matchResultPrediction

March 19, 2024

0.1 Contexte et Motivations

-> Collecte des données ->Enjeux “métiers”

0.2 Préparation des données et visualisation

->Data Cleaning (Jointures, Outliers) ->Corrélations, Boxplots, Histogramme … ->Train/Test/Valid

0.3 Benchmark de méthodes

->RandomForest ->Régression linéaire ->Boosting ->GAM ->SVM (->Clustering)

0.4 Discussion des résultats

->Interprétation “métier”

1 Introduction

Ce projet est une classification à 3 classe dans le but de prédire le résultat d'un match de football en fonction des informations que l'ont a sur le match au moment de la mi-temps. On a 3 résultats possibles pour un match, le gagnant est l'équipe à domicile, l'équipe qui ne joue pas domicile et enfin l'égalité. Dans ce projet on cherche à savoir si il est possible d'identifier à l'aide de modèles de machine learning l'un des 3 trois résultats avec un niveau de confiance suffisant. Pour ce faire nous implémenterons plusieurs méthodes et nous les comparerons au sein d'un benchmark pour choisir la meilleure.

Le plan de ce projet ce découpe en plusieurs partie. Dans un premiers temps nous ferons un nettoyage et un choix de variables à partir de la libraire statsbombpy qui est une librairie de resultats de match de football en open source. Dans un second temps nous analyserons les données entre elles en effectuant des box-plot de certaines variables que nous considerons comme significatives, nous afficherons la matrice de correlation des variables et enfin nous effectuerons une PCA afin d'essayer de réduire le nombre de variables utilisées. Enfin nous implementerons différents modèles à partir des données nettoyées et nous comparerons leurs performances afin de choisir le modèles qui nous donne les meilleurs prédictions.

Problème de classification multiclasse (3 classes)

Objectif : - Découvrir si les données d'évènements à la mi-temps permettent de prédire le résultat d'un match ; - Déterminer s' 'il existe un modèle capable d'identifier le vainqueur d'un match.

2 Contexte et Motivations

Lors de ce projet, notre première idée a été d'essayer de faire des prévisions de résultats de match de football dans le cadre des site de paris en ligne qui donne des cotes de paris en fonction des résultats des matchs. On voulait donc récupérer les côtes de différents sites de paris en ligne car on avait à dispositions de larges bases de données de cotes. Néanmoins avec cette idée, on a rencontré plusieurs problèmes. Le premier et que les sites de paris en ligne n'expliquent évidemment pas leurs modèles de prédictions, donc il n'est pas possible d'interpréter simplement les résultats des différentes variables. Suite à ça, on s'est donc dit qu'on allait utiliser les données internes de ces sites par rapport aux matchs et aux équipes sur lesquels les sites de paris en ligne construisent leurs modèles, mais ces données sont soit indisponibles pour le public, soit elle sont derrières des abonnements payants.

On a donc decidé de réorienter le projet vers la prévision du résultat du match après la mi-temps, car cela nous permet d'utiliser la librairie statsbombpy qui nous fournit des données au sein d'un match (nombre de passes, action du goals etc…). La librairie statsbombpy nous permet donc une meilleur compréhension des données, nous permettant de faire nos propres modèles et de les interpréter.

2.1 Collecte des données

L'ensemble des données ont été collectées via le package python statsbombmpy (openData).

[2]:
```
from statsbombpy import sb
```

statsbombpy est un package python opensource permettant de récupérer différentes données de match de Football de différentes compétitions. l'API complète étant payante, nous nous concentrerons ici uniquement sur les données disponibles en OpenData.

La version OpenData du package ne permettant pas de récupérer des données groupées de manière accessibles, une grande partie de notre travail a consisté en la récupération et le nettoyage des données à notre disposition.

Le package consiste en 3 fonctions principales : - sb.competitions() : permet de récupérer les identifiants de différentes compétitions, ainsi que différentes données par compétitions (compétition ,saison, genre, pays …) ; - sb.matches() : permet de récupérer les identifiants de matchs appartenant à une compétition et une saison donnée. Cette fonction permet aussi de récupérer des informations générales sur un match (équipes, scores, stade, arbitre …). De cette fonction, seules les données d'équipes et de résultat de matchs nous intéresse ; - sb.events() : permet de récupérer l'ensemble des évènements d'un match donné.

La fonction sb.events est la plus complète de ce package. En effet, chaque évènement est décrit par un type d'évènement (passe, tir, action du goal …). Chacun de ces types d'évènements possèdent ses propres indicateurs, concernant notamment les identifiants du joueur effectuant l'action, la position de ce joueur, les autres évènements liés, le TimeStamp de l'évènement …

Dans un souci de simplicité, nous nous sommes concentrés sur l'obtention du nombre d'occurrence d'évènements de chaque type pour un match donné. N'ayant aucun a-priori sur les indicateurs significatifs pour notre problème, cette méthode nous semblait la plus simple à mettre en place, en espérant produire des résultats significatifs.

3 Préparation des données et Visualisation

3.1 Construction de la donnée cible

Avec l'impossibilité de récupérer directement les données agrégées, une grande partie de notre travail a été la récupération de ces données afin de créer le dataset d'intérêt "à la main".

[3]:
```
###Bloc import des libraries
from statsbombpy import sb
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import seaborn as sns

import pandas as pd

import category_encoders

import warnings
warnings.filterwarnings("ignore")

from os.path import exists
```

Regardons d'abord les indicateurs de compétitions à notre disposition en openData.

[4]:
```
competitionData = sb.competitions()
```

[5]:
```
competitionData.head()
```

[5]:

	competition_id	season_id	country_name	competition_name	competition_gender	competition_youth	competition_international	season_name	match_updated	match_updated_360
0	9	27	Germany	1. Bundesliga	male	False	False	2015/2016	2023-12-12T07:43:33.436182	None
1	1267	107	Africa	African Cup of Nations	male	False	True	2023	2024-02-14T05:41:27.566989	None
2	16	4	Europe	Champions League	male	False	False	2018/2019	2023-03-07T12:20:48.118250	2021-06-13T16:17:31.694
3	16	1	Europe	Champions League	male	False	False	2017/2018	2021-08-27T11:26:39.802832	2021-06-13T16:17:31.694
4	16	2	Europe	Champions League	male	False	False	2016/2017	2021-08-27T11:26:39.802832	2021-06-13T16:17:31.694

```
  match_available_360             match_available
0                None  2023-12-12T07:43:33.436182
1                None  2024-02-14T05:41:27.566989
2                None  2023-03-07T12:20:48.118250
3                None  2021-01-23T21:55:30.425330
4                None            2020-07-29T05:00
```

Regardons les différentes compétitions à notre disposition :

[6]:
```
competitionData['country_name'].unique()
```

[6]:
```
array(['Germany', 'Africa', 'Europe', 'Spain', 'England', 'International',
       'India', 'Argentina', 'France', 'United States of America',
       'North and Central America', 'Italy'], dtype=object)
```

[7]:
```
competitionData['competition_name'].unique()
```

[7]:
```
array(['1. Bundesliga', 'African Cup of Nations', 'Champions League',
       'Copa del Rey', "FA Women's Super League", 'FIFA U20 World Cup',
       'FIFA World Cup', 'Indian Super league', 'La Liga',
       'Liga Profesional', 'Ligue 1', 'Major League Soccer',
       'North American League', 'NWSL', 'Premier League', 'Serie A',
       'UEFA Euro', 'UEFA Europa League', "UEFA Women's Euro",
       "Women's World Cup"], dtype=object)
```

Dans le dataset disponible en openData, nous avons donc à notre disposition différentes compétitions (masculines/féminines, jeune, nationale/internationale) des principaux pays dans le monde du Football (Europe, Amérique ...). Afin de récupérer le maximum de données, nous avons choisi de sélectionner l'ensemble des compétitions à notre disposition.

Regardons un exemple de matchs à notre disposition, pour une compétition donnée (Bundesliga 2015/2016).

