Predicting Soccer Matches Results

Advanced Data Science Capstone

Project presentation

Author: Igor Palmieri

Index

- Use case and data set
- Data exploration and assessment
- Feature engineering
- Modeling
- Performance evaluation

Index

- Use case and data set
- Data exploration and assessment
- Feature engineering
- Modeling
- Performance evaluation

Use Case: predicting soccer maches results

- Soccer is a sport that generates more than **US$50 billion/year** globally[1]
- There are **over 200 countries** with active soccer leagues for clubs[2]
- In a rough estimate, **more than 50.000 professional soccer matches** are played every year in the world[3]
- Revenue in the **sports betting market** is projected to reach **US$78 billion** in 2025[4]

Context

High **data availability** and growing **demand for forecasts**

Main objective

Predict match results

1) Source: https://www.statista.com/outlook/amo/sports/soccer/worldwide
2) Source: https://sportrankers.com/soccer/how-many-football-leagues/
3) Considering at least 20 clubs per league, plus international club competitions and national teams.
4) For all sports. Source: https://www.statista.com/outlook/amo/gambling/sports-betting/worldwide

Dataset source: Brazilian national championship

Brazilian Men's National Soccer League from 2003 to 2023

- 8.405 matches total, 21 seasons
- 20-24 teams play each season (1st division only)*
- Every team plays against one another in two rounds (home and away)
- Dataset summary:

Table file	Description	Table size (cols x rows)
campeonato-brasileiro-cartoes.csv	Details of every card (red, yellow) issued	8 x 18.857
campeonato-brasileiro-estatisticas-full.csv	Detailed match metrics for each team	13 x 16.810
campeonato-brasileiro-full.csv	Metrics and results of each match (main dataset)	16 x 8.405
campeonato-brasileiro-gols.csv	Data for every goal scored	6 x 8.932

- Available for download at:

Github: https://github.com/adaoduque/Brasileirao_Dataset

Kaggle: https://www.kaggle.com/datasets/adaoduque/campeonato-brasileiro-de-futebol

* Since 2006, the number of teams is fixed at 20

Dataset source: Brazilian national championship

Brazilian Men’s National Soccer League from 2003 to 2023

- 8.405 matches total, 21 se...
- 20-24 teams play each sea...
- Every team plays against o...
- Dataset summary:

Main dataset features

- ID
- Match round
- Date
- Hour
- Home team (HT)
- Away team (AT)
- Home formation
- Away formation
- Home coach
- Away coach
- Winner team*
- Stadium
- Home goals
- Away goals
- Home state
- Away state

Table file		Table size (cols x rows)
campeonato-brasileiro-cartoes.csv		8 x 18.857
campeonato-brasileiro-estatisticas-full.csv	Detailed match metrics for each team	13 x 16.810
campeonato-brasileiro-full.csv	Metrics and results of each match (main dataset)	16 x 8.405
campeonato-brasileiro-gols.csv	Data for every goal scored	6 x 8.932

- Available for download at:

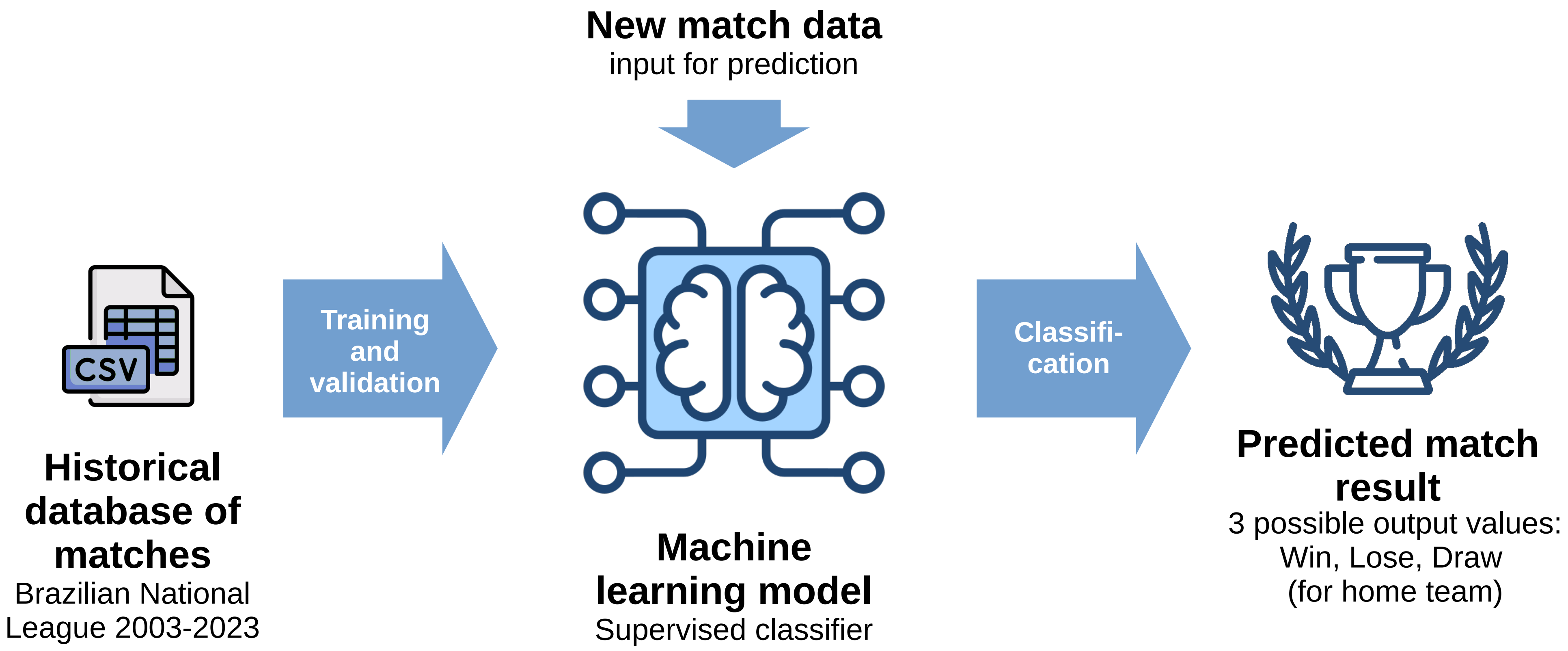
Github: https://github.com/adaoduque/Brasileirao_Dataset

