QRT Football Challenge Report

Timothé Boulet **Théo Saulus**
Master MVA
{timothe.boulet, theo.saulus}@student-cs.fr

Abstract

In this report we present our attempts and results on the QRT Data Challenge on football games winner prediction (https://challengedata.ens.fr/challenges/143). Our main contributions are in team features engineering, with winrate and Elo, player features engineering with tensorial data analysis, and findings about the distribution shift of the test dataset. Our code will be made available publicly once the competition is over, at https://github.com/tboulet/QRT-Data-Challenge-Football.

Contents

1 Introduction **3**
1.1 Challenge presentation . . . 3
1.2 Structure of this report . . . 3

2 Pipeline overview **4**

3 Preprocessing **5**
3.1 Features correlations and drop . . . 5
3.2 Missing data handling . . . 5

4 Team features engineering **5**
4.1 Team winrates and team-couple winrates . . . 5
4.2 Team elo . . . 6
4.3 Team name prediction on test set . . . 6

5 Player features engineering **7**
5.1 Naive usage of player features . . . 7
5.2 Statistical analysis and aggregation . . . 7
5.2.1 Introduction to RGCCA and TGCCA . . . 7
5.2.2 Aggregate by position then feature selection with TGCCA . . . 8
5.2.3 TGCCA then aggregate by position . . . 8

6 Model **9**

Preprint. Under review.

6.1 XGBoost presentation 9
6.2 Model hyperparameters tuning and feature selection 9
6.3 Our best model 9

7 Findings **10**
7.1 Train/test imbalance 10
7.2 Further features filtration 11

8 Further directions **11**

1 Introduction

1.1 Challenge presentation

In the QRT data challenge of this year, we had to develop a predictive model to forecast the results of football games using historical data on teams and players. The data set spans numerous leagues and divisions around the world, so that the proposed model is adaptable and applicable to any football league, regardless of competition level or geographic location. Input data is provided with match data split between home and away teams and very detailed across both team and player levels. Teams data have 25 metrics about attacks, possession, goals, etc., while player data includes 52 metrics, all of them being normalized. Note that players' and teams' names are not available at test time.

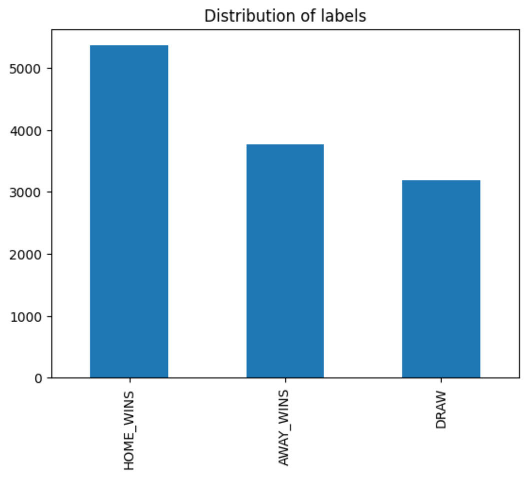
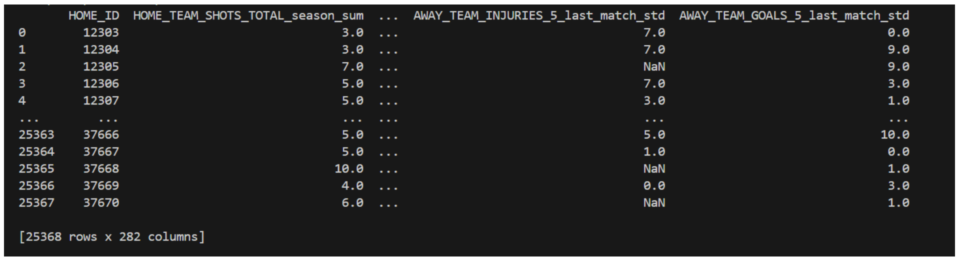


