**Football Match Prediction**

**Final report for the Data Product Architecture course**

**ITAM Data Science Masters Degree, 2022**

**Team : ubiquitous-goggles**

Fernando Miñaur Olivares

Miguel Calvo Valente

Marco Ramos

**Table of Contents**

[**Problem Definition 3**](#_Toc103517842)

[**System design 3**](#_Toc103517843)

[**Data source, ingestion and feature engineering 3**](#_Toc103517844)

[**Machine Development 3**](#_Toc103517845)

[**Model Evaluation 3**](#_Toc103517846)

[**Model Serving 4**](#_Toc103517847)

[**Reflection 4**](#_Toc103517848)

[**Broader Impacts 4**](#_Toc103517849)

[**References 4**](#_Toc103517850)

## Problem Definition

250 words

● This section explains the problem the team is solving, discusses related work, and proposes and justifies their solution.

## System design

500 words

**●** This section explains and justifies central design decisions, including that of which technologies the team chose to use to support their system.

●  This section details the key components of the data product architecture

●  A diagram of the architecture should be included to illustrate the interplay between system components.

## Data Source, Ingestion and Feature Engineering

We are using the Sportmonks Football API (<https://www.sportmonks.com/football-api/>) to acquire the data. It contains historical and future data regarding football matches, teams, leagues, etc.

We retrieve data from the *Head 2 Head* endpoint. Given the ids of two teams, this endpoint provides us with the historical match information between them. Each record is a football match, and it contains information such as the venue, the position of each team in the table, the weather, amongst others.

Our response variable is whether the local team won or not (binary classification problem). We calculate it as 1 if the local team id matches the winner team id, and 0 otherwise (which implies a loss or a tie).

After performing an exploratory data analysis, alongside with model testing, we chose the following set of variables as our features:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Kind** | **Description** |
| league\_id | Categorical | Id of the league f.e. Copa del Rey |
| season\_id | Categorical | Id of the season f.e. 2022 |
| venue\_id | Categorical | Id of the venue f.e. Santiago Bernabéu |
| referee\_id | Categorical | Id of the referee |
| localteam\_id | Categorical | Id of the local team |
| visitorteam\_id | Categorical | Id of the visitor team |
| localteam\_position | Numerical | Current position of the local team in the table |
| visitorteam\_position | Numerical | Current position of the visitor team in the table |

We perform an ELT process, in which we store the data exactly as we get it from the API into a table in our DB called *source*. Then, we connect to it, retrieve only the features we need, and perform minor feature engineering. The feature engineering consists in filling missing values (with a -1 label) and passing the numerical values to integers (since they are stored as strings). Finally, we store these processed and selected columns to the *model* table.

Machine Development

We chose a Random Forest model to predict the outcome of the match (justification of this model through results can be found in ***Model Evaluation***). This model is an extension to Decision Trees, in which the predictions are obtained as a series of binary decisions in the features, and an information metric, such as entropy, is minimized. Random Forests are a collection of Decision Trees to reduce variance (the majority is most often right), and they also randomize the features used to predict in each tree to reduce covariance. This results in more biased models that greatly compensate this through the reduction of variability.

As part of the training pipeline, we need to preprocess the data to use the *RandomForestClassifier* in *sklearn*. For this, we apply OHE to the categorical variables. The OHE preprocessor we built takes into consideration both non-observed values (when using the test sample or new values) and poorly represented values i.e. observations that occurred too few times.

The train and test datasets are taken from the *model* table (see the details in the ***Data Source…*** section). We use a 5-year window filter to select the data to avoid training with outdated, non-representative matches.

We train the OHE preprocessor with only the train set, since in practice the test set represents unseen data (such as future data will be), and save it in the *processor\_state.pkl* file. We transform the train data and fit the Random Forest model, which is also saved in the *model.pkl* file.

We then create a CustomModelPrediction instance, which will load both .*pkl* files and be able to preprocess new data and make predictions. The reason for using this class as a wrapper is to comply with AI Platform model specifications to predict from the endpoint.

We package our distribution using *setuptools*, and upload all the required files into Cloud Storage, under the v$***N*** folder, where ***N-1*** equals the latest version. Finally, we create the new v$***N*** version of the model in AI Platform by pointing to the Cloud Storage folder.

