Football Prediction Model

Disruptive Data Summer School, University of la Tuscia, ByTek Marketing AI, Viterbo, Italy

Team members :

- Samir Si-Mohammed
- Kristine Avagyan
- Taras Oliynyk
- Davide Momi
- Daniel Rocconi
- Emanuele D’Intino

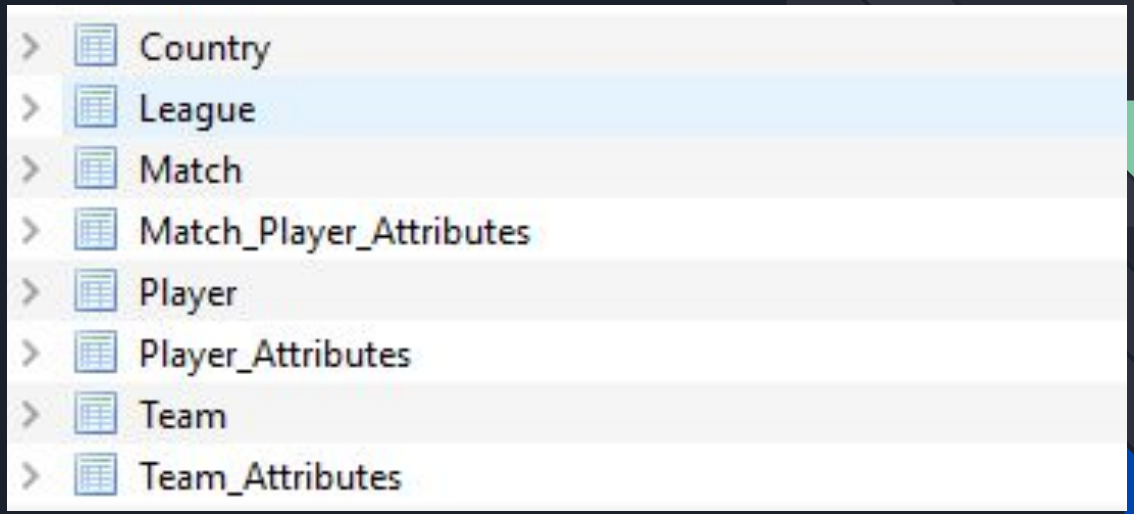
Goal

- Develop a model that makes the prediction of the outcome of a match (Win, Draw, Model) from a huge amount of previous matches

