Soccer Guru:
a football match prediction pipeline

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

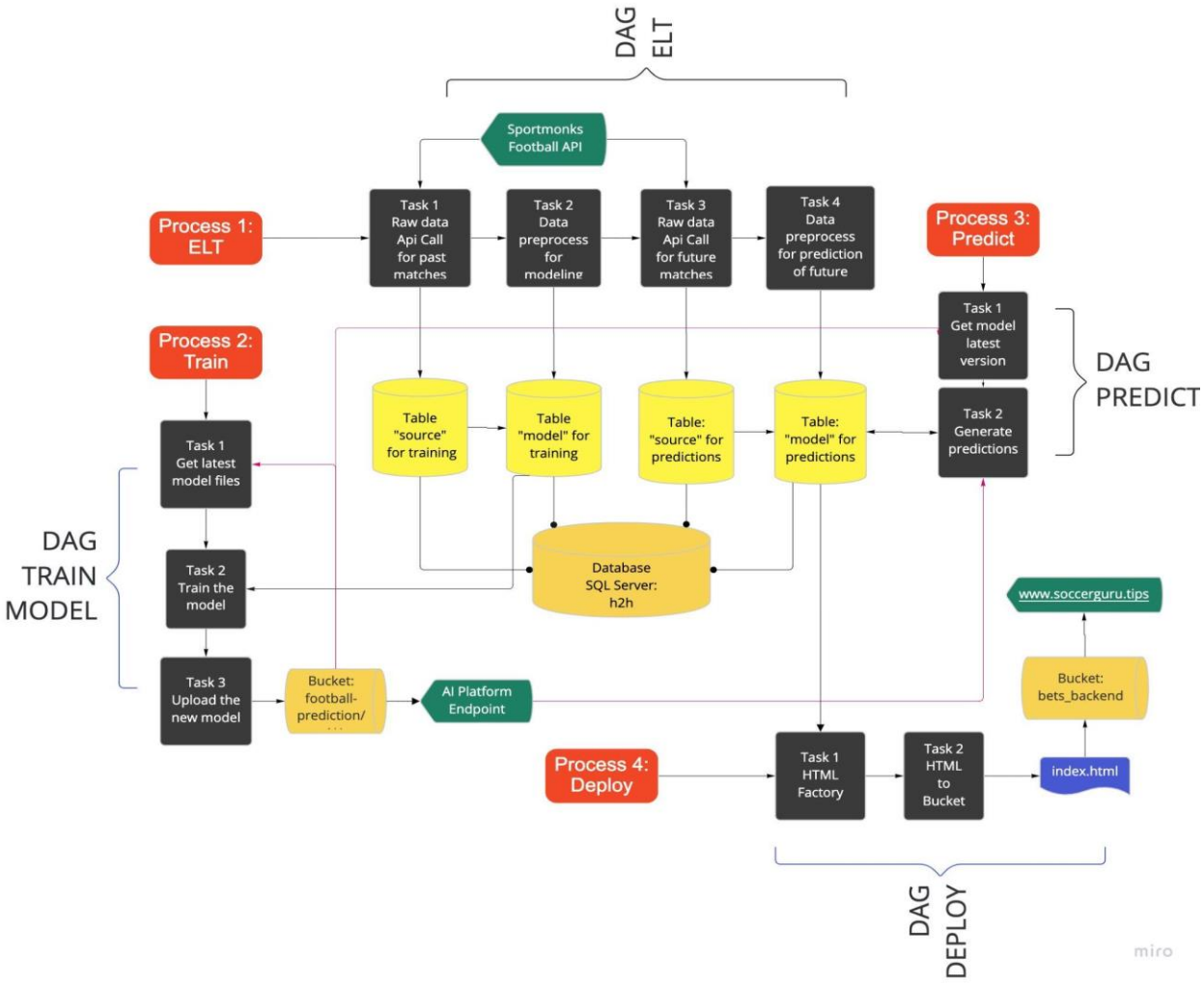
In the ecosystem of sports models and predictions, where some are more accurate, some are more expensive, and some are embedded silently in gamblings platforms, we want to provide a free, transparent, accessible, well documented, reliable and open source model-platform for anyone to use as a tool in their gambling and soccer passion. For this purpose, the problem that we want to solve throughout this project is to predict the result of soccer matches with accurate information using historical data of past confrontations.

