Figure 4. Support vector machine (“1.4. Support Vector Machines”, n.d.)

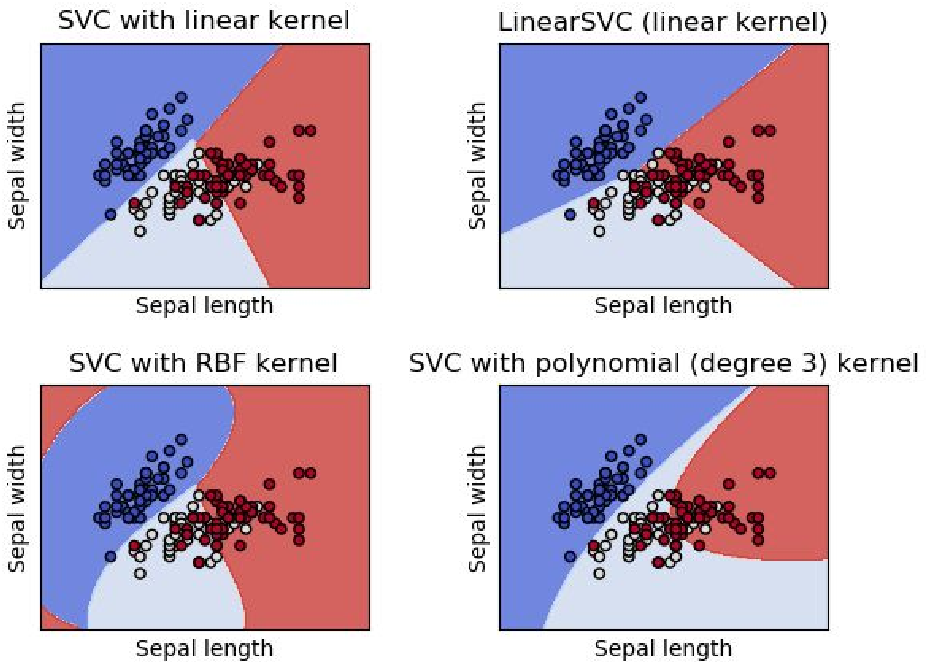


Figure 5. SVC with different kernels (“Plot different SVM classifiers in the iris dataset”, n.d.).

Sklearn provides multiple classes (SVC, NuSVC and LinearSVC) that are capable of performing multi-class classification on a dataset using SVM. SVC seems to be the most popular option, and therefore worth a try. It is also recommended by sklearn for non-text data classification problems ("Choosing the right estimator", n.d.). Support vector machines are effective in high dimensional spaces, memory efficient (use a subset of training points in the decision function), and versatile due to the possibility of using multiple kernels. The biggest downside of SVMs is that they do not directly provide probability estimates. These are computed using 5 fold cross validation, which increases computation time significantly.

"In general, learning algorithms benefit from standardization of the data set."("4.3. Preprocessing data", n.d.). Therefore I will apply feature scaling to the dataset features. Due to input restrictions of some of the algorithms, I will be using MinMaxScaler, which scales the features to lie between zero and one.

There is usually some correlation between certain features. We can use PCA to reduce the number of features and still maintain the accuracy of the model. This is very useful when running SVC since it speeds it up considerably. The more features we have the longer it takes to compute the results.

Many models suffer overfitting if they have only been trained and tested with the same set of data. This means they are very good at predicting already known data, but they fail at predicting unknown data. Cross validation ("3.1. Cross-validation: Evaluating estimator performance", n.d.) is a technique that tries to solve this problem. With cross validation I will make sure the model is trained and tested against different subsets of the dataset.

Benchmark

The bookmakers (BWIN) get the predictions right 56.18% of the time in the Spanish league. You can see the confusion matrix for BWIN predictions in Figure 6. We also know that the percentage of home victories adds up to 48.84%. If we bet constantly for home victory we will get an accuracy of 48.84%. Therefore, our goal is to get an accuracy at least higher than 48.84%, and hopefully better than 56.18%. Anything lower than 48.84% should be considered as a failure.