[8]:
```
matchStats = sb.matches(competition_id=9, season_id=27)
```

[9]:
```
matchStats.head()
```

[9]:
```
   match_id  match_date      kick_off             competition     season  \
0   3890561  2016-05-14  15:30:00.000  Germany - 1. Bundesliga  2015/2016
1   3890505  2016-04-02  15:30:00.000  Germany - 1. Bundesliga  2015/2016
2   3890511  2016-04-08  20:30:00.000  Germany - 1. Bundesliga  2015/2016
3   3890515  2016-04-09  15:30:00.000  Germany - 1. Bundesliga  2015/2016
4   3890411  2015-12-20  16:30:00.000  Germany - 1. Bundesliga  2015/2016

       home_team            away_team  home_score  away_score match_status  \
0     Hoffenheim           Schalke 04           1           4    available
1  Bayern Munich  Eintracht Frankfurt           1           0    available
2  Hertha Berlin          Hannover 96           2           2    available
```

```
3   Hamburger SV         Darmstadt 98           1           2    available
4  Hertha Berlin         FSV Mainz 05           2           0    available

   … last_updated_360 match_week competition_stage                stadium  \
0  …             None         34    Regular Season          PreZero Arena
1  …             None         28    Regular Season          Allianz Arena
2  …             None         29    Regular Season  Olympiastadion Berlin
3  …             None         29    Regular Season       Volksparkstadion
4  …             None         17    Regular Season  Olympiastadion Berlin

          referee           home_managers        away_managers data_version  \
0     Felix Brych       Julian Nagelsmann  André Breitenreiter        1.1.0
1   Florian Meyer  Josep Guardiola i Sala           Niko Kovač        1.1.0
2  Benjamin Brand              Pál Dárdai       Daniel Stendel        1.1.0
3    Peter Sippel          Bruno Labbadia        Dirk Schuster        1.1.0
4    Peter Sippel              Pál Dárdai       Martin Schmidt        1.1.0

  shot_fidelity_version xy_fidelity_version
0                     2                   2
1                     2                   2
2                     2                   2
3                     2                   2
4                     2                   2

[5 rows x 22 columns]
```

Pour une compétition données, seules les premières colonnes nous sont essentielles pour notre problème : home_team, away_team, home_score, away_score.

De ces données, nous pouvons alors classifier chaque match dans 3 catégories : Home Win, Away Win, Draw.

De ce dataset, nous pouvons donc construire notre target : Pour chaque match de chaque compétition, il nous faut alors déterminer le vainqueur de chaque match en comparant leurs scores.

[10]:
```python
### Création de la fonction de détermination du vainqueur
def result(row):
    resultLabel = ""

    if row['home_score'] > row['away_score']:
        resultLabel = "Home"

    elif row['home_score'] < row['away_score']:
        resultLabel = "Away"

    else:
        resultLabel = "Draw"
```

```
    return resultLabel
```

Nous allons maintenant construire le dataset de covariables pour chaque match.

3.2 Construction des variables

Objectifs : - Récuperer pour chaque compétition chaque match (par son identifiant) ; - Pour chaque match, récupérer le nombre d'évènements à la mi-temps.

On va d'abord construire l'ensemble des comptes d'évènements pour une compétition donnée :

[11]:
```
#Variables disponibles dans le package statsbomb
EVENT_NAMES = ["Ball Receipt*", "Ball Recovery", "Dispossessed", "Duel",␣
 ↪"Block", "Offside", "Clearance", "Interception", "Dribble", "Shot",␣
 ↪"Pressure", "Substitution", "Own Goal Against", "Foul Won", "Foul␣
 ↪Committed", "Goal Keeper", "Bad Behaviour", "Player On", "Player Off",␣
 ↪"Shield", "Pass", "50/50", "Tactical Shift", "Error", "Miscontrol",␣
 ↪"Dribbled Past", "Injury Stoppage", "Referee Ball-Drop", "Carry"]


def oneCompetionEventsCount(competitionId, seasonId):

    try :
        ##On récupère l'ensemble des matchs de la compétition
        matchStats = sb.matches(competition_id= competitionId, season_id=␣
 ↪seasonId)

        #On récupère les colonnes utiles pour déterminer la colonne cible
        matchStats = matchStats[['match_id', 'home_team', 'away_team',␣
 ↪'home_score', 'away_score']]

        #On crée la colonne cible
        matchStats['Result'] = matchStats.apply(result, axis = 1)

        #On indexe par matchId (unique)
        matchStats.set_index('match_id', inplace=True)

        for matchId in matchStats.index:
            #Récupérer tout les events du matchs
            matchEventFull = sb.events(match_id = matchId)

            #Ne conserver que ceux pour lesquels period = 1 (1ere mi-temps)
            matchEventFull = matchEventFull.loc[matchEventFull['period'] == 1]

            #On compte le nombre d'occurences de chaque event par équipe
            eventCountByTeam = matchEventFull.groupby(by=['team',␣
 ↪'type'])['type'].count().unstack(level=1)
```

```
            #On récupère les noms de chaque équipe
            homeTeamName = matchStats.loc[matchId,'home_team']
            awayTeamName = matchStats.loc[matchId,'away_team']


            #On ajoute les colonnes features intéressantes au dataframe␣
↪matchStats
            for name in EVENT_NAMES:
                matchStats.loc[matchId, "Home-" + name] = eventCountByTeam.
↪loc[homeTeamName, name] if name in eventCountByTeam.loc[homeTeamName].index␣
↪else 0
                matchStats.loc[matchId, "Away-" + name] = eventCountByTeam.
↪loc[awayTeamName, name] if name in eventCountByTeam.loc[awayTeamName].index␣
↪else 0


            ###Si valeurs manquantes, on rempli les valeurs vides de 0␣
↪(Evenement non rencontré)
            matchStats.fillna(0, inplace= True)

        #On crée le dataset "final"
        matchStats = matchStats.iloc[:, 4:]
        target = matchStats['Result']
        matchStats = matchStats.iloc[:,1:].astype(int)

        return matchStats, target
    except:
        #Message d'erreur
        print("Erreur dans la fonction oneCompetitionEventsCount")
```

On peut alors boucler sur l'ensemble des compétitions pour construire les datasets finaux. Afin d'éviter de recréer les datasets à chaque fois, on les stockes dans un fichier .csv externe.

[12]:
```
def allCompetitionEventsCount():
    try :
        #Si le fichier existe déjà, inutile de le recréer
        if exists("Data/fullMatchData.csv") & exists("Data/fullTargetData.csv"):
            return pd.read_csv("Data/fullMatchData.csv"), pd.read_csv("Data/
↪fullTargetData.csv")

        else:

            #On récupère l'ensemble des compétitions
            competitionData = sb.competitions()

            #On initialise les dataframes
            fullMatchData = pd.DataFrame()
```

```
            fullTargetData = pd.DataFrame()

            #On boucle sur l'ensemble des compétitions
            for idx in competitionData.index:

                #On récupère les Ids nécessaires identifier une compétition
                competitionId, seasonId = competitionData.loc[idx,␣
↪['competition_id', 'season_id']]

                #On construit les datasets de variables/targets pour cette␣
↪compétition
                matchData, target = oneCompetionEventsCount(competitionId,␣
↪seasonId)

                #On les ajoute aux datasets globaux
                fullMatchData = pd.concat([fullMatchData, matchData], sort=␣
↪False)
                fullTargetData = pd.concat([fullTargetData, target])

            #Dans le cas ou des évènements seraient manquants, on complète les␣
↪valeurs vides par 0
            fullMatchData.fillna(0, inplace= True)

            #On stocke les datasets finaux
            fullMatchData.to_csv("Data/fullMatchData.csv", index= False)
            fullTargetData.to_csv("Data/fullTargetData.csv", index= False)

            return fullMatchData, fullTargetData


    except:
        print("Erreur dans la fonction allCompetitionEventsCOunt")
```

[13]:
```
fullMatchData, fullTargetData = allCompetitionEventsCount()
```

Expliquer shape

[14]:
```
fullMatchData.shape
```

[14]: (3316, 58)

[15]:
```
fullMatchData.head()
```

[15]:

	Home-Ball Receipt*	Away-Ball Receipt*	Home-Ball Recovery	\
0	241	262	23	
1	322	178	23	
2	342	141	19	
3	304	130	30	

```
4                379                160                  25

   Away-Ball Recovery  Home-Dispossessed  Away-Dispossessed  Home-Duel  \
0                  19                  4                  5         14
1                  27                  3                  5         21
2                  24                  4                  3         14
3                  20                  6                  4         18
4                  15                  5                  3         11

   Away-Duel  Home-Block  Away-Block  …  Home-Miscontrol  Away-Miscontrol  \
0         11          12          10  …               12                7
1         18           7           1  …                6                8
2         22           7          10  …                9               14
3         24           5           7  …                7                5
4         20          11           8  …                5                5

   Home-Dribbled Past  Away-Dribbled Past  Home-Injury Stoppage  \
0                   2                   3                     1
1                   2                   3                     1
2                   7                   5                     1
3                   5                   3                     1
4                   3                   4                     1

   Away-Injury Stoppage  Home-Referee Ball-Drop  Away-Referee Ball-Drop  \
0                     2                       0                       0
1                     2                       2                       2
2                     0                       0                       0
3                     1                       1                       1
4                     0                       0                       0

   Home-Carry  Away-Carry
0         224         204
1         251         102
2         292         126
3         246          81
4         305         110

[5 rows x 58 columns]
```

On obtient alors un dataset de taille raisonnable, 3316 observations de 58 variables numériques entières. De manière prévisible, certaines variables sont nettement plus élevées que d'autres en moyenne, certains évènements intervenant de manière beaucoup plus fréquente que d'autres ; ceci justifie l'utilisation d'un Scaler dans les différents modèles évalués.

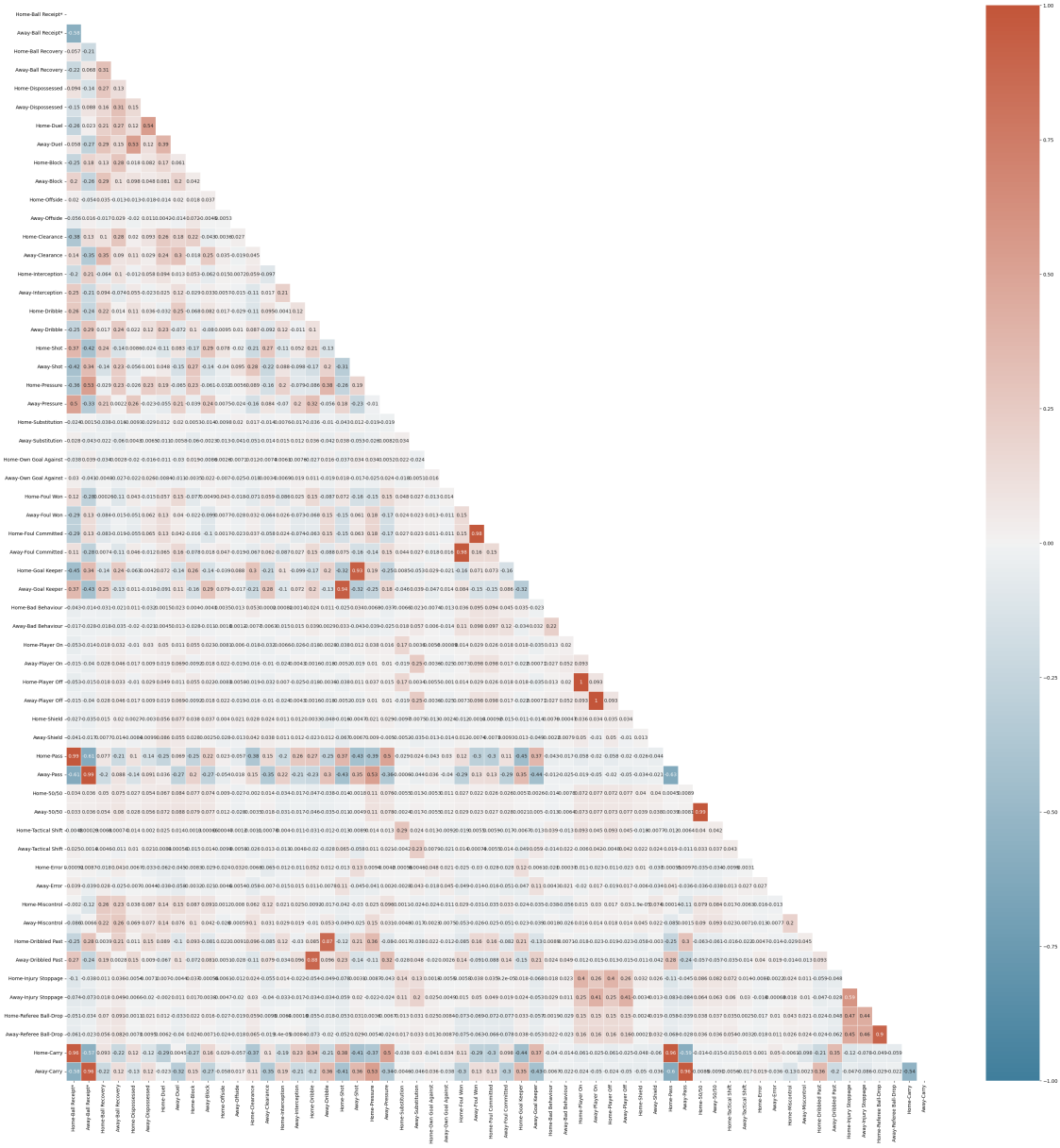
4 Data Visualization

Avant d'implémenter les modèles de classification, on va d'abord s'attaquer à la visualisation des données, afin de repérer certaines variables significatives.

4.1 Corrélation entres variables

[16]:
```
#Palette de contraste
COLOR_MAP = sns.diverging_palette(230, 20, as_cmap=True)

#Figure matplotlib
fig, ax = plt.subplots(figsize= (40,40))
#
sns.heatmap(fullMatchData.corr(), #Plot de la matrice de corrélation
            mask = np.triu(np.ones_like(fullMatchData.corr(), dtype = bool)),␣
 ↪#On n'affiche que la partie triangulaire inférieure de la matrice de␣
 ↪corrélation
            cmap = COLOR_MAP,
            annot = True, #On affiche les valeurs du coeeficient de corrélation
            linewidths = .5,
            vmin = -1,
            vmax = 1)
```

[16]:
```
<AxesSubplot: >
```



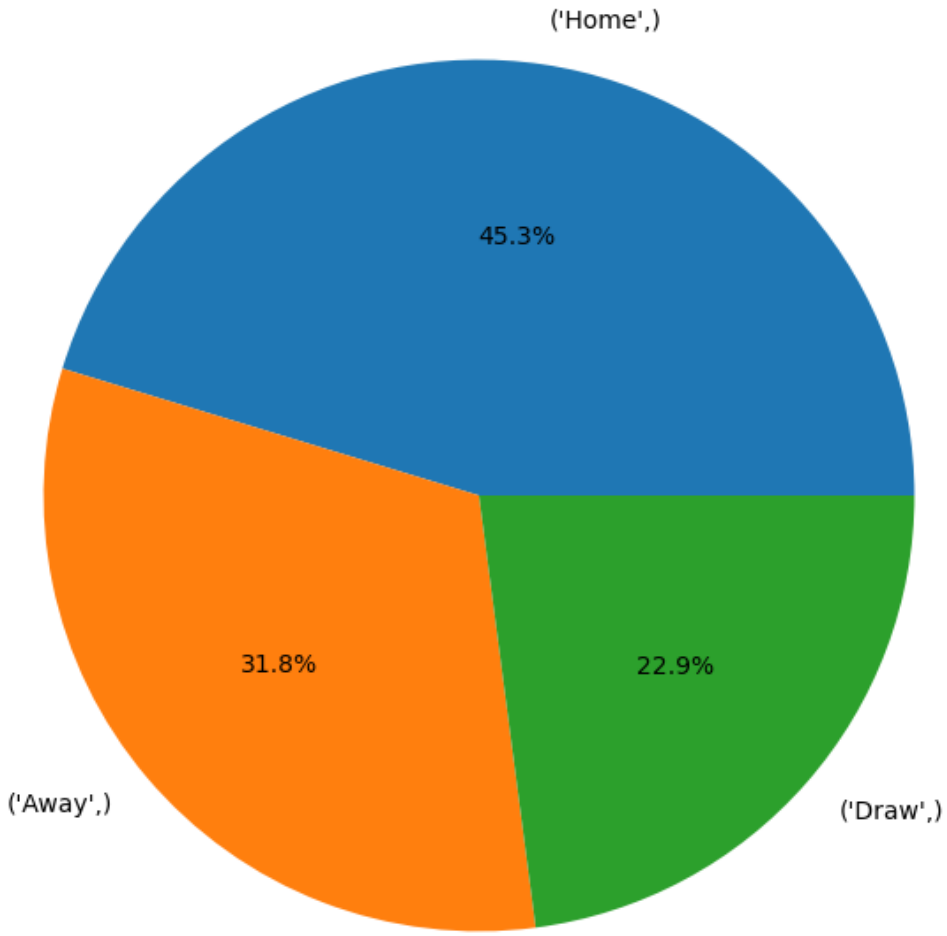
On peut alors remarquer que certaines variables sont très fortement corrélées entre elles ; en regard de la dénomination de ces variables, cela n'a rien d'étonnant. En effet, on remarque que le nombre de passe d'un match est fortement corrélés avec une possesion de balle élevée, qu'un 50/50 se joue toujours entre une équipe et l'équipe adverse, ou qu'un retrait de joueur sur le terrain et toujours accompagné d'une entrée de joueur. On remarque également qu'un tir d'une équipe et accompagné d'une action du goal de l'équipe adverse.