Kaggle: https://www.kaggle.com/datasets/adaoduque/campeonato-brasileiro-de-futebol

* original target column

* Since 2006, the number of teams is fixed at 20

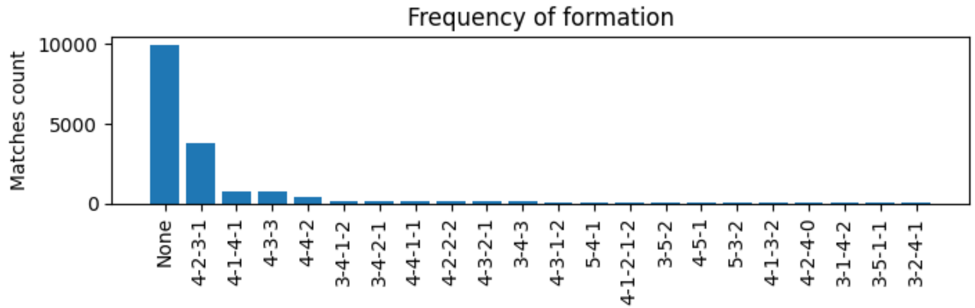
Proposed solution: a machine learning classifier

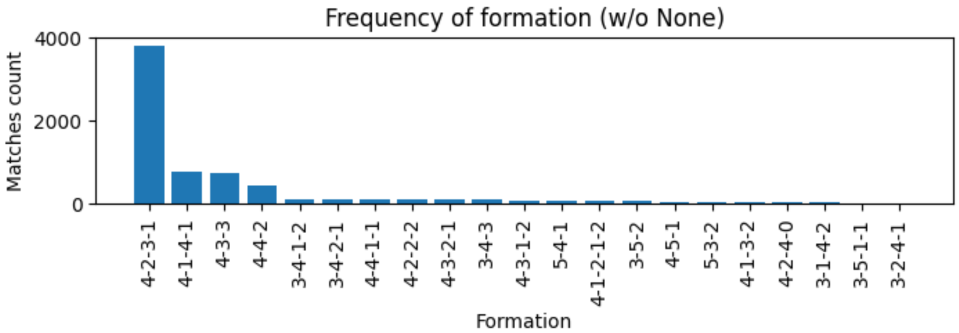


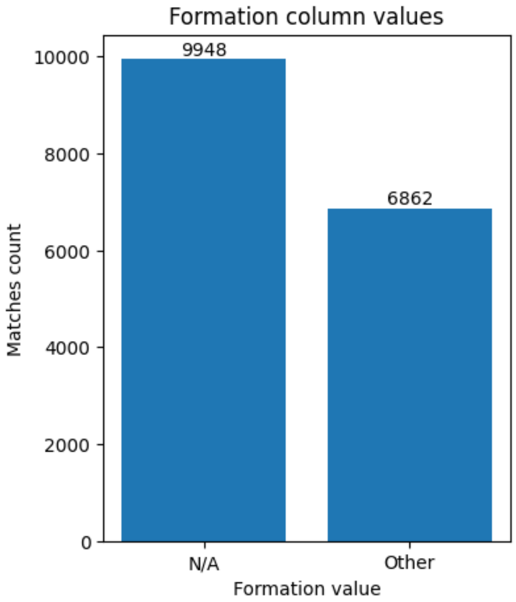
Index

- Use case and data set
- $\boxed{\text{Data exploration and assessment}}$
- Feature engineering
- Modeling
- Performance evaluation

Data Quality Assessment example: team formations







Formations are usually a major variable in soccer matches, thus it is important to properly address this feature

Decision: remove rows with no values (i.e. *None*)

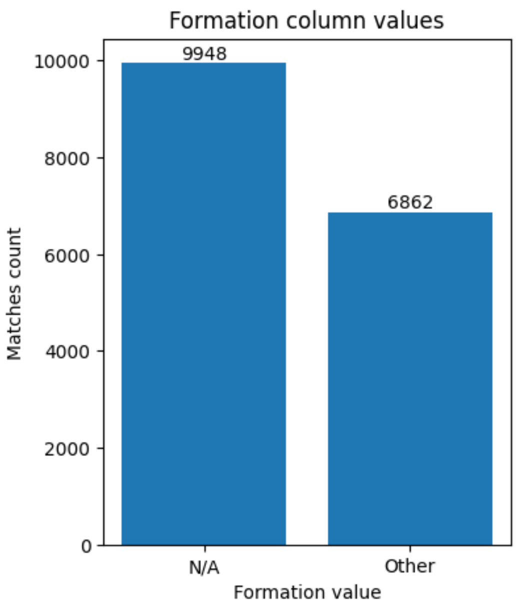
Data Quality Assessment process summary

The following data quality asessment and cleanup process was applied:

- Ⓐ **Plot distributions values for each column**
- Ⓑ **Count number of missing values** (e.g. None or N/A)
- Ⓒ **Calculate outliers and assess if they are useful**
- Ⓓ **Fix typos and errors to consolidate categories**