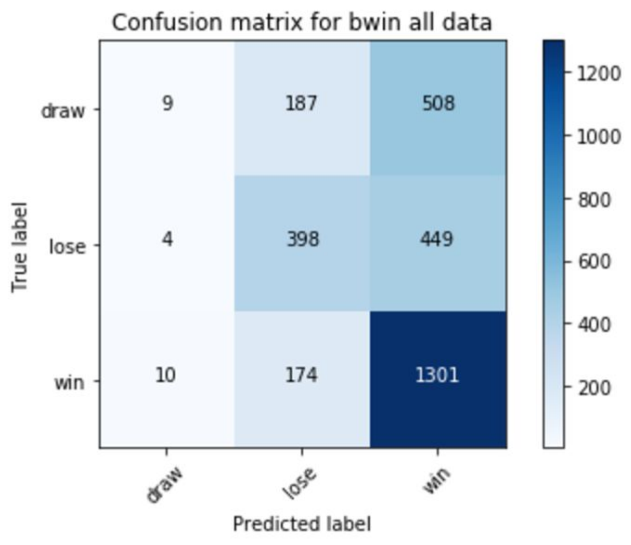
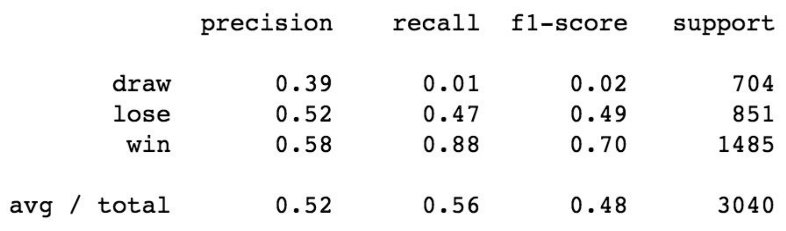


Figure 6. Confusion Matrix for BWIN predictions.

Due to the fact of having an imbalanced dataset, I will have to look at the metrics discussed in the previous chapter for a more detailed and fair comparison. Sklearn provides a class (“Sklearn.metrics.classification_report”, n.d.) for generating a report which returns the precision, recall and f1-score. Figure 7 shows the classification report for BWIN predictions.



```
             precision    recall  f1-score   support

       draw       0.39      0.01      0.02       704
       lose       0.52      0.47      0.49       851
        win       0.58      0.88      0.70      1485

avg / total       0.52      0.56      0.48      3040
```

Figure 7. Classification report for BWIN predictions.

The benchmark should be done based on the F1-score of the models, which in the case of BWIN’s model accounts to 0.48.

III. Methodology

Data Preprocessing

Machine learning algorithms expect data to be in a certain format. In the case of classifiers, they expect numeric features. My data is mostly numeric, and therefore only a few adjustments are necessary. These are the steps I followed when preprocessing the data and making it ready for the classifiers:

1. **Filter only Spanish matches via SQL**: The dataset contains matches from many different football leagues. Since I am only interested in a subset of those, I applied a filter when querying the SQL database to only fetch matches from the Spanish league.
2. **Keep only match results**: I only kept the columns related to match results in terms of goals. Player stats, and extra game stats were excluded for simplicity reasons.
3. **Add extra computed features**: I generated multiple new features using a python function. For each game, I added season stats to date such as win/draw/loss, strike of win/draw/loss, points per team, position in league, goals total and average for each team.
4. **Convert stage from string to numeric**: For sorting purposes, I had to convert the stage column into a numeric type.
5. **Convert teams into binary features**: String values for the home and away teams are not well suited for being used as input for machine learning algorithms.
6. **MinMaxScaler**: In order to speed up learning time and get more accurate results, I used a scaling technique that scales values to be between 0 and 1. Other scalers did not work in my case since MultinomialNB only accepts positive values.