All this process is orchestrated via Airflow to retrain the preprocessor and the model once every month, or whenever it is needed.

## Model Evaluation

500 words

● This section describes the team’s efforts to validate and evaluate their model performance as well as its limitations. ●  The results are included and presented in a clear and informative manner.

We explored 3 models: A logistic regression, an l1 penalized logistic regression, and a Random Forest (RF). Out of these, the best model proved to be the RF:

|  |  |  |
| --- | --- | --- |
| **Model** | **Train Accuracy** | **Test Accuracy** |
| Logistic Regression | 63.58% | 61.28% |
| L1 - Logistic Regression | 62.14% | 61.14% |
| Random Forest ***v1*** | 67.20% | 69.02% |

These results apply to the initial modeling exploration with a subsample of the data i.e. ***v1***. When we get more data, there could be a different new best model over RF (f.e. L1 – Logistic Regression). We will discuss this in the ***Reflection*** section later. Nevertheless, we decided to keep the Random Forest also because several previous works tend to incline towards them.

The latest version of the Random Forest, ***v12***, which takes into consideration a dataset 10 times bigger i.e. a lot more different values for each category, yielded the following results:

|  |  |  |
| --- | --- | --- |
| **Model** | **Train Accuracy** | **Test Accuracy** |
| Random Forest ***v12*** | 65.71% | 66.2% |

These results are comparable to those obtained previously in ***v1***. On the one hand, we expected it to be better since we are having more observations, which in theory helps the model learn patterns more effectively. On the other one, we are glad that the great increase of unique observations in each categorical variable did not affect as much as could have happened.

We believe these are solid results since, if football matches were so easy to predict, the gambling market in this sport would have long disappeared. Also, some of the previous work did not seem to get considerable better results. Nevertheless, they could have been better. Our primary idea for this is using more and better features (f.e. a time weighted average of the most recent goals, both scored and received). More details of this in the ***Reflection*** section.

## Model Serving

200 words

●  This section shows how the model serves in production. what kind of predictions is it delivering? Is there any additional functioning that should need to be addd.

●  What are the manual components of it? Who should own it? What if the DAG fails?

The model endpoint is deployed in AI Platform. It is named *football\_match\_predictions*, under our project, *ubiquitous-goggles*. The current default version is ***v12***. To make a prediction, we used the python script in GCP documentation (<https://cloud.google.com/ai-platform/prediction/docs/online-predict#python>).

This function receives the following parameters: ***project*** (*ubiquitous-goggles*), ***model*** (*ubiquitous-goggles*), ***instances*** and ***version***. The instances variable is meant to receive a string of the dictionary of features to make predictions on.

The prediction is a list of lists, each of which contains two values: The probability of the local team winning and the complement of this. Notice we only need one of either of this to get the other. However, we left it like this since in the future we would like to return two values: the probability of the local team winning and the probability of the visitor team winning (thus, 1 minus these two probabilities would result in the probability of a tie).

Even though we wanted this endpoint to be available for anyone to make predictions on, we find it unviable due to the nature of the categorical variables. That is, only people with access to the Sportmonks would now the indices for the venue, league, season, teams, etc.

For this reason, we rather chose to deploy a simple application that makes predictions for the matches in the following 7 days. Through the API, we can get these matches’ information (of course, the *winning\_team\_id* is null). We make the call for

## Reflection

500 words

● This section provides a comprehensive post-mortem on the project, including - but not limited to - answering the following: - What worked? (In terms of technology, design decisions, team dynamics, etc.). - What didn’t work? What would you improve next time? - If given unlimited time and resources, what would you add to your application? - If you have plans to move forward with this application, what are they? (We’re always excited to see how students use the tools they’ve learned in this class to pursue topics they’re excited about!)

## Broader Impacts

250 words

●  This section discusses intended uses of your application and possible unintended uses and the associated harms.

●  This section reflects upon the design decisions that the team undertook to mitigate harms associated with unintended use of the system.

## References

Unlimited

<https://medium.com/@nicholasutikal/predict-football-results-with-random-forest-c3e6f6e2ee58>

<https://algobeans.com/2021/03/29/random-forest-tutorial-predicting-goals-in-football/>