Aucune variable ne se trouve être parfaitement négativement corrélées, mais certaines interactions sont à remarquer : Le nombre de réception de passe est négativement corrélés au nombre d'actions du goal d'une équipe, le nombre de passe d'une équipe est négativement corrélés au nombre de passes de l'équipe adverse …

La plupart des autres variables ne semblent pas avoir d'interactions linéaires entres elles, mais regardons maintenant comment cela se comporte vis-à-vis des résultats des matchs.

4.2 Visualisations

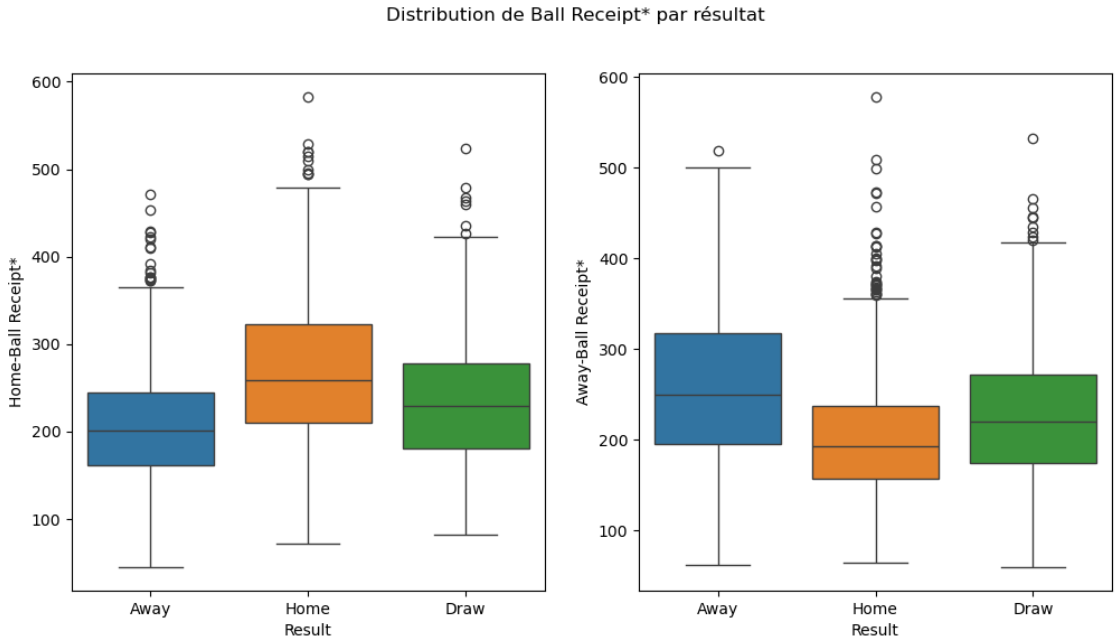
[17]:
```
plt.figure(figsize= (8, 8))
plt.pie(fullTargetData.value_counts(), labels = fullTargetData.value_counts().
  ↪index, autopct= '%1.1f%%')
plt.show()
```



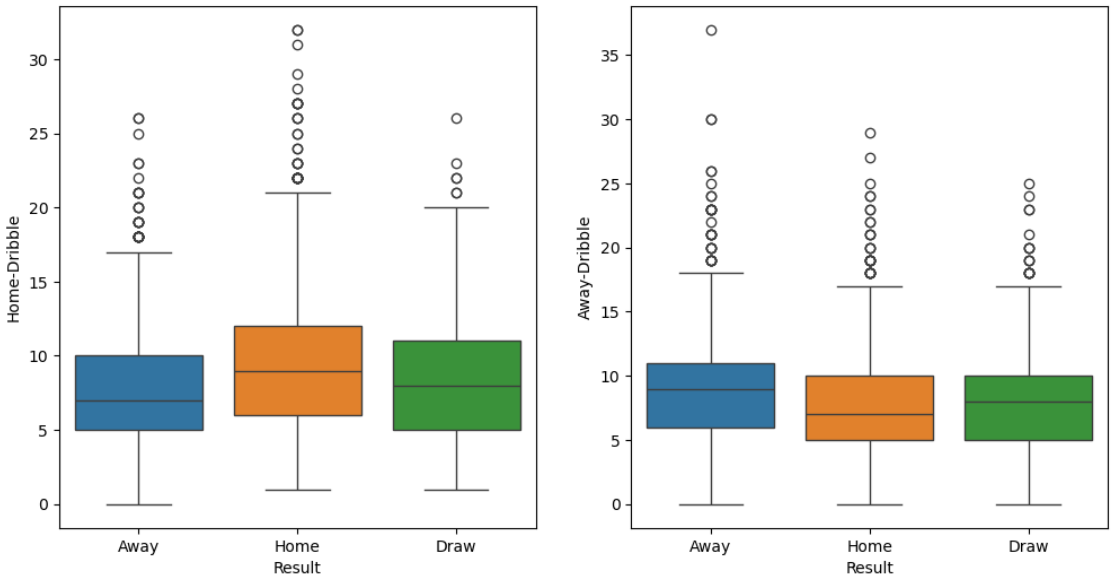
On remarque alors une disparité des données : prêt de la moitié des matchs sont gagnés par l'équipe à domicile.

On va s'intéresser alors à certaines visualisation significatives pour notre problème.

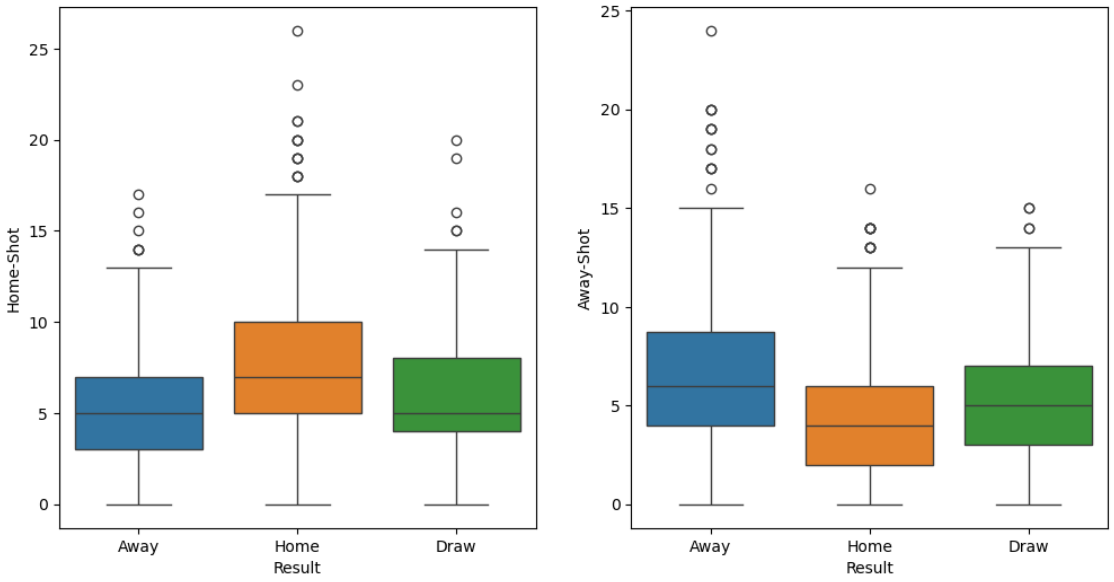
[18]:
```
fullData = pd.concat([fullMatchData, fullTargetData], axis=1)

for eventType in ['Ball Receipt*', 'Dribble', 'Shot', 'Pressure', 'Goal␣
 ↪Keeper', 'Pass', 'Carry']:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))

    sns.boxplot(y='Home-'+eventType, x='Result', data= fullData, hue =␣
 ↪'Result', ax= ax1)
    sns.boxplot(y='Away-'+eventType, x='Result', data= fullData, hue =␣
 ↪'Result', ax= ax2)
    plt.suptitle(f'Distribution de {eventType} par résultat')
```