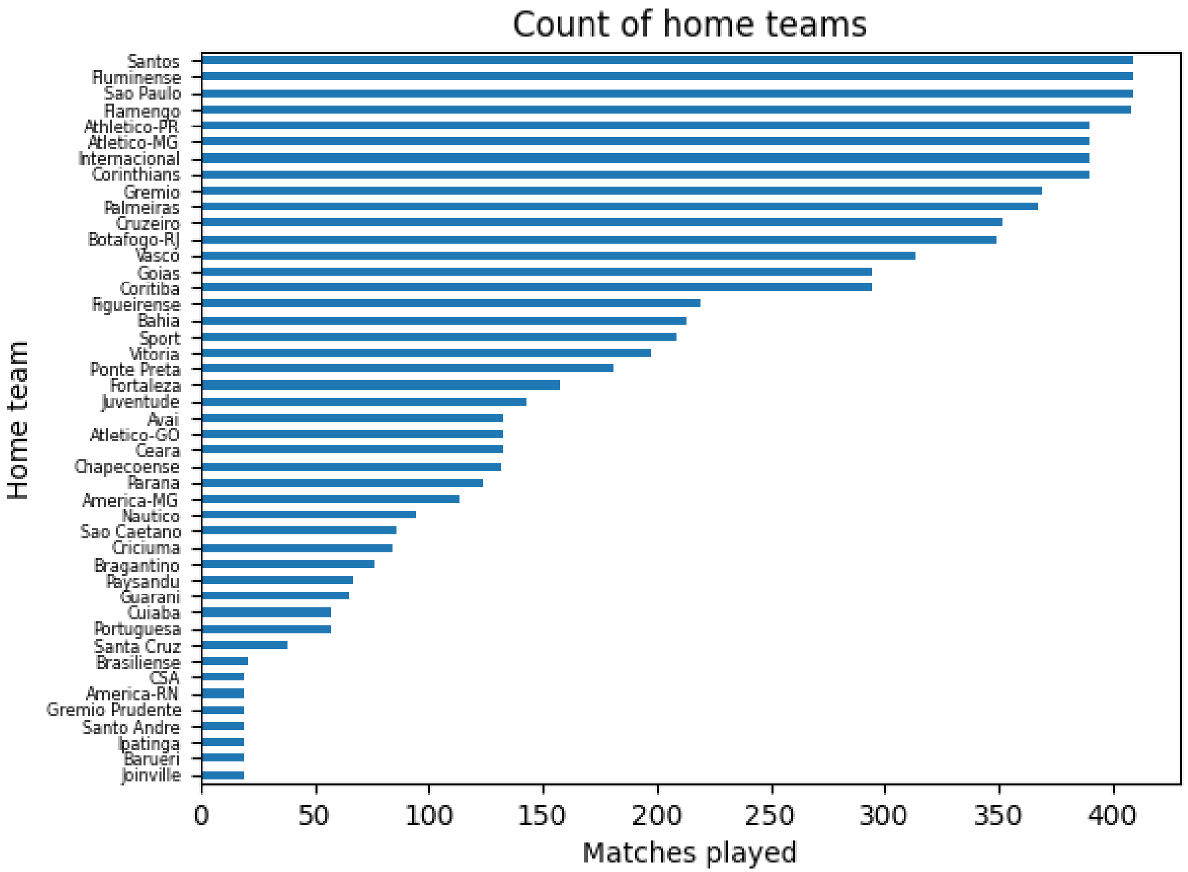
This process was repeated for all the features of interest, e.g.

- Home and away team names (HT and AT)
- Teams coach names
- Teams formation in the match
- Match stadium name and location
- Match scores
- Match final result



Decision: remove rows with no values (i.e. *None*)

Data Exploration example: number of matches played



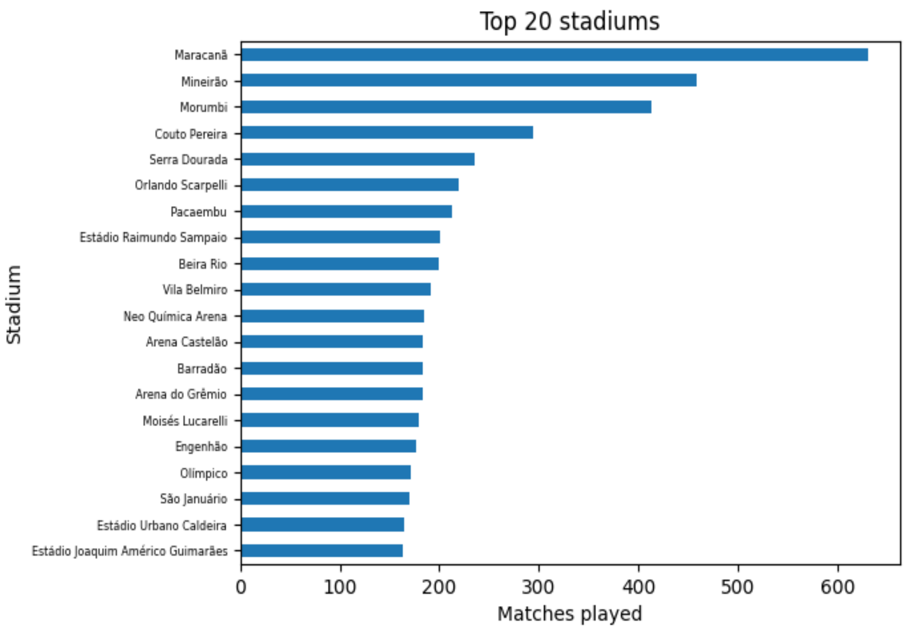
Some takeaways:

- Only 4 teams have played **all seasons** (top of the chart)
- 8 teams played only **one season** in the period (bottom)
- Approx. 16 teams played more than **half** of the seasons
- Overall, there are **45 teams** in the database
- Note that if a team played *x* matches as a home team, it will have played *2x* matches **total**

Important: the maximum number of samples of matches between the same two teams is 42 (for the top 4 ones)

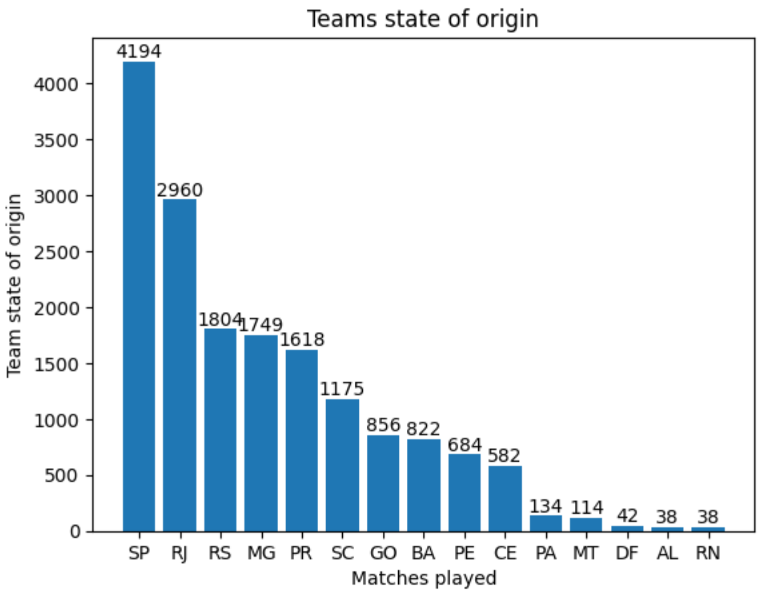
Data Exploration examples: stadiums and place of origin

Match stadium name



Big stadiums usually favor the home team, so this is a variable of interest

Clubs state of origin

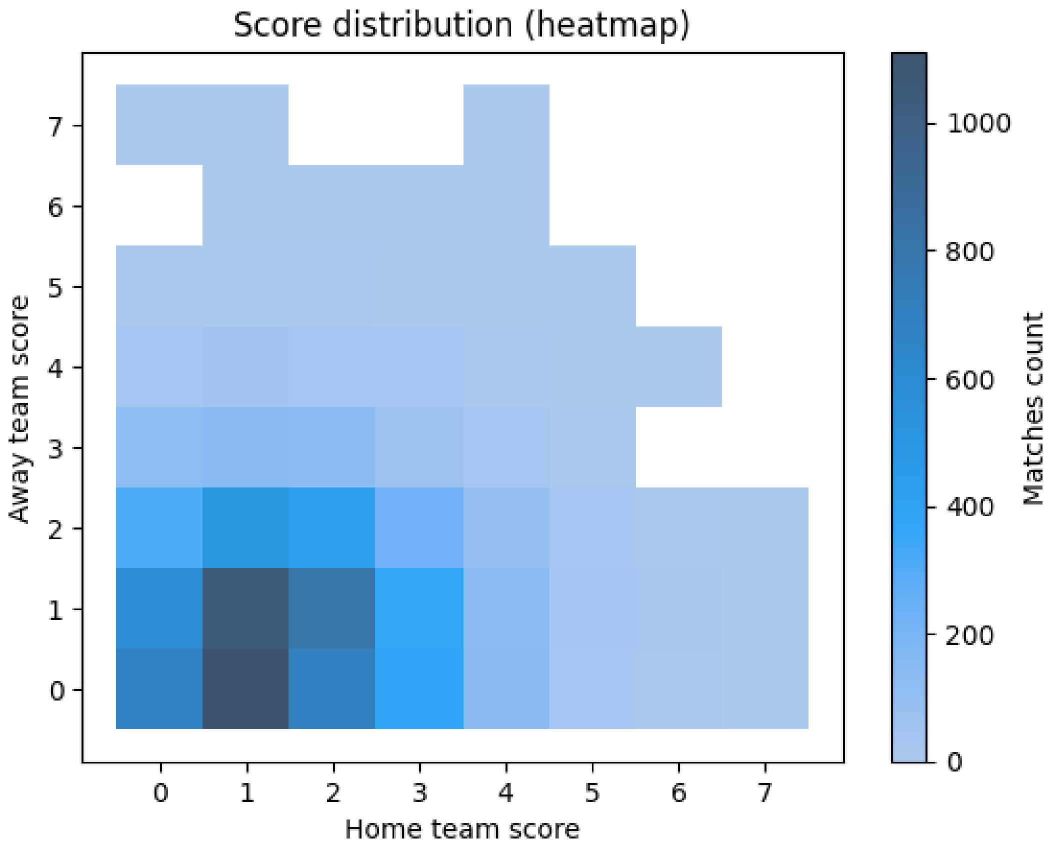


Usually it is not a variable expected to influence the outcome of a match

Data Visualization example: scores distribution

Some takeaways:

- **Most common result:** home team wins
- **Most common scores:** 1x0 home team
  1x1 draw
  2x1 home team



Index

- Use case and data set
- Data exploration and assessment
- Feature engineering
- Modeling
- Performance evaluation

Feature Engineering: we vastly expanded features count

New features were added to give historical context of the performance of both teams

1 Series of overall past results

For each variable below, we added N_p new features containing their values in the last to N_p^{th} last game each team played

- **scored_mandante:** goals scored by HT
- **scored_visitante:** goals scored by AT
- **against_mandante:** goals conceded by HT
- **against_visitante:** goals conceded by AT
- **results_mandante:** results (win, draw, loss) of HT
- **results_visitante:** results (win, draw, loss) of AT