Implementation and Refinement

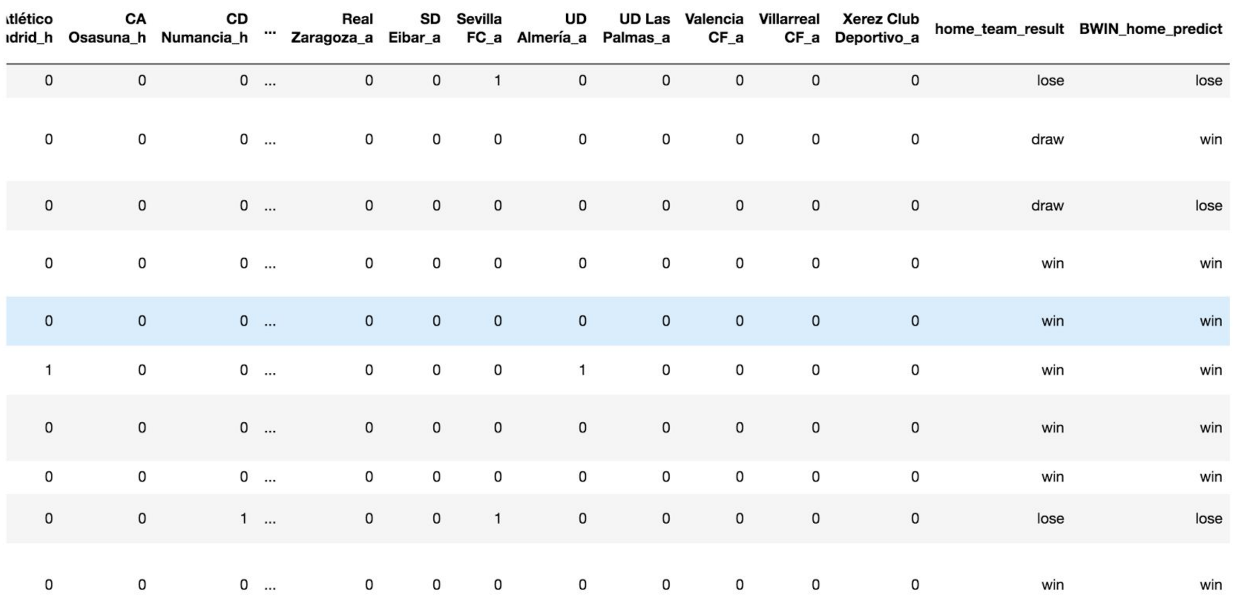
The most complex part of the project was the implementation of a python function that computes a snapshot for each stage in a season, containing statistics until that stage in mentioned season. I called it `create_season_stage_stats_til_date`. The function returns a dictionary with snapshots for each stage. I kept the dictionary in memory for later usage. See Figure 8 for an example output of the mentioned function. The keys for the main dictionary are seasons, for each season there is also a dictionary in which the keys are the stages, and for each stage there is a dictionary where they keys are the football teams, and the values are the statistics for the given teams. Changing values for nested dictionaries in Python was the most painful part of the project. I was able to overcome the issues using `collections.defaultdict` Python class instead of the normal dictionary.

```
{
  "2011/2012": {
    "0": {
      "Athletic Club de Bilbao": {
        "avg_goals_against_til_date": 0,
        "avg_goals_favor_til_date": 0,
        "d_til_date": 0,
        "goals_against_til_date": 0,
        "goals_favor_til_date": 0,
        "l_til_date": 0,
        "points": 0,
        "pos": 0,
        "strike_d_til_date": 0,
        "strike_l_til_date": 0,
        "strike_w_til_date": 0,
        "w_til_date": 0
      },
      "Atletico Madrid": {
        "avg_goals_against_til_date": 0,
        "avg_goals_favor_til_date": 0,
        "d_til_date": 0,
        "goals_against_til_date": 0,
        "goals_favor_til_date": 0,
        "l_til_date": 0,
        "points": 0,
        "pos": 0,
        "strike_d_til_date": 0,
        "strike_l_til_date": 0,
        "strike_w_til_date": 0,
        "w_til_date": 0
      }
    }
  }
}
```

Figure 8. Example output of ‘create_season_stage_stats_til_date’ Python function.

The next step involved the conversion of the home and away team names into binary features, where each combination of <team_name>_<home/away> represented a column in the new dataframe (Figure 9). If the game was Real Madrid against Barcelona, then the columns “Real Madrid_h” and “Barcelona_a” contained the value 1, whereas the rest of the columns contained 0. The implementation was done based on pandas functions.