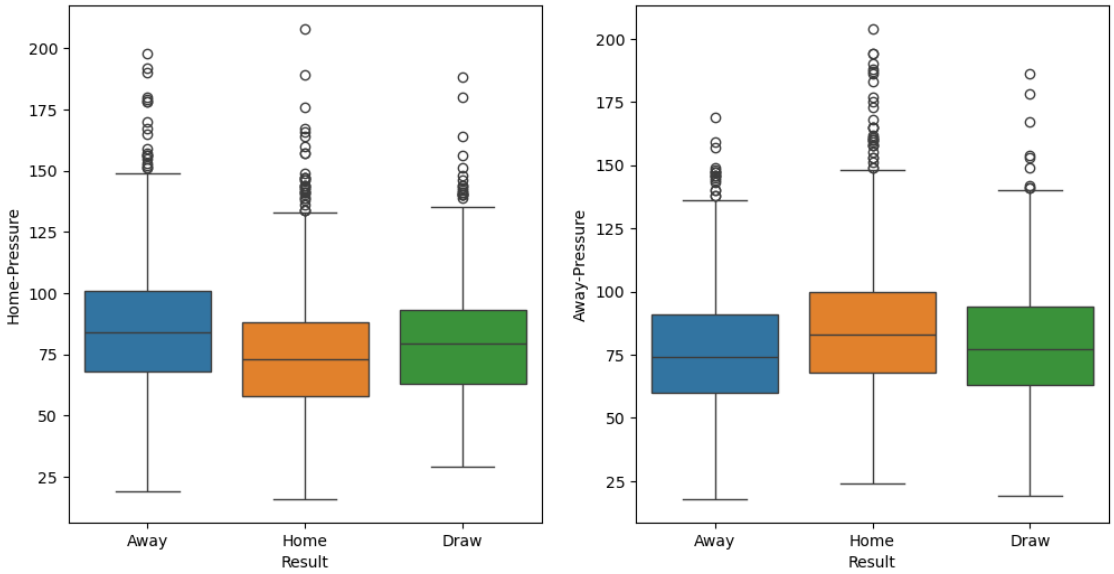
Distribution de Dribble par résultat



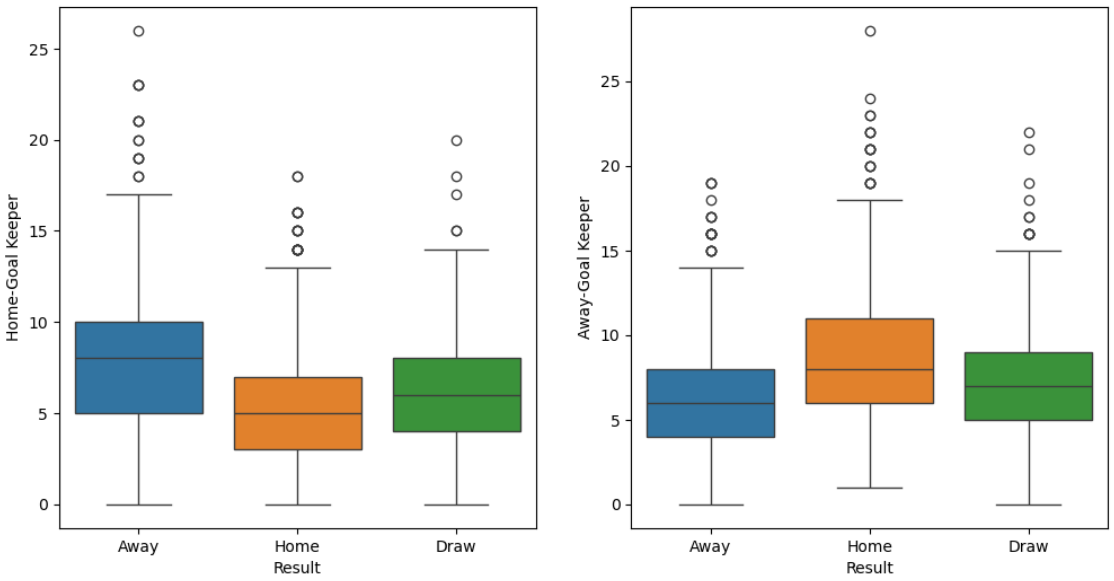
Distribution de Shot par résultat

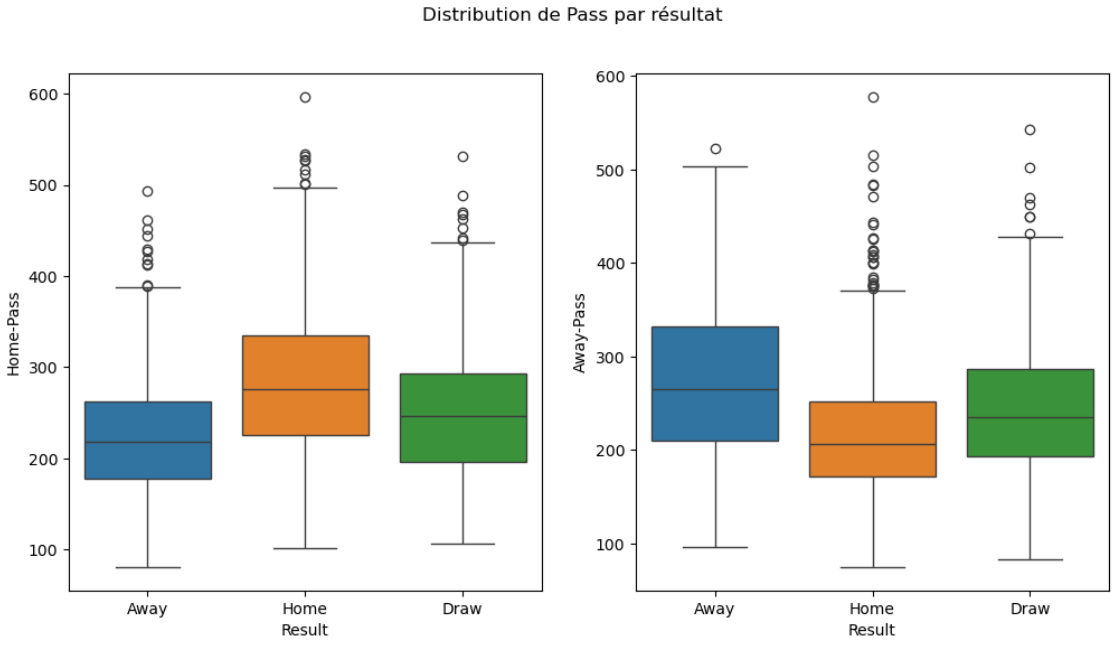


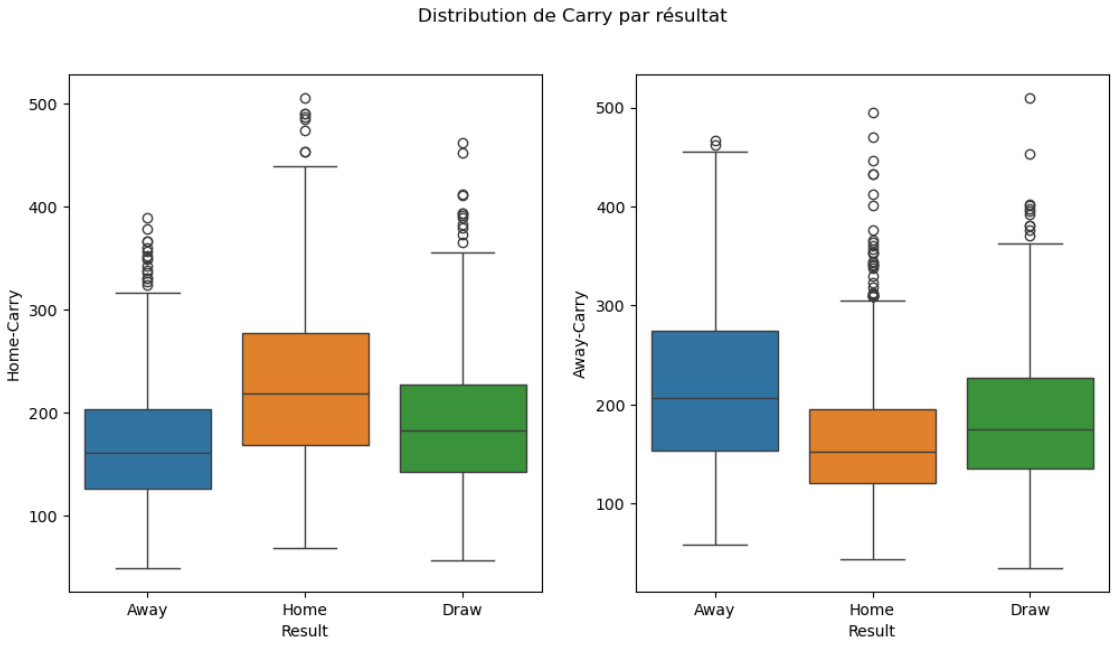
Distribution de Pressure par résultat



Distribution de Goal Keeper par résultat







Nous avons ici sélectionné les écarts les plus signifiants dans les représentations boxplots des variables en fonction du résultat.

On remarque alors que les variables les plus signifiantes sont sensiblement logiques dans le contexte : - Ball Receipt, Dribble, Pressure font surtout référence à une notion de possession de ballon ; en effet, plus une équipe possède le ballon, plus elle développe son jeu de passe et a sensiblement plus

de chances de gagner le match ; - L'autre aspect à considérer et l'aspect Shot, Goal Keeper ; plus le nombre de tirs et important, plus le nombre d'actions du Goal de l'équipe adverse est important (On retrouve ici la corrélation notée précédemment), plus l'équipe a de chance de gagner.

On suppose donc que les modèles évalués vont donner de l'importance à ses variables-ci.

5 Modélisation

On commence par séparer notre jeu de test en jeu de test d'entraînement et de test, afin d'évaluer nos modèles après entraînement sur des données indépendantes.

[19]:
```
from sklearn.model_selection import train_test_split


xTrain, xTest, yTrain, yTest = train_test_split(fullMatchData, fullTargetData,␣
↪stratify= fullTargetData, random_state=42)
```

6 PCA

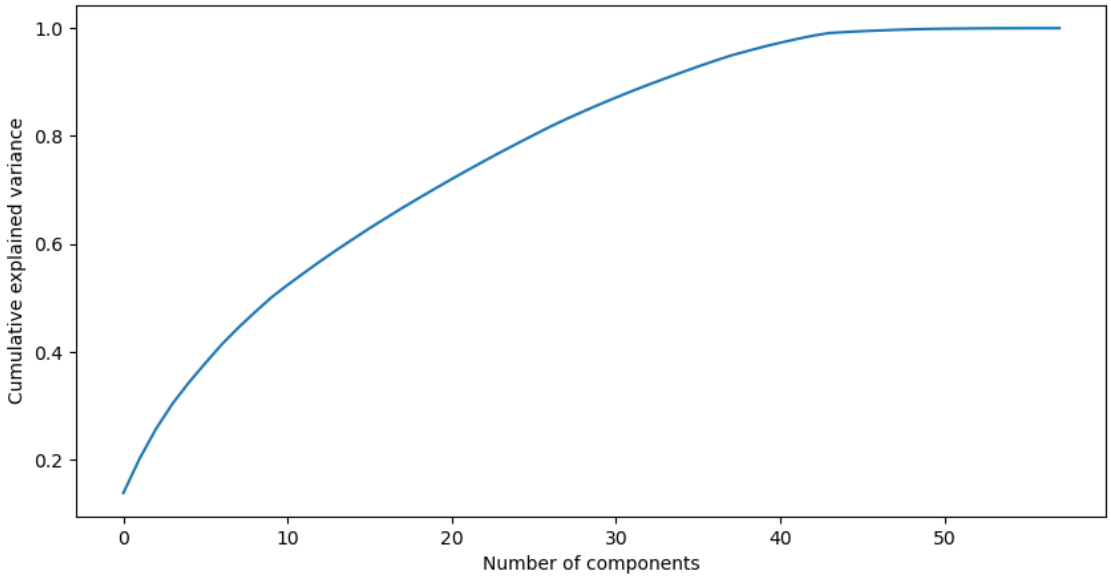
Avant d'implémenter les modèles et au vu du nombre importants de variabeles à notre disposition, regardons s'il est utile d'appliquer une PCA à notre dataset.

[20]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

pca = PCA()
scl = StandardScaler()

pca.fit(scl.fit_transform(xTrain))
```

[20]:
```
PCA()
```

[21]:
```
plt.figure(figsize=(10,5))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

[21]:
```
Text(0, 0.5, 'Cumulative explained variance')
```



Comme on peut le voir sur le graphique de la variance expliquée, on ne distingue aucun coude net permettant une PCA sgnificative. De plus, pour expliquer 95% de la variance initiale, il faudrait sélectionner 40 variables sur les 58 initiales. En effet les dernières valeurs (40 à 50) ne rajoute pas d'explication à la variance car elle sont identiquement corrélées.