These predictions can be constructed through Machine Learning (ML) algorithms. The justification for occupying ML methods lies in their superior ability to catch complex patterns otherwise impossible to detect by inspection. The power of ML is vastly dependent on the number and quality of observations at hand. For this, we retrieved a dataset of approximately 9000 matches for 4 of the most important European leagues through *Sportmonks Football API*, and continue to expand it as new observations become available.

We implemented an orchestrated, end-to-end data product in Google Cloud Platform to solve this problem. The reason for this is due to the daily frequency of multiple football matches occurring around the world. We find that having an automated, periodical pipeline reduces the human error in each process, as well as saves up a vast amount of time that would otherwise be dedicated to these processes.

System design

The following image shows the overall flow of our product:



Our pipeline is integrated by 4 different processes orchestrated via airflow. Each one of them runs independently in order to minimize and mitigate failure risks.

1. ELT

We perform an Extract - Load - Transform (ELT) process every 7 days on the data. Note that we previously initialized the DB with thousands of registers.

1.1. Raw data API call for past matches

We make API calls to the Sportmonks Football API to get 7 days worth of historical football matches data. We load this data as it is into the *source* table.

1.2. Data preprocess for modeling past matches

We select the columns that we need for the model, fill NAs, and store this preprocessed data into the *model* table. We chose to separate the source from the preprocessed data to have both a complete unaltered dataset in case we need to use new variables, and a dataset readily available when used for training.

1.3. Raw data API call for future matches

We make API calls to the Sportmonks Football API to get 7 days worth of future football matches data. We load this data as it is into the *source_predictions* table.

1.4. Data preprocess for prediction of future matches

We select the columns that we need for prediction, fill NAs, and store this preprocessed data into the *model_predictions* table for ready availability when it is used to make predictions.

2. Train

We retrain the model each month with the newly acquired data, and store the model as a new version in AI Platform.

2.1. Get latest model files

We get the model files from the latest version v$***N-1*** (where ***N-1*** is the latest version index). In particular, we include the *.py* scripts that contain the classes of both the *DataPreprocessor* and the *CustomClassPrediction*.

2.2. Train the model

We get data from the past 5 years from the *model* table, perform minor transformations, and split it into train and test sets. We train a *DataPreprocessor* instance in order to build a One Hot Encoder (OHE) that filters out low represented categories. This instance is stored in *processor_state.pkl*. We then train the model and store it in *model.pkl*.

2.3. Upload the new model

We create the new distribution through *setuptools*. Then, we upload all the *.pkl*, *.py* and *.tar.gz* files to a new v$***N*** folder in Cloud Storage, and create a new *AI Platform* endpoint with version v$***N***.

3. Predict

We make predictions through the latest version in AI Platform and store them in the DB on a weekly basis..

3.1. Get model latest version

We retrieve the current index ***N*** of the latest version in order to run the most recent model.

3.2. Generate predictions

We connect to the *model_predictions* table and get all of the matches 7 days into the future. We perform minor transformations and make predictions through the AI Platform endpoint, using the v$***N*** version. We store these predictions by updating the *probs* column in the *model_predictions* table.

4. Deployment

4.1. HTML Factory

We request data from the *model_predictions* table of the current week matches, add the team logos, names and context info, and create an output table in HTML that will be rendered in our webpage.

4.2. HTML to bucket

We copy the HTML in a bucket that serves as the backend of our webpage, which is hosted in GCP. Having already configured the load balancer and our site credentials, the HTML in the bucket is rendered in the webpage automatically:

https://www.soccerguru.tips/index.html

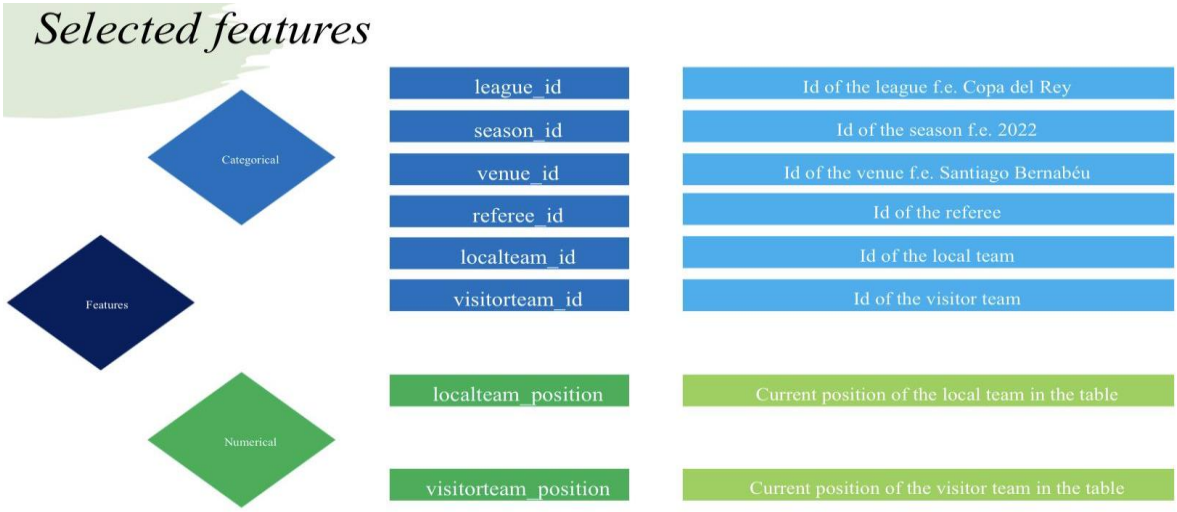
Data Source, Ingestion and Feature Engineering