Figure 1: Distribution of the three labels in the train dataset

Each data point represents a match and is composed of the home team and away team features, as well as the players of each team. The number of players in a team and their position is not constant and must be aggregated, which was one of the main specificities of this challenge. The goal is to predict the outcome of the match, which can be a win for the home team, a win for the away team, or a draw.



```
       HOME_ID  HOME_TEAM_SHOTS_TOTAL_season_sum  ...  AWAY_TEAM_INJURIES_5_last_match_std  AWAY_TEAM_GOALS_5_last_match_std
0        12303                               3.0  ...                                  7.0                               0.0
1        12304                               3.0  ...                                  7.0                               9.0
2        12305                               7.0  ...                                  NaN                               9.0
3        12306                               5.0  ...                                  7.0                               3.0
4        12307                               5.0  ...                                  3.0                               1.0
...        ...                               ...  ...                                  ...                               ...
25363    37666                               5.0  ...                                  5.0                              10.0
25364    37667                               5.0  ...                                  1.0                               0.0
25365    37668                              10.0  ...                                  NaN                               1.0
25366    37669                               4.0  ...                                  0.0                               3.0
25367    37670                               6.0  ...                                  NaN                               1.0

[25368 rows x 282 columns]
```

Figure 2: Some data points of the test dataset

The team and player characteristics were given in the form of different aggregated metrics in the last season. For example, we had access to the mean, sum, and standard deviation of the number of goals scored by each team in the last season and in the last 5 matches. The features were normalized beforehand by the challenge organizer so that they take integer values between 0 and 10 (for the team features) and 100 (for the player features). We plot here the distribution of these features for one metric (the accurate passes).

1.2 Structure of this report

Thoughout this challenge, we have attempted many techniques to improve our score (sections 3.1,4, 5, 3.2 and 6). However we came late to the realization that the test set has a significantly different

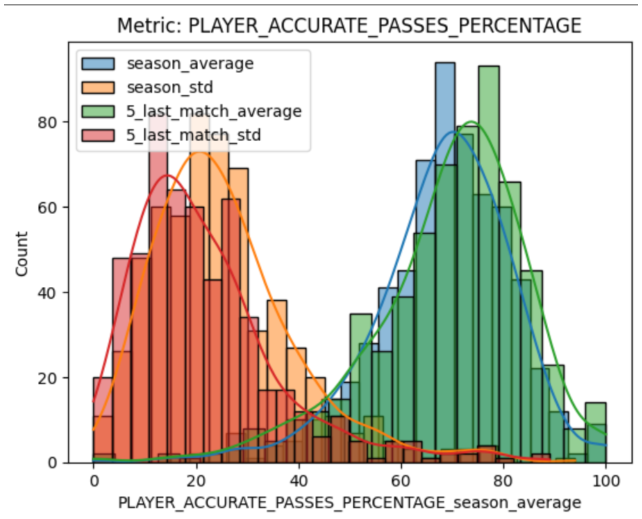


Figure 3: Distributions of 4 features: the aggregated statistics of the player accurate passes metric

distribution from the train set, and some features should therefore be discarded since they may not help in the test predictions (section 7). Therefore, the methods developed had little influence on the predictions, but they are still relevant to discuss in the scope of the class.

2 Pipeline overview

We divided the original provided train data set into a train data set and a validation data set with K-fold cross-validation, for which we used $K \in \{10, 20\}$. Our metric to assess how well our model performed without submission on the challenge website was the mean or median of the accuracy of the K models.

We first apply a feature engineering step to both train and validation datasets, as well as to the test dataset, in the case where we wanted to predict and submit our results. The feature engineering phase consist of the following steps:

1. Loading the data
2. Dropping the team features and player features that were redundant or irrelevant
3. Perform feature crafting for the team features (home team, away team, and couple of teams features)
4. Perform feature crafting for the player features (feature selection and summarizing features)
5. Aggregating the player features to obtain a fixed number of player features, as this is the way we will feed the data to our learning algorithm
6. Other standard data preprocessing steps such as feature imputation, data shuffling, etc.