Process

1. Explore the data
2. Clean the data
3. Merge the tables
4. Build the model (Test and Optimize)

Explore the data

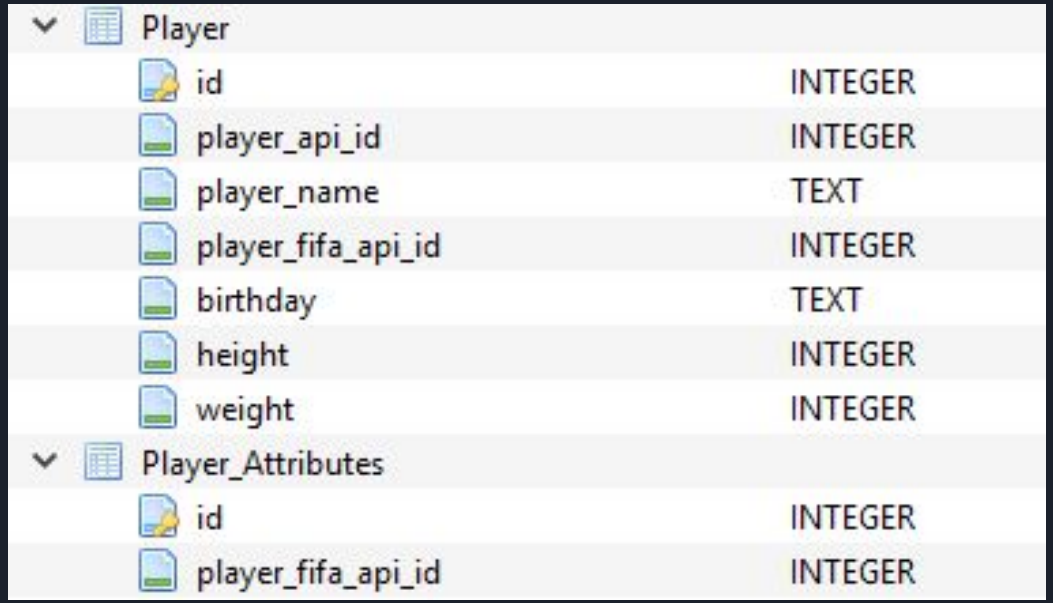


Clean the data

GBH	GBD	GBA	BSH
Filtre	Filtre	Filtre	Filtre
1.78	3.25	4	1.73
1.85	3.25	3.75	1.91
2.5	3.2	2.5	2.3
1.5	3.75	5.5	1.44
4.5	3.5	1.65	4.75
4.5	3.4	1.7	*NULL*
1.85	3.25	3.75	2.1
2.8	3.2	2.25	2.88

```
toDrop = []
for i in betWins:
    toDrop.append(i)
for i in betDraws:
    toDrop.append(i)
for i in betLoss:
    toDrop.append(i)

tempMatch = tempMatch.dropna(axis=0, how='all', thresh=None, subset=toDrop, inplace=False)

toDrop = ['goal', 'shoton', 'shotoff', 'foulcommit', 'card', 'cross', 'corner', 'possession']
tempMatch = tempMatch.drop(columns=toDrop)

#Fill the Null values of bet scores
newMatches = pd.DataFrame()
for row in tempMatch.iterrows():

    #Get the columns into Pandas dataframes
    valuesBetWins = row[1][betWins]
    valuesBetDraws = row[1][betDraws]
    valuesBetLoss = row[1][betLoss]

    #Calculate the average for each one
    avgBetWins = valuesBetWins.mean(skipna=True)
    avgBetDraws = valuesBetDraws.mean(skipna=True)
    avgBetLoss = valuesBetLoss.mean(skipna=True)

    #Replace the Null values
    for i in betWins:
        if str(row[1][i]) == 'nan':
            row[1][i] = avgBetWins
```

Merge the tables



Normalization - Standardization

```
import getNumericalData
from sklearn import preprocessing
import pandas as pd

train_set = pd.read_csv('dataTrain.csv')
test_set = pd.read_csv('dataTest.csv')

#drops the target column
y_train = train_set.final_score
x_train = train_set.drop(labels='final_score', axis=1)

y_test = test_set.final_score
x_test = test_set.drop(labels='final_score', axis=1)

#MinMax scaler, it does not distort data
mm_scaler = preprocessing.MinMaxScaler()
mm_scaled_x_train = mm_scaler.fit_transform(x_train)
mm_scaled_x_test = mm_scaler.fit_transform(x_test)

#Standard Scaler in case of the normal distribution
std_scaler = preprocessing.StandardScaler()
std_scaled_x_train = std_scaler.fit_transform(x_train)
std_scaled_x_test = std_scaler.fit_transform(x_test)

#Normalization of the data processed by Standard Scaler
normalized_std_train = preprocessing.normalize(std_scaled_x_train)
normalized_std_test = preprocessing.normalize(std_scaled_x_test)
```

Build the model

```
#Initializing classifiers
RF_clf = RandomForestClassifier(n_estimators = 200, random_state = 1, class_weight = 'balanced')
AB_clf = AdaBoostClassifier(n_estimators = 200, random_state = 2)
GNB_clf = GaussianNB(priors=None, var_smoothing=1e-05)
KNN_clf =  KNeighborsClassifier()
clfs = [RF_clf, AB_clf, GNB_clf, KNN_clf, LOG_clf]
```

```
for clf in clfs:
    pca_X_train = normalized_std_train
    pca_X_test = normalized_std_test
    clf.fit(pca_X_train, y_train)
    print("Score of {} for training set: {:.4f}.".format(clf.__class__.__name__, accuracy_score(y_train, clf.predict(pca_X_train
    print("Score of {} for test set: {:.4f}.".format(clf.__class__.__name__, accuracy_score(y_test, clf.predict(pca_X_test))))

    print(classification_report(y_train, clf.predict(pca_X_train)))
```

Compare the models

Score of AdaBoostClassifier for training set: 0.7302.
Score of AdaBoostClassifier for test set: 0.4256.

	precision	recall	f1-score	support
-1	0.84	0.75	0.79	537
0	0.69	0.55	0.61	515
1	0.69	0.82	0.75	894
accuracy			0.73	1946
macro avg	0.74	0.71	0.72	1946
weighted avg	0.73	0.73	0.73	1946

Score of KNeighborsClassifier for training set: 0.6053.
Score of KNeighborsClassifier for test set: 0.4196.

	precision	recall	f1-score	support
-1	0.52	0.67	0.58	537
0	0.52	0.41	0.46	515
1	0.72	0.68	0.70	894
accuracy			0.61	1946
macro avg	0.59	0.59	0.58	1946
weighted avg	0.61	0.61	0.60	1946

Score of GaussianNB for training set: 0.5139.
Score of GaussianNB for test set: 0.5208.