$N_p = 5$

***30** new features*

2 Series of results between the same teams

For each variable below, we added N_v new features containing values in the last to N_v^{th} last game between the two teams of a match

- **scored_past:** goals scored by HT
- **against_past:** goals conceded by HT
- **results_past:** results (win, draw, loss) of HT

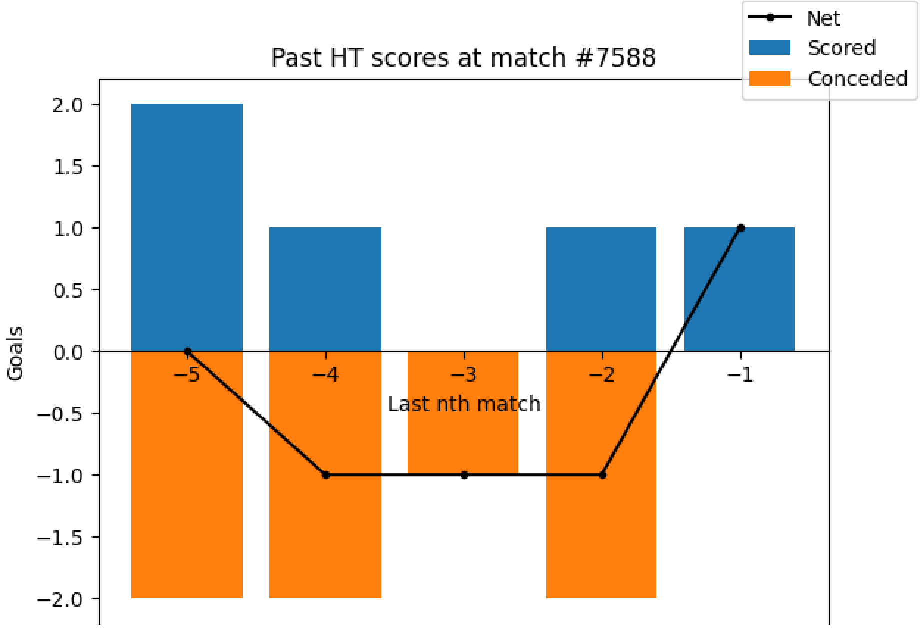
$N_v = 3$

***9** new features*

39 new features

Feature Engineering: time series of past results

Example: last N_p results of a home team



Some takeaways:

- **Recent history** is very important to build the context of a match
- Teams usually show **short term trends** towards good (or bad) performance
- Past encounters of the **same two teams** involved in a match is also important
  - can indicate a scenario with a strong team vs a weaker one
- History of **goals scored and conceded** can also indicate attack and defense qualities
- Increasing N_p/N_v values (timespan) increases **model complexity**

Feature Encoding: preparing features to feed the model

Feature	Description	Encoding	Implementation
Team names	Name of each team in a match (2 features)	**One-hot encoding** of each name	*pyspark.ml.feature. OneHotEncoder()*
Other cate-gorical features	HT/AT formation, HT/AT coach, stadium, HT/AT state, stadium (7 features)	**String indexing**	*pyspark.ml.feature. StringIndexer()*
Results (target)	Match result (win, draw or loss)	**Manual string indexing** 2: win, 1: loss, 0: draw	*convert_result()* (local function)

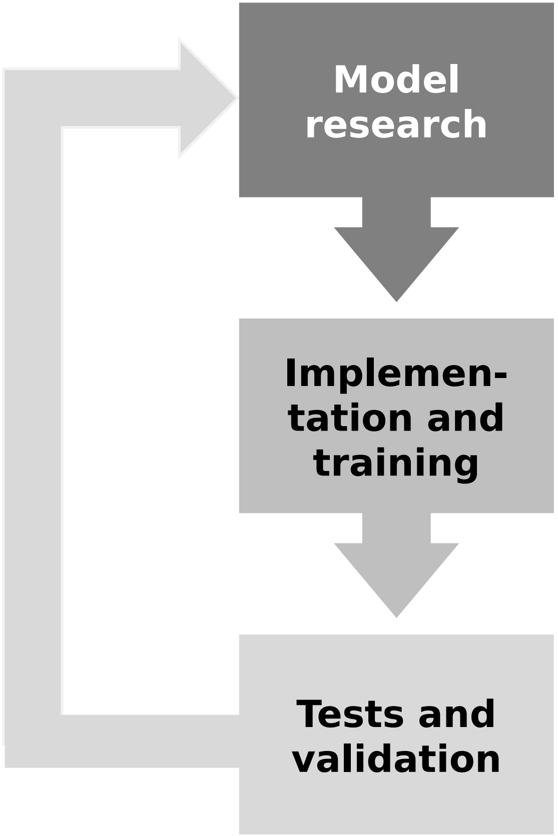
All categorical features were encoding following the scheme above

The final dataframe was eventually encoded using *pyspark.ml.feature.VectorAssembler()*

Index

- Use case and data set
- Data exploration and assessment
- Feature engineering
- Modeling
- Performance evaluation

Overview of the model development process



- Study the **available techniques** and their fitness to the problem
- Research **libraries** that provide an **implementation** of a model

- **Implement** and **run the code** to train each model
- Find/optimize **parameters** of the models to achieve best results

- Run the **trained model** with the validation-test dataset
- Define and calculate **error metrics** to assess model performance

Overview of the model development process

Model research

- Study the **available techniques** and their fitness to the problem
- Research **libraries** that provide an **implementation** of a model

3 machine learning models were assessed

1. Multilayer Perceptron
2. Gradient Boost
3. Random Forest

Implemen-tation and training

Tests and validation

- Run the **trained model** with the validation-test dataset
- Define and calculate **error metrics** to assess model performance