Then we trained a model of our choice on the train dataset, and we evaluated its performance on the validation dataset (if we used cross-validation) or we predicted the labels of the test data set (if we wanted to submit our results).

In practice, we implemented our pipeline in `Python` and relied on the packages `Pandas`, `Numpy`, `Scikit-Learn` and `Hydra` for the most part. Some analysis were performed in `R` using the `RGCCA` package.

3 Preprocessing

3.1 Features correlations and drop

We noticed some high correlation between some aggregated function of the same metric. The sum and the average in particular were almost always highly correlated (see Figure 4 left and right), and the standard deviation were sometimes also correlated with the sum and the average (see Figure 4 right). We decided to drop the features whose correlation with an other feature was exceeding a certain threshold (typically 0.8) to avoid overfitting and to reduce the dimensionality of the data.

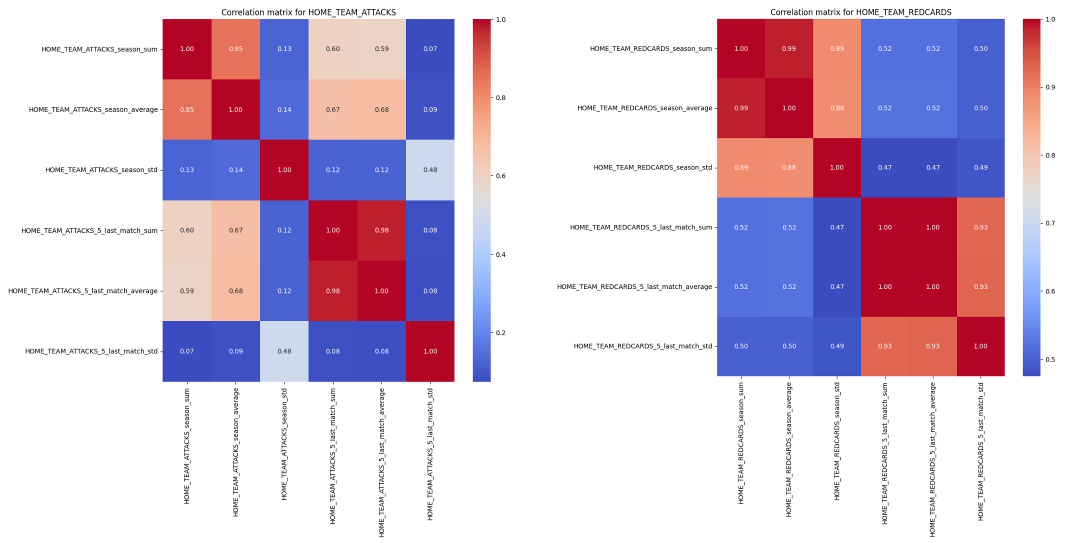


Figure 4: Correlation matrix for the features of the same metric (home team attack on the left, home team red card on the right). One can see that for red cards, the sums and averages provide almost identical results.

We also dropped certain features that were absent too often in the dataset (e.g., PLAYER_LONG_BALLS, PLAYER_SHOTS_OFF_TARGET).

3.2 Missing data handling

For a few data points, some feature values were missing. We imputed those missing values using the mean or the median of the feature over the entire dataset.

However, because their absence can potentially mean something, we also added a binary feature indicating whether a feature was missing or not, and we did the same if the entire metric (each aggregated function) was missing.

Those preprocessing steps, coupled with a simple aggregation of players features, allowed us to obtain a test prediction accuracy of 0.487 on the public test (ranked 17^{th}), and 0.4900 on the private test (ranked 2^{nd}), which have remained our best score (more details about our best model in Section 6). *None of the following features engineering allowed us to improve our score*. We explore some reasons why in section 7.

4 Team features engineering

4.1 Team winrates and team-couple winrates

Definitions. Beyond team metrics, we wanted to compute and include other key factors. One of them is the team's win rate and their head-to-head win rate when competing. Additionally, we incorporated the win differential, for both match-specific and overall performance.