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 retrieved 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 winning 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:



We perform an ELT process, in which we store the data exactly as we get it from the API into a table in our database (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 XGBoost 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 of 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.

As part of the training pipeline, we need to preprocess the data to use the *GradientBoostingClassifier* in *sklearn*. For this, we apply the method 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 XGBoost 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

Throughout this project we explored 4 models:

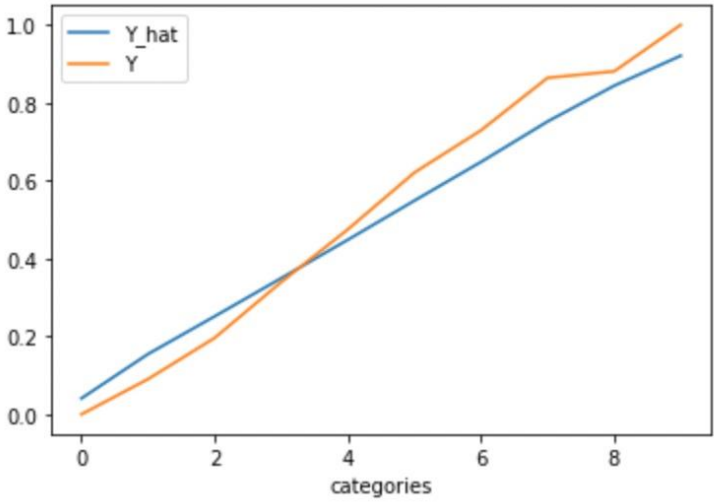
- Logistic Regression,
- L1 penalized Logistic Regression
- Random Forest (RF)
- XGBoost

Out of these, the best model proved to be the XGBoost in terms of its accuracy:

Model	Train Accuracy	Test Accuracy
Logistic Regression	63.58%	61.28%
L1 – Logistic Regression	62.14%	61.14%
Random Forest	67.20%	69.02%
XGBoost	77.34%	74.29%

We decided to keep the XGBoost in the pipeline given these results and also because several previous works tend to incline towards them.

Also we made monotonicity tests in order to ensure not the performance but the common sense and logic of our predictions. For this we used a lift table that measures for each decile (category in the plot) of the predicted probabilities (Y hat) the propensity of actual Ys that occurred. If the model makes sense we would expect that the propensity increases as the decile increases: it is expected that the matches with higher local winning probability will tend to accumulate more actual local wins, thus, our predictions are monotonic.



We can also observe through the lift table that we are overestimating the probability when the team is actually less likely to win, and we are underestimating in the opposite scenario.

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 works did not seem to get considerably better results. Nevertheless, they could have been better with a more thorough feature engineering.

Our primary idea for upgrading accuracy is using more and better features. Our 2 current candidates are time weighted averages of the most recent goals, both scored and received. We worked locally in these 2 features and tested them with a Random Forest. It yielded the following results:

Model	Train Accuracy	Test Accuracy
Random Forest (trial)	84.83%	85.65%

These results are definitely better than the preceding ones. We are definitely going to implement these in the pipeline in the near future. The reason for which they have not been added yet is that certain files must be carefully modified, their acquisition is not straightforward, and it would require some time to test the correct functioning of the full pipeline.

Another way in which we pretend to evaluate our model is to bet a fictional 1$ for every match that has a high inclination towards one team winning (given a threshold) over the course of a period. Then, the net average earning at the end of the period would give us a metric to evaluate our models. We have yet to implement this because we have to cross our results with the odds of multiple betting houses in real time, which is considered in further upgrades to the pipeline.