Target columns also needed to be computed. `determine_home_result` and `determine_bwin_result` helpers functions were used in order to generate them. Both functions get a match as an input. The former compares home and away goals, whereas the latter compares the betting odds. Both target columns contained values from the following set: win, draw or lose. See Figure 9.



tlético adrid_h	CA Osasuna_h	CD Numancia_h	...	Real Zaragoza_a	SD Eibar_a	Sevilla FC_a	UD Almería_a	UD Las Palmas_a	Valencia CF_a	Villarreal CF_a	Xerez Club Deportivo_a	home_team_result	BWIN_home_predict
0	0	0	...	0	0	1	0	0	0	0	0	lose	lose
0	0	0	...	0	0	0	0	0	0	0	0	draw	win
0	0	0	...	0	0	0	0	0	0	0	0	draw	lose
0	0	0	...	0	0	0	0	0	0	0	0	win	win
0	0	0	...	0	0	0	0	0	0	0	0	win	win
1	0	0	...	0	0	0	1	0	0	0	0	win	win
0	0	0	...	0	0	0	0	0	0	0	0	win	win
0	0	0	...	0	0	0	0	0	0	0	0	win	win
0	0	1	...	0	0	1	0	0	0	0	0	lose	lose
0	0	0	...	0	0	0	0	0	0	0	0	win	win

Figure 9. Dataframe including target columns and a binary representation of the teams.

After adding the teams as binary features and adding the target columns, I looped through the list of matches, and added the necessary columns for the home and away teams based on the dictionary containing the stage snapshots . Win/draw/loss, strike of win/draw/loss, points per team, position in league, total goals and average goals for each team were the generated features.

Finally I was able to start trying out different machine learning algorithms.

The language and libraries I used in this project are Python 3, sklearn 0.18.1, pandas 0.20.1
, and numpy 1.12.1.

Machine learning algorithms expect the features and target to be provided separately. Therefore I took out ‘home_team_result’ and ‘BWIN_home_predict’ from the dataframe and saved them as target variables.

A common practice is to split the dataset into train and test data. We train the algorithm with a subset of our data, and test it against the remaining data. We do this to avoid overfitting. Sklearn offers a function called `train_test_split` for this purpose. I applied `train_test_split` to the features and target (‘home team result’), and set the test data to be 35% of the total.

MultinomialNB was the algorithm I wanted to try first. I fed the model with the train data, and ran predictions against the test data with the default settings. I got an accuracy of 48.21% in the first run. It was worse than a naive predictor that always predicts home victory. I tried Cross Validation with 5 folds on the full dataset, and the results were similar, 48.45% average. I needed to change something in order to improve the model. One of the techniques for improving the model is feature scaling.

I used a scaling technique called MinMaxScaler. It scales values to a range between 0 and 1. Other scalers did not work in my case since MultinomialNB only accepts positive values. I used a sklearn ‘Pipeline’ for combining the MinMaxScaler and the MultinomialNB classifier, and ran the same steps as for the non scaled data, also with the default settings, for fit and predict. This time I got better results. 52.63% in the train and test split data, and 53.81% with Cross Validation.

I still wanted to try more classifiers to see if I could get even better results. The next classifier that I used was SVC, a Support Vector Machine algorithm. With the default settings I got a prediction accuracy of 53.75% on average. SVC was much slower than the previous algorithms, so I decided to use PCA (Principal Component Analysis) for reducing the number of features and speeding up the training. I selected 8 as the number of components. This was a random choice. I did not look into the ideal number of components. The computing time decreased, and surprisingly I got even better results than before, with an accuracy score of 54.64% after using cross validation.

Still not satisfied with the results, I tried Logistic Regression. Using the default constructor, the score was 53.06% average, which I found quite disappointing, but the speed of the computation was much faster than SVC.

Additionally, I tried BernoulliNB, a naive bayes implementation, which gave 51.25% accuracy. This time I also used the default parameters.

My next step was attempting to tune the parameters of the best classifier until then, which was SVC. However, I was not happy with the training time, so I decided to tune parameters for both SVC and Logistic Regression. I used GridSearchCV from sklearn for running the algorithms with different combinations of parameters, and finding the best performing one.

For SVC I tried multiple C and gamma values (C=[0.001, 0.01, 0.1, 1, 10], gamma=[0.001, 0.01, 0.1, 1]), whereas for the Logistic Regression I used different C values (0.001, 0.01, 0.1, 1, 10, 100, 1000) and two different solvers, ‘lbfgs’ and ‘newton-cg’. The best combination after applying a grid search with 8 fold cross validation was the following: Logistic Regression(C=0.1,solver=’lbfgs’). The accuracy was indeed higher than before, with 55.07%. For SVC the combination with the highest accuracy score was SVC(C=0.1,gamma=0.001) with 54.7%.

In the following *Results* section, will analyze the results and compare them against the benchmark model.