Let T_A, T_B denote the identifier for the two teams, $N_{total}(T_A)$ denote the total number of games of T_A, and $N_{total}(T_A, T_B)$ denote the total number of games between T_A and T_B. Please note that due to the limited amount of data available, we do not differentiate whether the teams played home or away.

The winrate of T_A is defined as:

$$W(T_A) = \frac{N_{\text{Win}}(T_A) + 0.5 \times N_{\text{Draw}}(T_A)}{N_{total}(T_A)} \tag{1}$$

The team-couple win rate of T_A considered against T_B is defined as:

$$W(T_A \mid T_B) = \frac{N_{\text{Win}}(T_A \mid T_B) + 0.5 \times N_{\text{Draw}}(T_A \mid T_B)}{N_{total}(T_A, T_B)} \tag{2}$$

where $N_{\text{Win}}(T_A \mid T_B)$ can be read as the number of games won by T_A when the opponent was T_B.

Data leakage and rebalancing. If no care is taken, winrates could be computed from the initial training dataset. This creates an issue of data leakage, in the sense that the winrate would contain information about games that occur in the validation dataset. This will cause the model to overfit during the training phase, and provide performances that are way too optimistic. For example, if team A has won 2 times and lost 1 time against team B, and only 2 of those games are included in the training fold, the information about the 3rd game result is included in the winrate. But this does not help learning about new future games, like those in the test dataset.

To correct this, we rebalanced these winrates so that the rebalanced winrate corresponds to the winrate computed on the initial training dataset without the current data point. We performed this with the following formula:

$$\text{Rebalanced winrate} = \frac{\text{Winrate} \times \text{Number of games} - \mathbb{1}_{\text{Win}} - 0.5 \times \mathbb{1}_{\text{Draw}}}{\text{Number of games} - 1} \tag{3}$$

where $\mathbb{1}_{\text{Win}}$ (respectively $\mathbb{1}_{\text{Draw}}$) indicates if the current data point is a win (resp. a draw) for the considered team.

4.2 Team elo

We also implemented a notion of "elo" score for each team, which represents the overall strength of the team and is computed based on the matches of the initial training dataset. We then added the elo of each team as well as the difference of elo between the two teams.

For each match:

- Calculate the expected scores for each team using their Elo ratings (initially 1500)
- Compare the expected scores to the actual scores.
- Update each team's Elo rating based on the K-factor (e.g. 32) and the disparity between expected and actual scores.

$$\text{Score}(T_A) = 1_{\text{A won}} + 0.5 \times 1_{\text{Draw}} \tag{4}$$

$$\text{Elo}(T_A) = \text{Elo}(T_A) + K \times \left(\text{Score}(T_A) - \frac{1}{1 + 10^{\frac{\text{Elo}(T_A) - \text{Elo}(T_A)}{400}}} \right) \tag{5}$$

We did not perform rebalancing for this feature, but we acknoledged that the validation accuracy was systematically overestimistic when using this. Thus, our only actual measure was to submit against the test set, which did not prove to be very conclusive.

4.3 Team name prediction on test set

Because the team names were missing in the test dataset, we had to find a way to impute those identifiers in the test dataset. For this, we trained beforehand an other model to predict the name of each team among the 350 teams that appear in the train dataset. This model was using the same features as the main model (see Section 6). We reached 95% accuracy on the team-prediction validation dataset.

An idea we had, but did not have time to implement, is to constrain the team-couple prediction to a couple of two teams that are in the same league, since other kind of combinations are impossible.

This would involve:

1. using a multi-logistic regression model that gives numerical (not categorical) scores, e.g. probabilities $P(t_A = T_A)$, of a team t_A to be a given team T_A, based on available informations.
2. predicting the team names (T_A, T_B) subject to the constraint of belonging to the same league:

$$(T_A, T_B) = \underset{(t_A, t_B) \in \text{Same league}}{\arg\max} \quad P(t_a = T_A)P(t_b = T_B). \tag{6}$$

5 Player features engineering

5.1 Naive usage of player features

The simplest and most intuitive way to process the player features is by performing the preprocessing described in Section 3 on the player features, and aggregate the remaining features. This aggregation is performed with mean/median/etc. operations for all the players in the team, and concatenated with the team features.