Cette PCA peut s'expliquer du fait que la plupart des variables ne dependent pas des autres donc pour faire des prédictions les modèles ont besoins de toutes les variables à disposition. Les modèles suivants seront donc implémentés sans PCA préalable. En revanche, vu que les données ne sont pas à la même échelle, nous utiliserons un StandardScaler du package scikit-learn afin de normaliser les données.

6.1 Modèles

Pour simplifier les études suivantes, nous implémenterons les modèles en utilisant le système de Pipeline de scikit-learn. Caque modèle sera donc défini par une pipeline comprenant un StandardScaler puis le modèle en question avec ses paramètres par défaut. Ces modèles seront ensuite stockés dans un dictionnaire, et seul les plus performants seront sélectionnés pour être optimisés. On se basera pour cela sur la métrique d'accuracy du package scikit-learn.

Nous implémenterons les différents modèles de classification multi-classes disponibles dans le package scikit-leanr, soit : - RandomForest ; - LogisticRegression ; - SVC ; - Gradient Bossting ; - XGBoost Classifier.

[22]:
```
#Import libraries
from sklearn.metrics import accuracy_score

from sklearn import (
    datasets,
    decomposition,
```

```
    linear_model,
    metrics,
    model_selection,
    naive_bayes,
    pipeline,
)
```

[23]:
```
###Logistic Regression Pipeline

from sklearn.linear_model import LogisticRegression

pipeLogisticRegression = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', LogisticRegression())
])
```

[24]:
```
###RandomForest Pipeline

from sklearn.ensemble import RandomForestClassifier

pipeRandomForest = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', RandomForestClassifier())
])
```

[25]:
```
###SVC Pipeline

from sklearn.svm import SVC

pipeSVC = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', SVC())
])
```

[26]:
```
###Gradient Boosting Classifier Pipeline

from sklearn.ensemble import GradientBoostingClassifier

pipeGradientBoosting = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', GradientBoostingClassifier())
])
```

[27]:
```
###XGBoost Classifier Pipeline

import xgboost as xgb
from category_encoders.target_encoder import TargetEncoder
```

```
pipeXGBoost = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', xgb.XGBClassifier(objective = 'multi:softmax', enable_categorical=␣
 ↪True))
])
```