IV. Results

Model Evaluation and Validation

As described in the *Implementation* section, the chosen final model was built using the following algorithm and parameters: Logistic Regression(C=0.1,solver='lbfgs') with MinMaxScaler. The model is able to predict results with 55.07% accuracy.

The model was chosen after analyzing MultinomialNB, SVC, BernoulliNB and Logistic Regression. SVC and Logistic Regression were the most accurate ones out of the box, therefore further optimizations were only applied to these two algorithms.

As mentioned in the previous section, GridSearch did the hard job of finding the best parameter combinations for the two aforementioned algorithms. Due to the slowness of SVC, I decided to focus on Linear Regression since the results from both models were very similar in the previous phase. The optimal parameters for Linear Regression after using GridSearch were C=0.1 and solver='lbfgs'.

C parameter is the inverse of regularization strength. Smaller values specify stronger regularization. The chosen value, 0.1, is lower than the default value which is 1.0. This means our model is highly regularized. We need to apply regularization for our classifier to not become too complex and overfit to our training data. Generalization is very important in our dataset due to the difficult predictability of football matches.

The selection of the solver is normally decided by the training time and the format of the data. In our case the training time difference was not significant between the different solvers. I could have used the 'sag' solver for this problem also, but from previous experience I found it to be slower than the other solvers. The score of the model was not affected by changing the solver.

The final model has been tested with various inputs to evaluate whether the model generalizes well to unseen data. Cross validation was used for feeding the algorithm with different input (training/test) sets. When using this technique we avoid overfitting.

An algorithm that predicts results correctly just above 50% of the time is difficult to trust. There are many factors in football that affect the game which are not possible to include in a mathematical function. Therefore, aiming at a 100% accuracy is not realistic. Depending on the use case, an accuracy of more than 50% would be enough for the model to be trustworthy. A betting system in which the betting odds are always "double or nothing", for example, would end-up losing against us if our algorithm's accuracy is higher than 50%.

However, the latter is not the case in present day bettings systems. More realistic metrics for evaluating our model are the confusion matrix, precision, recall and F1-score. I will compare the metrics of my final model with BWINs' in the next section.

Justification

If we only look at the accuracy of the model, 55.07%, we have fulfilled our minimum expectations of getting a better accuracy than constantly betting for the home team (48.84%). However, we have not been able to reach BWIN's results of 56.18%.

In order to get more insight about the performance of our model, and be able to perform a valid comparison with the benchmark model, we need to look at the confusion matrix (Figure 10, Figure 11) and the classification report for precision, recall and F1-score (Figure 12, Figure 13).

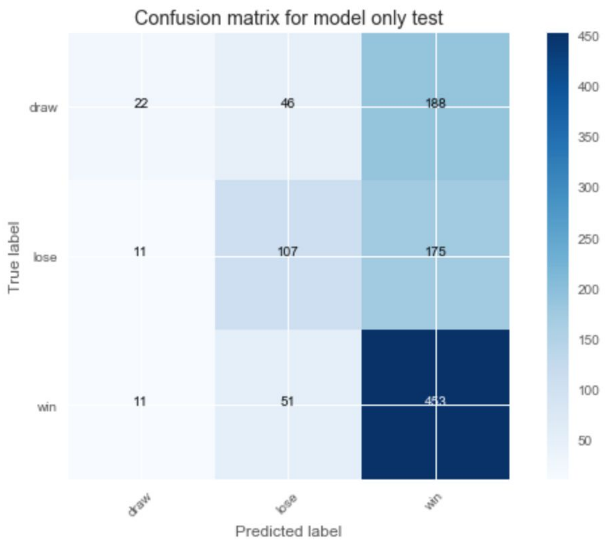


Figure 10. Confusion Matrix for our model predictions in test data.