5.2 Statistical analysis and aggregation

In order to obtain a relevant set of features to compute, we looked for a method that is able to handle naturally the players' data structure. More precisely, we need to handle a random matrix of n_p players $\times$ f_p features for each game played, with $n_p \in \{1, \dots, 25\}$ depending on the game, for m matches. Our aim is to build a weight vector that indicates how important each feature is, so that a new summarizing metric is computed.

The naive method returns f_p features, but it suffers from two drawbacks:

1. It generates a very large number of features, which may need to be filtered out
2. It blurs the features, since very different players are aggregated.

To address those issues we propose two complementary analysis:

1. Aggregate players that have the same positions, then perform a Tensor Generalized Canonical Correlation Analysis (TGCCA) [Girka et al., 2023] to obtain a limited number of relevant features for each position
2. With this TGCCA obtained for each player position, use the features weights to compute a linear combination of features that helps explain the result.

5.2.1 Introduction to RGCCA and TGCCA

In classical Regularized Generalized Canonical Correlation Analysis (RGCCA) [Tenenhaus and Tenenhaus, 2011], we consider a set of L random vectors $\{X_1, X_2, \dots, X_L\}$, where each X_l is an element of $\mathbb{R}^{p_l}$ and is assumed to have a mean of zero. The covariance matrix $\Sigma_{lk} = \mathbb{E}[X_l X_k^T]$ represents the cross-covariance between the l-th and k-th vectors. For a given random vector X_l, we seek to determine the canonical vector $y_l = w_l^\top X_l \in \mathbb{R}^{p_l}$ which is essentially a linear combination of the elements in X_l that captures the most informative shared characteristics among the blocks of data. RGCCA solves the following optimization problem:

$$\begin{aligned} &\underset{w_1, \dots, w_L}{\text{maximize}} \quad \sum_{l,k=1}^{L} c_{lk} g\left(w_l^T \Sigma_{lk} w_k\right) \\ &\text{subject to} \quad w_l^T M_l w_l = 1, \quad \text{for } l \in \{1, \dots, L\}, \end{aligned} \tag{7}$$

where the function g is a continuously differentiable convex function, for optimization reasons, and each block regularization matrix $M_l \in \mathbb{R}^{p_l \times p_l}$ is symmetric and positive-definite, contributing to the stability of the solution. The design matrix $C = [c_{lk}]$ is a symmetric $L \times L$ matrix containing non-negative elements, which indicate the connections between the data blocks (i.e., do they relate). See Figure 5 for details about our case.

Note that this formulation encompasses classical data analysis techniques such that Principal Component Analysis ($L = 1$ and $M = 1$), or Canonical Correlation Analysis ($L = 2$, $M_1 = \frac{1}{2} X_1^\top X_1$, and $M_2 = \frac{1}{2} X_2^\top X_2$).

TGCCA [Girka et al., 2023] is the extension of RGCCA to tensors. Typically for this project, players data can be considered as order 3 tensors of dimension $m \times n_p \times f_p$, for which we want to obtain relevant linear combination of features. A tensor $\mathcal{X} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ of order d is said to be of rank one if there exist d vectors $(w_1, \ldots, w_d) \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ of unit norm and $\lambda \in \mathbb{R}$ such that $\mathcal{X} = \lambda w_1 \circ \ldots \circ w_d$. During our analysis, we will assume that players blocks are (approximately) rank one, since we will aim to compute such a decomposition. The TGCCA optimization problem, for a mixture of matrices and order 3 tensors is essentially the same as (7), with additionnal constraints on the weight vectors w_i related to tensor blocks, and adaptations for the covariance matrix.