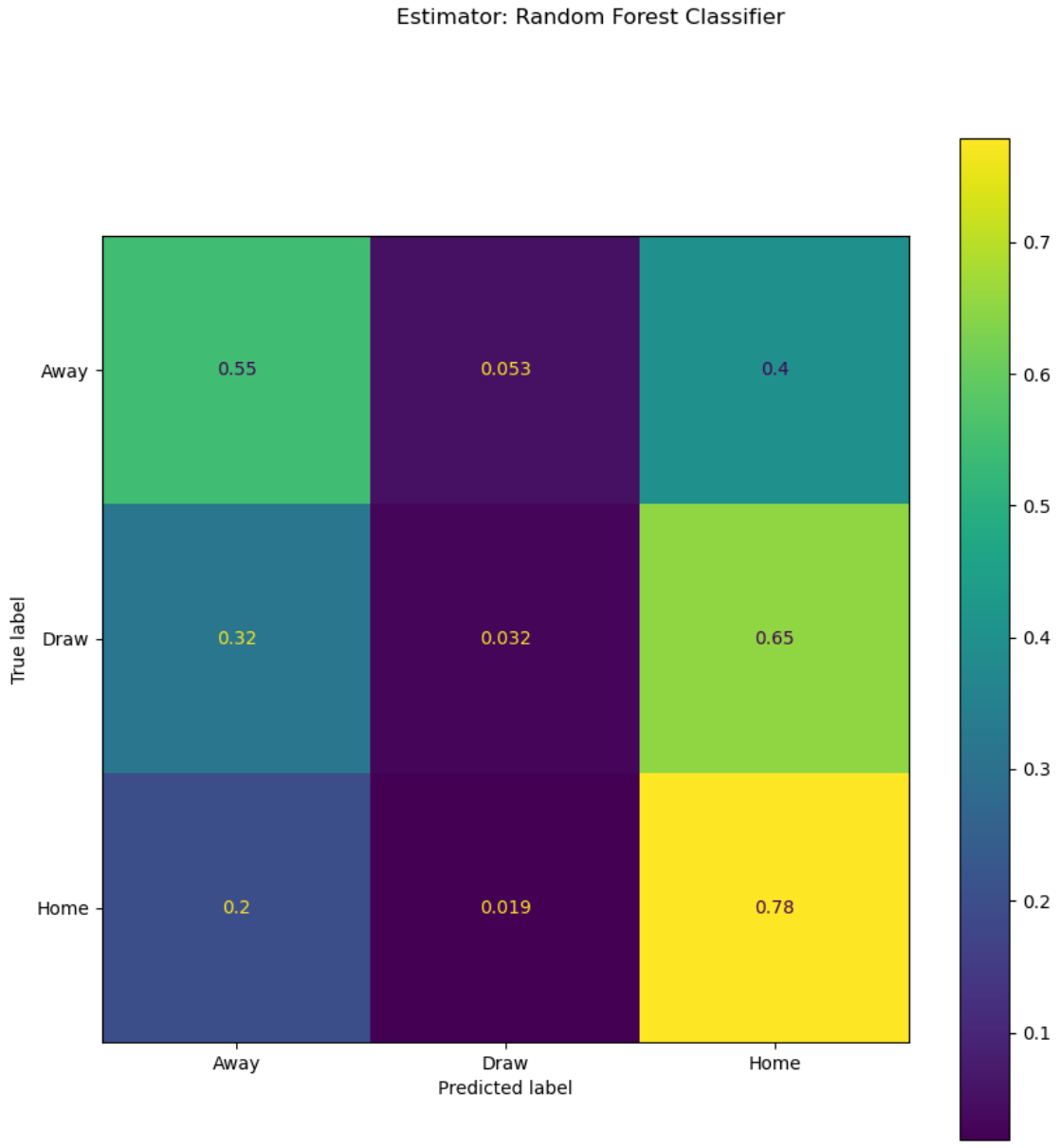
[28]:
```
models = {
    'Logistic Regression': pipeLogisticRegression,
    'Random Forest Classifier': pipeRandomForest,
    'SVC': pipeSVC,
    'Gradient Boosting Classifier': pipeGradientBoosting
}
```

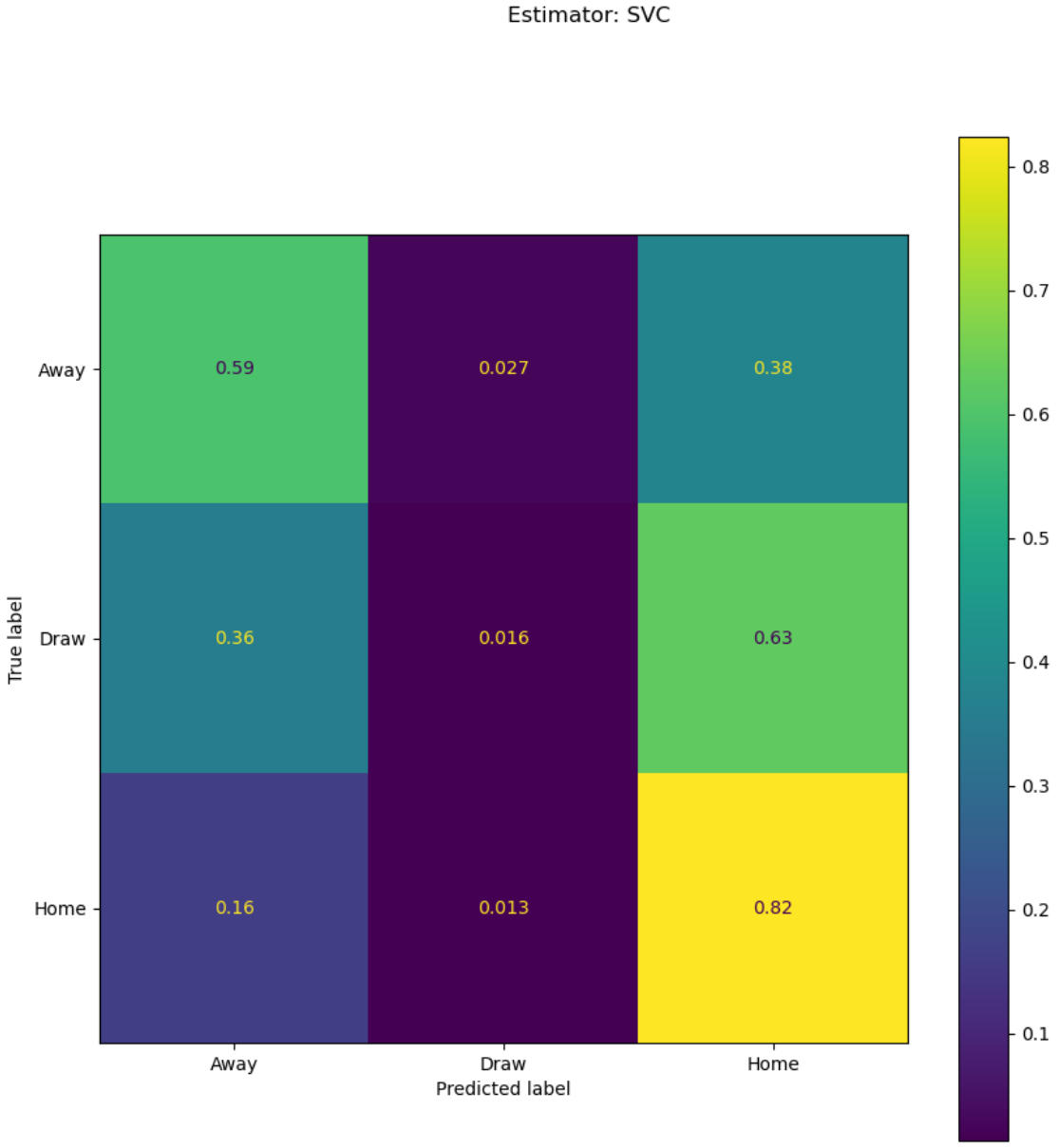
[29]:
```
models_scores = [] ###list of model names + score

for idx, (name,model) in enumerate(models.items()):
    ###fitting model on the training set and computing the accuracy score
    model.fit(xTrain, yTrain)
    y_pred = model.predict(xTest)
    score = metrics.accuracy_score(y_pred, yTest)

    models_scores.append((name, score))

    sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
```