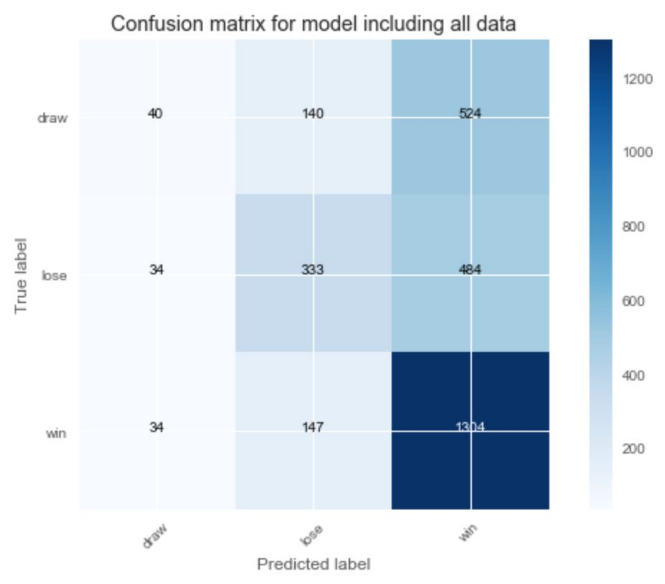


Figure 11. Confusion Matrix for our model predictions in whole dataset.

The confusion matrices of our model after predicting test data and the full dataset look very similar, which means our model does not suffer from overfitting. When comparing to BWIN's confusion matrix (Figure 6), we can also see the same patterns. My final model and BWIN's model seem to work in a similar way. It predicts a home victory most of the times, followed by home loss, and finally with a draw in rare occasions.

For more detailed statistics, we need to look at the classification report. See the classification report of our final model when predicting test data in Figure 12, and for predicting the full dataset in Figure 13.

```
             precision    recall  f1-score   support

       draw       0.50      0.09      0.15       256
       lose       0.52      0.37      0.43       293
        win       0.56      0.88      0.68       515

avg / total       0.53      0.55      0.48      1064
```

Figure 12. Classification report for our final model predictions in test data.

```
             precision    recall  f1-score   support

       draw       0.37      0.06      0.10       704
       lose       0.54      0.39      0.45       851
        win       0.56      0.88      0.69      1485

avg / total       0.51      0.55      0.49      3040
```

Figure 13. Classification report for our final model predictions in whole dataset.

In the *Benchmark* section I showed the classification report for BWIN predictions (Figure 7). I will now compare them below.

For each metric I will analyze the results for draw, win and lose.

When it comes to **precision**, my model is slightly worse than BWINs' when predicting draws and wins. Our score is 0.02 lower. However, when we look at loss predictions, our model outperforms BWIN's model by 0.02. Overall we get very similar results.

If we look at the **recall**, the results are also very similar between the two models. The recall for draw results is close to 0, which means the algorithms are not very good at guessing draws. My model has a slightly higher score, but I do not think it is significant. Home win predictions have a recall score of 0.88, which is the exact same number as in the benchmark model. The recall score for loss predictions is 0.08 lower in my model. The differences are not significant.

The **F1-Score**, a weighted average of precision and recall, confirms the similarities between my model and the benchmark. BWIN's model has a slightly higher score for win and lose, whereas for draws, my model gives better results. The average F1-score for BWIN is 0.48 compared to 0.49 for my model.

Based on the results discussed above, I am even with the BWIN benchmark. I am able to get very similar results as BWIN, but I cannot assure benefits against betting houses with my current model.

V. Conclusion

Free-Form Visualization

The most interesting finding of the project has been the similarity between BWIN's model and my own. We can see both confusion matrices side by side in Figure 14. I have not been able to overcome the benchmark model, but as I discussed in the previous chapter, the results in terms of accuracy, precision and recall are almost identical.