	precision	recall	f1-score	support
-1	0.48	0.65	0.55	537
0	0.38	0.41	0.40	515
1	0.67	0.49	0.56	894
accuracy			0.51	1946
macro avg	0.51	0.52	0.50	1946
weighted avg	0.54	0.51	0.52	1946

Reminder : Metrics meanings

From the confusion matrix we can create several indicators of good performances.
***TP=True Positive**, **TN=True Negative**, **FP=False Positive**, **FN=False Negative**.*

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$

Accuracy
(fraction of correct predictions)

$$\text{Recall} = \text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Recall or True Positive Rate
(fraction of correct "positive"'s over positive condition)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision
(fraction of correct predictions "positive")

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

False Positive Rate
(fraction of wrong "positive"'s over negative condition)


GitHub


slack

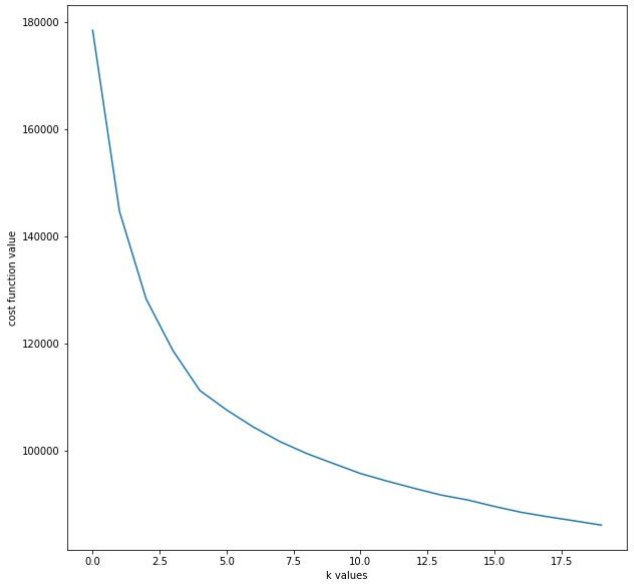

python™

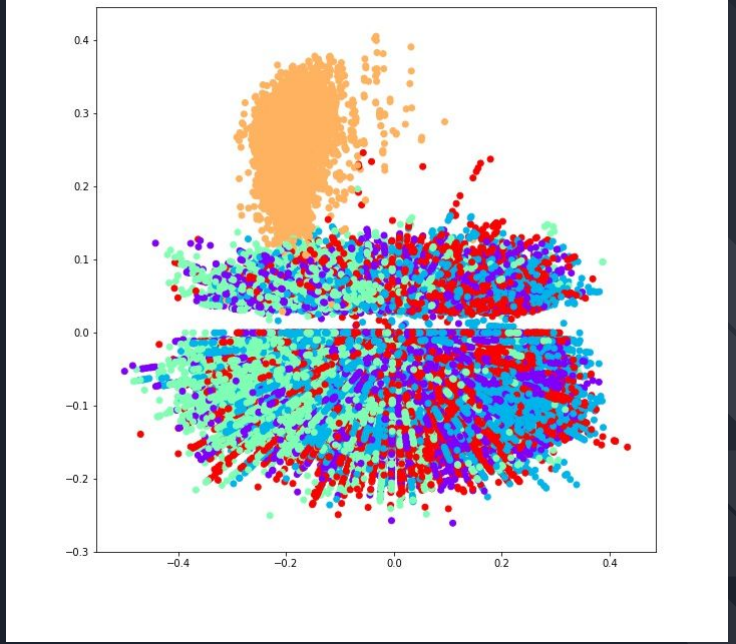

php

Optional (Players clusters)

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

A = df
for k in range (1, 21):
     # Create a kmeans model on our data, using k clusters.  random_state helps ensure that the algorithm
returns the same results each time.
    kmeans_model = KMeans(n_clusters=k, random_state=1).fit(A.iloc[:, :])

    # These are our fitted labels for clusters -- the first cluster has label 0, and the second has label
1.
    labels = kmeans_model.labels_

    # Sum of distances of samples to their closest cluster center
    interia = kmeans_model.inertia_
    print("k:",k, " cost:", interia)
```

Optional


180000
160000
140000
120000
100000
cost function value
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
k values


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2
0.4

Conclusion

1. Build a model from scratch
2. Manipulate messy data
3. Solve a real problem using ML
4. Work in group from different domains

Perspectives

1. Try our model on different leagues
2. Enrich our model with other data
3. Try to even predict the score
4. Try the same process with other sports

Thank you !

Any question?

(Filippo there you go)