Model Serving

The model endpoint is deployed in the AI Platform. It is named *football_match_predictions*, under our project, *ubiquitous-goggles*.

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 instance 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.

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.

For this reason, we rather chose to deploy a simple application that makes predictions for the matches in the following 7 days. We chose to use a webpage to show our predictions.

On a weekly basis, we request this week's matches with their prediction from our database and create a table with context information such as the match date, team logos, etc, and create an html file with this table that is copied to a GCP bucket. This bucket serves as the backend of a webpage hosted in GCP, so whatever files we put there are shown in our webpage https://www.soccerguru.tips/index.html. By doing this we are not forcing our audience to know how to use APIs but rather we are providing a really intuitive platform to track the predictions.

Reflection

Integrating our team with different profiles definitely was useful for us. Also luckily, all our team members were highly motivated and highly creative and experimental when getting to know the GCP platform.

Also we think that we could have benefited more if we as a team were more organized, we underestimated the complexity of some parts and this caused us to have a technological debt at the end of the semester, which we were able to reduce in the last weeks but at the cost of intensive work and prioritizing parts of the pipeline. For example we couldn't implement the best version of our model due to lack of time and fear of breaking an already functioning component of our pipeline.

Also, we couldn't dedicate more time to an efficient ELT (which we estimate will take us a couple of months and some extra courses) because a brute force massive API call is very slow (due to the combinatorial nature of possible matches), thus we had to load leagues one by one and as a consequence is for example that we are not contemplating the teams promoted and relegated across divisions. The matches of this kind of teams we will have to upload manually and quirurgically season by season.

If we had the time we would have invested more time in the EDA and Modeling part because at the end of the semester we were able to create impressive models (with accuracies above 85%) but we lack the time to implement them correctly. Also, we have made constant discoveries in the data, trends and patterns, so we think that we can integrate them to our model in a constant and persistent way in the future. Furthermore, we believe we haven't taken full advantage of all the information presented to us in the API. As we developed the final model, we started to realize the request complexity needed to extract additional and more valuable information found within the variables themselves. For example, the final score variable that contains the score information throughout the whole match through several variables including penalties, half time score and final score.

Another thing that worked was the granularity of the pipelines, because we used a different dag for each isolated process, thus we were able to mitigate risk of failure of a complex and long chain of events.

Finally, in order to give continuity to the project, we would like to focus more on the front-end and user experience of the model. We were able to create a platform that "does the job" at the cost of aesthetics.

Broader Impacts

Among the intended uses is that ordinary people could access a free, easy to use and ever-functioning model so they can have a more informed bet. However, we should consult a lawyer or expert in this industry about using a warning message to protect us from any lawsuit regarding the faith and bad use of our predictions.

Regarding our past point, at the beginning we wanted to suggest bets or outcomes, however at the end we concluded that it is better to give the model probability so that each individual assumes responsibility for their bet choice and risks.

Also, we need to think about the case where our web page becomes popular and maybe the costs of our hosting become unsustainable and the cases where we could get hacked or threatened by other platforms, individuals, or companies. For the moment, we have not invested more resources in cybersecurity than the GCP default practices.

Finally, a bad use could be that teams or individuals use the probabilities to rigged beneficial matches (for example, those ones against the odds), however this is not a particular problem of our platform, but rather a broader and bigger whole industry issue.

References

-Calvo, M., Miñaur, F., Ramos, M., & Lezama, J. (3 de February de 2022). *Github*. Obtenido de Ubiquitous-Goggles: https://github.com/JulioLezamaAmastalli/ubiquitous-goggles/

-Densmore, J. (10 February 2021). *Data Pipelines Pocket Reference: Moving and Processing Data for Analytics*

-SportsMonk. From SportsMonk: https://docs.sportmonks.com/football/

-Sze Yeung. *Football Prediction by XGBoost.* Kaggle https://www.kaggle.com/code/szeyeung/football-prediction-by-xgboost/notebook?scriptVersionId=94437712

-Utikal, N. (26 December 2019). *Predicting Football Results with Random Forest*. Obtenido de Medium: https://medium.com/@nicholasutikal/predict-football-results-with-random-forest-c3e6f6e2ee58

Project Development Leaders

Each member of the team was assigned with a specific part of the pipeline. Nevertheless all the members contribute to the development of each phase when the leader got stuck or ran out of ideas to solve each step.

EDA and Model Leader

- Fernando Miñaur

AI Platform and Predictions Leader

- Miguel Calvo

ELT and Deployment Leader

- Marco Ramos