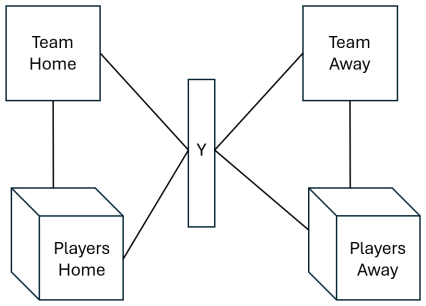


Figure 5: Connection model during TGCCA analysis. Squares indicate a matrix, and cubes indicate an order 3 tensor. Y is the game result.

5.2.2 Aggregate by position then feature selection with TGCCA

Our first proposition is to separate the training data as shown in Figure 5, and use the obtained weight vectors on the player features dimension w, to only keep the most informative features.

This analysis is repeated 5 times: once with all players, and four other times with players filtered by every position (attack, defense, midfielder, goalkeeper). This results in 5 weight vectors for the home side, and 5 others for the away side, denoted for instance as $w_{\text{attack, home}}$.

Then, we use those weights as a way to perform feature selection:

1. Compute the weight vectors
2. For each player, only keep the N features with larger weight, for the corresponding position (i.e., $w_{\text{position, home/away}}$). Typically, we used $N \in \{1, \ldots, 16\}$. In case the player position is not available, use the weights obtained with all players mixed.
3. Concatenate those $N \times 5$ selected features with team features.

This allows to reduce the amount of features kept, while increasing their relevance since we do not mix all players together anymore. We attempted to merge all players in one block, but this did neither improve our results nor the variance explained by the players blocks.

5.2.3 TGCCA then aggregate by position

Our second proposition is to use the weights provided by TGCCA to obtain summarizing features for every position, and a global one using all players mixed:

1. Compute the weight vectors

2. For each player, compute the weighted linear combination of its features, using the corresponding $w_{\text{position, home/away}}$, resulting into a summarizing feature
3. Aggregate the summarizing feature by position, for home and away teams separately
4. Concatenate those 5 features with the team features.

6 Model

6.1 XGBoost presentation

The model we used is Extreme Gradient Boosting, which is a popular and efficient gradient boosting algorithm. We used the `XGBClassifier` class from the `xgboost` package.

XGBoost is a powerful implementation of the gradient boosting algorithm, which is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

The package already implement the two following ensemble methods, so that we did not have to implement them ourselves:

- **Bagging**: It is a technique that aims to reduce the variance of an estimate. It is a special case of the model averaging approach. Bagging is short for bootstrap aggregating. It works by taking many bootstrapped samples from the training data and training a separate model for each sample. The final prediction is then made by averaging the predictions of all models.
- **Boosting**: It is a technique that aims to reduce the bias of an estimate. Boosting works by training a sequence of models, each of which learns to fix the prediction errors of the previous model in the sequence. The final prediction is then made by averaging the predictions of all models.

6.2 Model hyperparameters tuning and feature selection

Because we had a lot of model hyperparameters (such as the number of trees, or the maximal depth) which are not obvious to tune, and a lot of potential features (whose choice to use or not can be considered as hyperparameters), we used an automatic hyperparameters (and feature selection) sweep allowed by the `WandB` package.

WandB is a platform that allows to log and visualize the results of machine learning experiments, and also to perform hyperparameters tuning. The sweep performs a Bayesian (or random) search of the best hyperparameters in terms of the average validation accuracy by trying different combinations of features and logging the results on the `WandB` platform [Davies, 2023]. The result of our sweep can be found at the following link: `https://wandb.ai/timotheboulet/QRT-Data-Challenge-Football/reports/-24-03-17-00-47-36---Vmlldzo3MTgxODYy`.

6.3 Our best model

Our best method is an XGBoost model with all parameters to default, except for a learning rate at 0.1 and a maximum depth of 3, with the following preprocessing:

- Removal of features as described in 3
- Imputation of unavailable data with the median method
- Adding indicators for missing metics and missing features
- Naive player feature aggregation

This submission was quite early, and we believe that the relatively small depth of 3 has been a strength to limit overfitting.