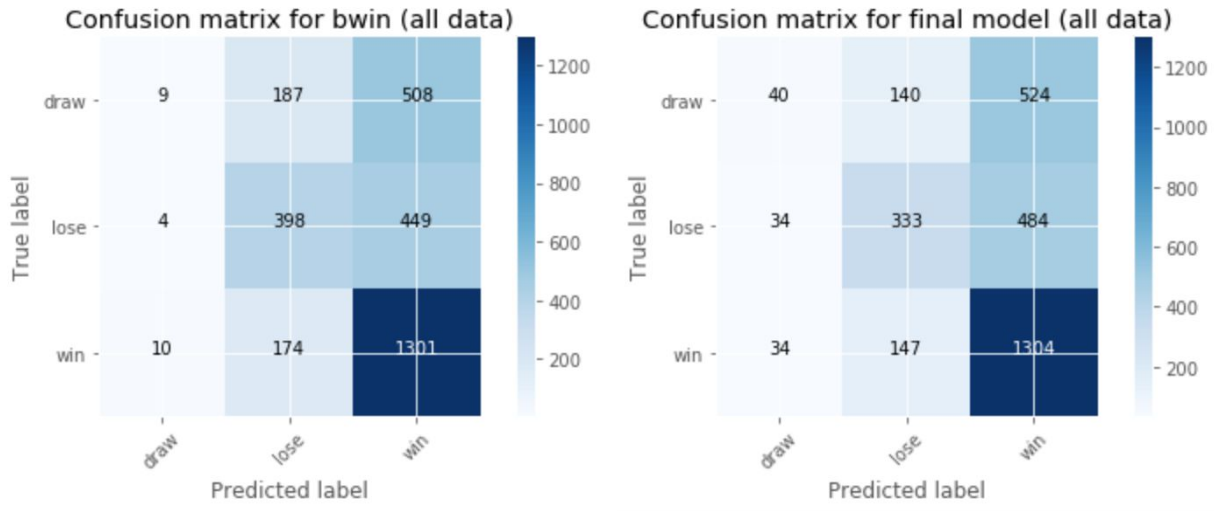


Figure 14. Confusion matrix comparison. BWIN vs our final model.

This research project lets us understand how betting houses may predict their results. I seem to be on the right track.

Reflection

The goal of the project was to predict football match results. I started by collecting the necessary data from Kaggle. The dataset was not ready to be used as the input for the sklearn classifiers, therefore I applied some pre-processing steps. I generated binary features for the labeled data, due to the classifiers only allowing numerical values as features. I got rid of unnecessary columns, and I computed new features from existing data in order to enrich the dataset. Feature scaling was also applied to the dataset, which significantly improved the performance of the machine learning models. Once the data was ready, I ran multiple classifiers and fine tuned the parameters of the best performing ones which helped me choose the final model. In addition to the implementation, I have utilized this paper to discuss the results and the comparison with the benchmark.

One of the aspects I found interesting with my project was the ease of use of sklearn algorithms. Once the dataset is in an expected format, it can be read by any kind of classifiers without extra work. This allowed me to run multiple of them and choose the best one with ease.

The similarities in the confusion matrices for BWIN and our model also deserve to be mentioned. The results are almost identical.

During the project there were also some difficulties. I had to write some Python functions which required certain programming skills and knowledge in order to enrich the dataset with “stage snapshots". Working with nested dictionaries in Python was more complicated than what I expected.

I should also mention that I found parameter tuning for different algorithms to be somewhat random and difficult to reason about.

The expectations I had before starting the project are satisfied by the final model. The accuracy of 55%, and the similarities with the benchmark model confirm this. However, it can still be improved. I will analyze some of the possible improvements in the next section.

Improvement

One possible improvement could be to add more features such as player statistics. This will require some pre-processing work, seeing as players may change teams within or after a season. Additional data for different leagues could also improve the model.

Another option would be to take a completely different approach, and use neural networks. However, due to the complexity of the configuration compared to classifiers I did not look further into the topic.

I could also try to predict the exact amount of goals for each team instead of a win/draw/lose, as this would provide much richer information. In this case, we could use multivariate linear regression or neural networks.

In other words, there is still room for improvement.

References

1.12. Multiclass and multilabel algorithms. (n.d.). Retrieved September 19, 2017, from

http://scikit-learn.org/stable/modules/multiclass.html

Choosing the right estimator. (n.d.). Retrieved October 24, 2017, from

http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

1.4. Support Vector Machines. (n.d.). Retrieved October 28, 2017, from http://scikit-learn.org/stable/modules/svm.html

Plot different SVM classifiers in the iris dataset. (n.d.) Retrieved October 31st, 2017, from http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html

3.1. Cross-validation: Evaluating estimator performance. (n.d.). Retrieved September 19, 2017, from http://scikit-learn.org/stable/modules/cross_validation.html

4.3. Preprocessing data. (n.d.). Retrieved September 19, 2017, from http://scikit-learn.org/stable/modules/preprocessing.html

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset. (2016, June 06). Retrieved September 17, 2017, from https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

Sklearn.metrics.classification_report. (n.d.). Retrieved September 17, 2017, from http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Sklearn.metrics.precision_score. (n.d.). Retrieved October 10, 2017, from http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

Sklearn.metrics.recall_score. (n.d.). Retrieved October 10, 2017, from http://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

Mathien, H. (2016). *European Soccer Database*. Retrieved September 17, 2017, from https://www.kaggle.com/hugomathien/soccer