Overview of the model development process

Model researc

- Models were implemented and validated using **PySpark** libraries
- Parameters **manually optimized** for the problem
- **Jupyter notebook** environment (Watson Studio and Google Colaboratory)

Implementation and training

- **Implement** and **run the code** to train each model
- Find/optimize **parameters** of the models to achieve best results

Tests and validation

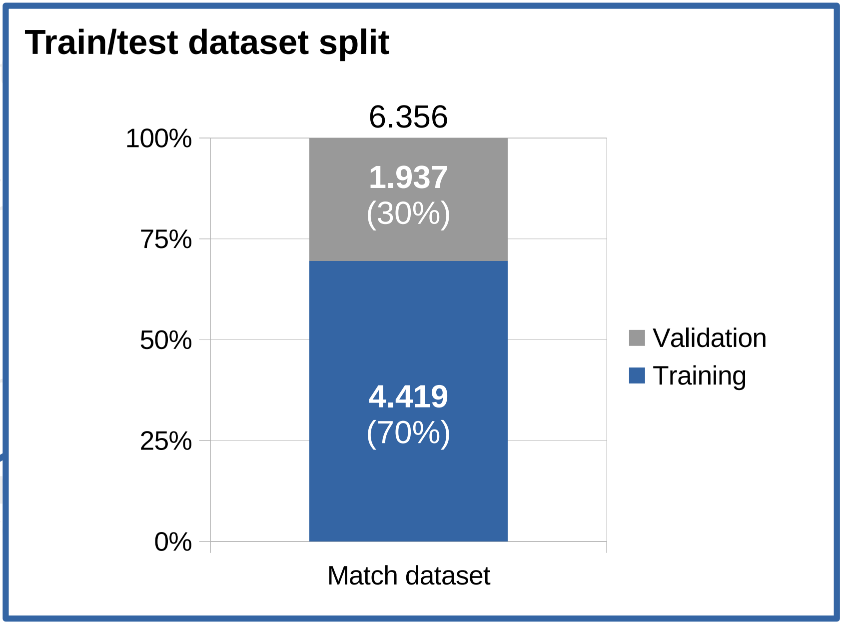
- Run the **trained model** with the validation-test dataset
- Define and calculate **error metrics** to assess model performance

Overview of the model development process

Model research

Implemen-tation and training

Tests and validation



- Run the **trained model** with the validation-test dataset
- Define and calculate **error metrics** to assess model performance

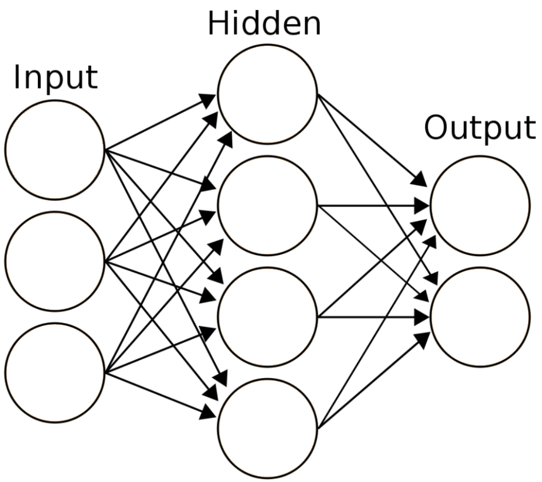
1 **Model:** Mutilayer Perceptron (MLP)

pyspark.ml.classification.MultilayerPerceptronClassifier

A multilayer perceptron (MLP) classifier is a feedforward **artificial neural network** (ANN) that uses **multiple layers of neurons** to classify data.

Each layer has **sigmoid activation function**, output layer has **softmax**.

Number of inputs has to be equal to the **size of feature vectors**. Number of outputs has to be equal to the **total number of labels** (3 in our case – W, D, L).



Example of a MLP with 3 layers

Model parameters for this project:

Number of layers: 4 (including input and output)
Hidden layers size: 64, 32

Implementation reference: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.MultilayerPerceptronClassifier.html

2 **Model:** Gradient Boosting (GBT)

pyspark.ml.classification.GBTClassifier
pyspark.ml.classification.OneVsRest

Combines multiple weak prediction models (e.g. decision trees) **sequentially** to create a **strong** predictive model

Each new model focuses on **minimizing the errors** made by the previous model, utilizing a **gradient descent optimization** approach

PySpark implementation only supports binary labels; thus, we combined it with a ***OneVsRest*** meta model

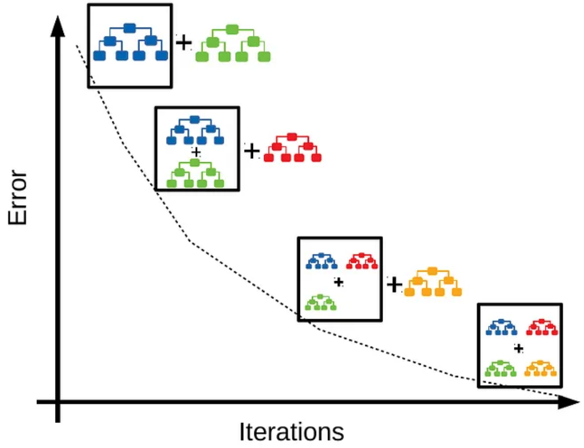


Illustration of the gradient boosting process

Model parameters for this project:

Max Bins: 256
all other parameters with default values

Implementation reference: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.GBTClassifier.html

3 **Model:** Random Forest (RF)