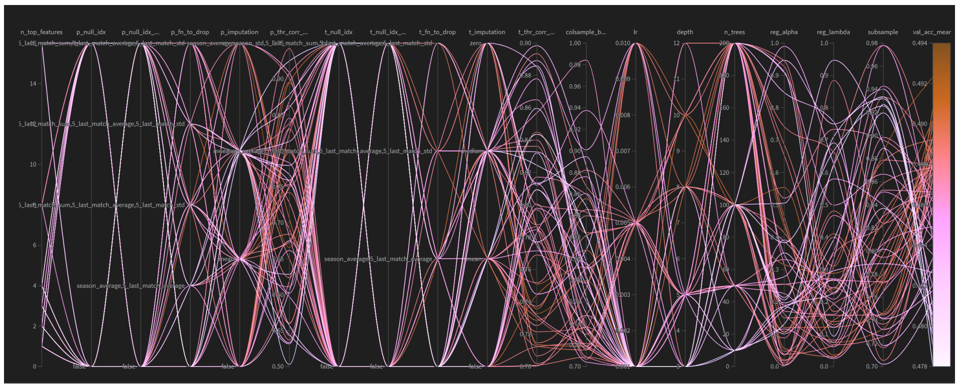


Figure 6: The hyperparameter and feature selection sweep. Colors indicate the performance in terms of average validation accuracy.

7 Findings

7.1 Train/test imbalance

Throughout our work on feature engineering and model selection, we performed visualizations and statistical analysis to understand the data and the potential relationships between the features. We present here some of our findings.

First, as mentioned in Section 3.1, we noticed that some aggregated functions of the same metric were highly correlated. However, we also noticed that the distribution of the features was not always the same for the three labels (see Figure 1).

Because our test score was much lower than our validation score, we suspected a data distribution shift between the training and the test dataset. Here we plot the train and test distributions of some features that appeared visually different (see Figure 7).

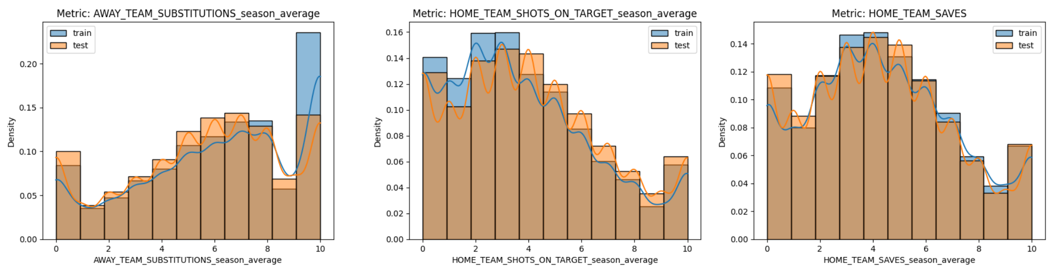


Figure 7: Comparison of the train and test distributions of some features

We can see on figure 7 that some features distribution are visually different (middle and in particular left) but not systematically (right). This indicates that there may be a distribution shift for at least some features, and that we have to be careful when using them.

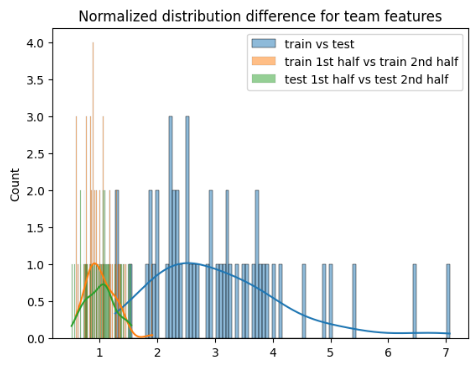
We quantified this shift by computing the distances $L(p, q)$ (where L is a distance between distributions, such as the L1 distance or the KL divergence) between, for each feature, the empirical distributions of the train data set p and of the test data set q. To ensure this was not due to the variance of the empirical distribution estimator, we normalized the distance by the (averaged over many runs) distance between p (or q) and an empirical distribution p' (or q') one can obtain by sampling from p (or q) several times (the same number of times as the size of the corresponding dataset).

$$R(p, q) := \frac{L(p, q)}{0.5 \times \mathbb{E}_{p' \sim p} L(p, p') + 0.5 \times \mathbb{E}_{q' \sim q} L(q, q')} \tag{8}$$

We obtain a ratio that represents the normalized distance between the train and test distribution of a feature. The value of this ratio helps us understand the relation between the train and test distributions:

- If the ratio is close to 1, the train and test distributions are similar
- If the ratio is significantly higher than 1, the train and test distributions are different
- If the ratio is lower than 1, it means that the train and test distribution are such that some data points in the test dataset are vrey close to some in the train dataset.

We then plotted the distribution of the normalized distribution L1-differences for all the features (see Figure 8). We also included the normalized distribution differences between, for each feature, half of the data points from the train data set, and the other half, which was expected to be around 1 (since the two halves come from the same distribution), and we did the same with the test dataset.



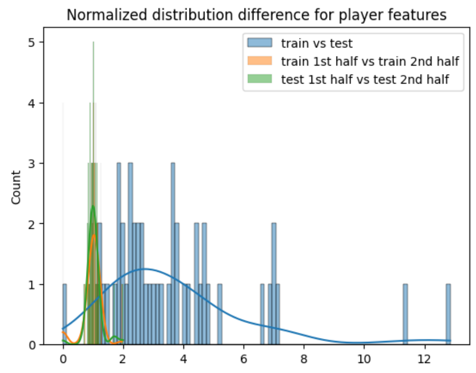


Figure 8: The distribution over the features of the Normalized Distribution Difference $R(p, q)$ for the team features (left) and the player features (right)

We observe the following:

- As expected, the ratio is very concentrated around 1 (or below 1) when considering two halves of the *train* dataset
- The same observation holds when considering two halves of the *test* dataset
- However, the ratio is significantly higher than 1 when comparing the train and test datasets, which indicates that their distributions are different.

This finding has clear consequences about the work we have presented:

- It is (more than ever) not sure that all teams in the test dataset are present in the train dataset, which is problematic for our winrate and elo team features.
- One reason for a distribution shift could be that the train dataset is older than the test dataset, and some teams/players performances may have significantly changed.
- If the distribution of player features is different in the test set, the statistical analysis we performed might not give us interesting features.

7.2 Further features filtration

We wanted to try to improve our perfomances on the test set by filtering out the features that vary too much between train and test. Due to time constraints, we could only make 2 attempts to remove features which R value is beyond 2, and beyond 3. However, we did not perform better than our best model on the public test set, and we did not keep this.

8 Further directions

Had we more time, we would have liked to try some more potential improvements:

- Improve our team names predictions by limiting possibilities in two team names that belong to the same league
- Predict the missing player positions to benefit better from the statistical analysis for all players
- Try other models such that CatBoost and LightGBM which are known to perform well on tabular data
- Test more filtering thresholds to only learn using the most stable features.

Acknowledgement

We thank Fabien Girka for his help regarding the installation and usage of the TGCCA in `R`.

References

Dave Davies. What is bayesian hyperparameter optimization? `https://wandb.ai/wandb_fc/articles/reports/What-Is-Bayesian-Hyperparameter-Optimization-With-Tutorial---Vmlldzo1NDQyNzcw#what-is-baysian-hyperparameter-optimization?`, 2023.

Fabien Girka, Arnaud Gloaguen, Laurent Le Brusquet, Violetta Zujovic, and Arthur Tenenhaus. Tensor generalized canonical correlation analysis. *arXiv preprint arXiv:2302.05277*, 2023.

Arthur Tenenhaus and Michel Tenenhaus. Regularized generalized canonical correlation analysis. *Psychometrika*, 76:257–284, 2011.