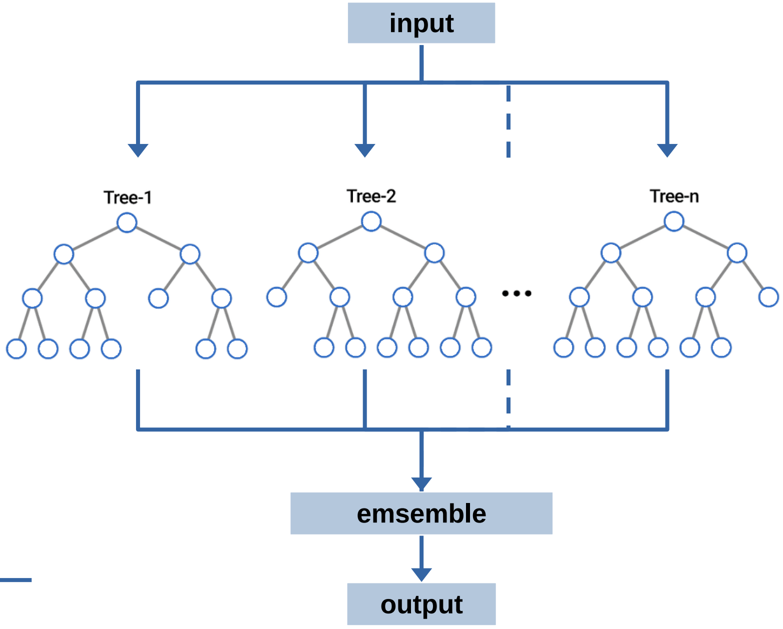
pyspark.ml.classification.RandomForestClassifier
pyspark.ml.classification.OneVsRest

Combines multiple **decision trees** in parallel

Each tree is trained independently using a **random subset of the data**

Final output is defined by a **ensemble mechanism** (e.g. voting the most common value

Despite supporting multilabel, we achieved best results when combined it with a ***OneVsRest*** meta model



Random Forest model composed of n trees

Model parameters for this project:

Max bins: 256
Max Depth: 25
Num Trees: 128

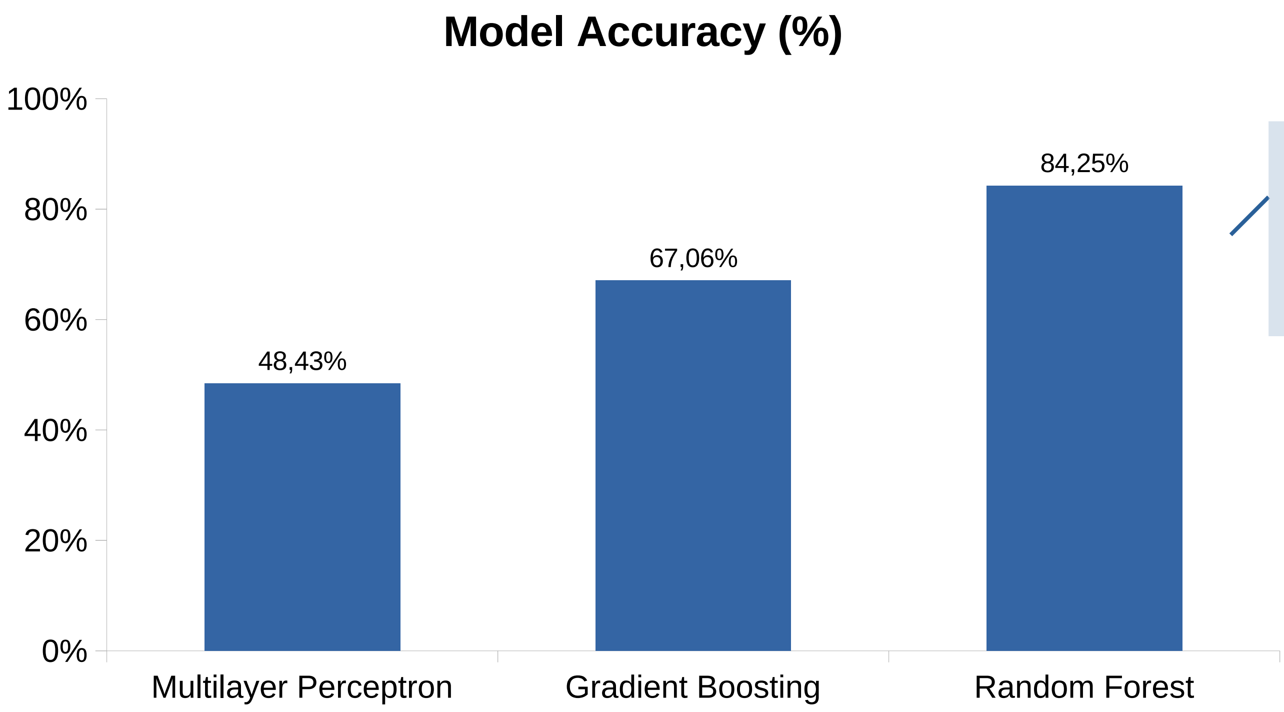
Implementation reference: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.RandomForestClassifier.html

Index

- Use case and data set
- Data exploration and assessment
- Feature engineering
- Modeling
- Performance evaluation

Evaluating the best model: **accuracy** is the main metric

Validation dataset

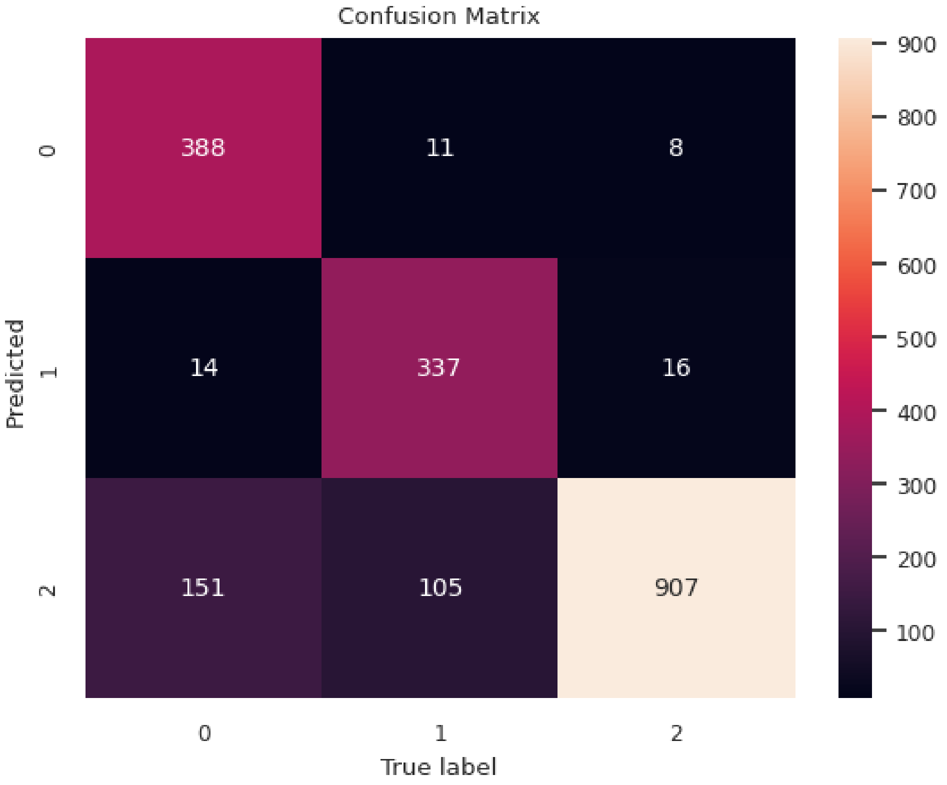


Selected model

Random Forest showed the best results for this first round of modeling

Confusion Matrix of the **Random Forest** model

Validation dataset



Label legend

0: Draw

1: Loss

2: Win

home team POV

Confusion Matrix of the **Random Forest** model

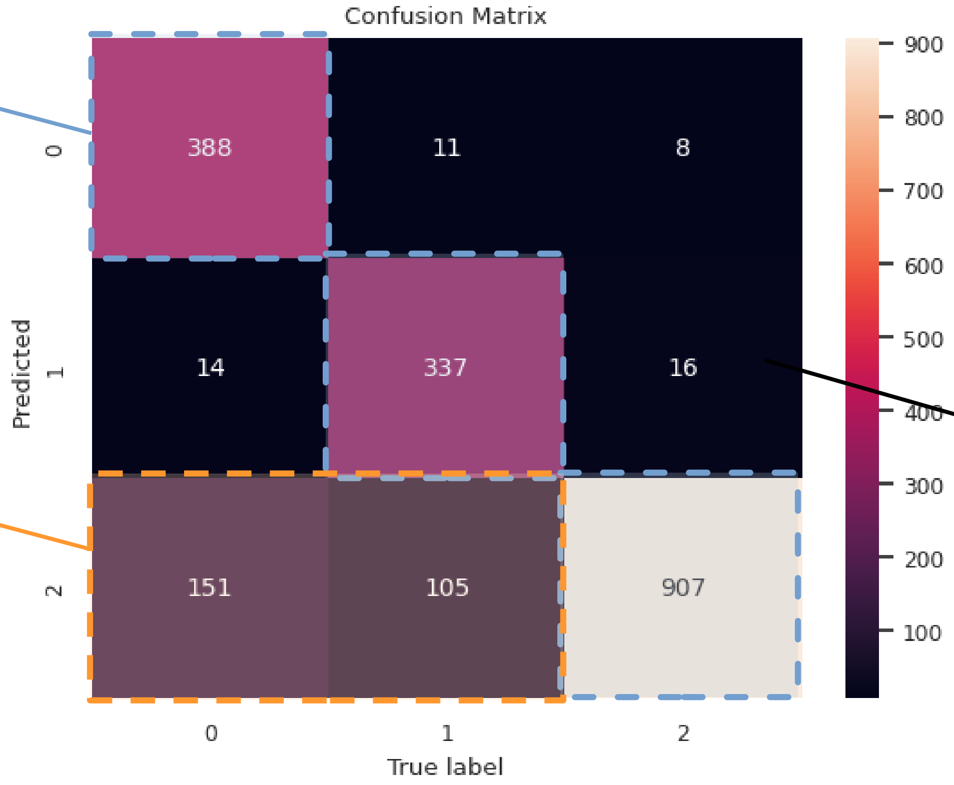
Validation dataset

Accuracy: 84,25%

Good result, considering the nature of the sport

Main room for improvement:

256 cases (13%) wrongly labeled as HT WIN



Label legend

0: Draw

1: Loss

2: Win

home team POV

Other residual errors:

All other errors sum up to only 49 cases (2,5%)

Conclusions and next steps

Important to mention the main obstacles

- Limited availability of **free computing resources**
  - Couldn’t fully develop on Watson Studio; had to be moved to Google Colab
- **Time constraints** vs. full project scope
  - Most models took several hours only to be trained
  - Overall, the capstone project surpassed expected work hours

There are still a lot of potential for continuity

- Better **hyper-parametrization** of current models
- **More data** (e.g. other leagues) and further data processing
- Research and test of other **advanced models**

palmieri.igor@gmail.com

github.com/ipalmieri

This presentation can be found at:

https://github.com/ipalmieri/advanced-data-science-capstone