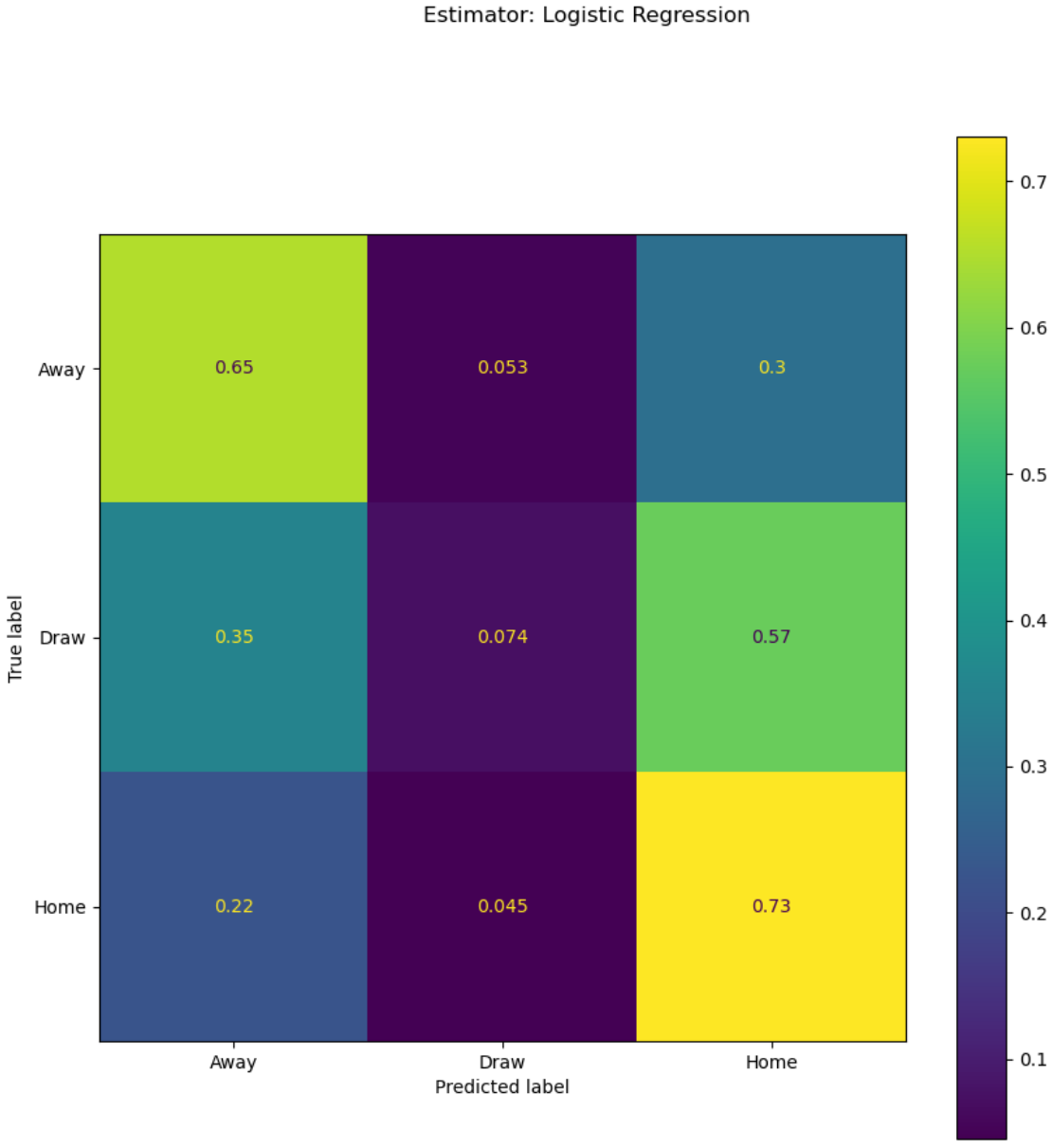
[30]:
```
for rank, (model_names, score) in enumerate(sorted_models, start=1):
    print('\nRank #%s :' % rank)
    print('Estimator: %s' % model_names)
    print('Test set accuracy score for default params: %.3f ' % score)
```

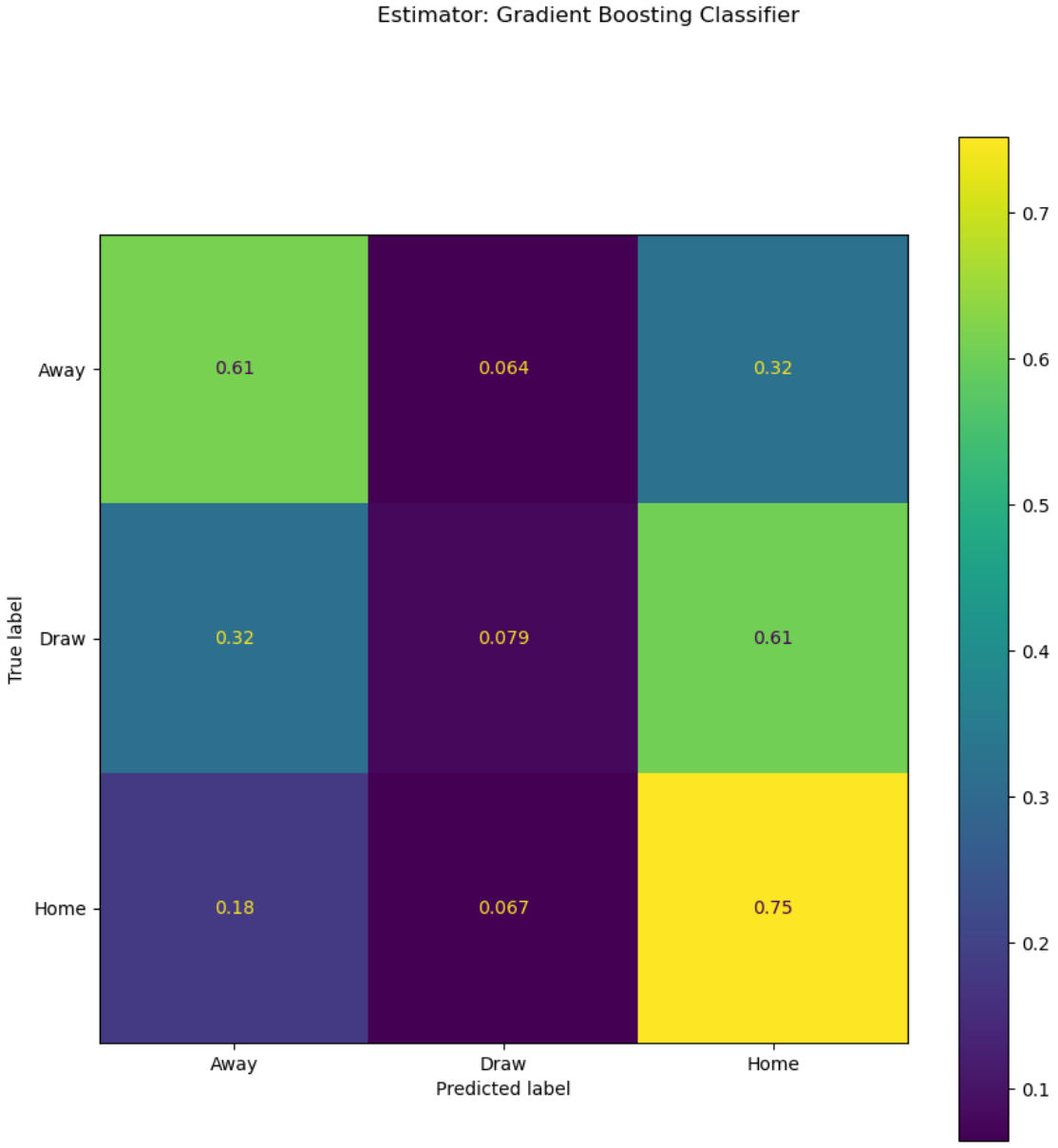
```
Rank #1 :
Estimator: SVC
Test set accuracy score for default params: 0.566

Rank #2 :
Estimator: Logistic Regression
Test set accuracy score for default params: 0.555

Rank #3 :
Estimator: Gradient Boosting Classifier
Test set accuracy score for default params: 0.552

Rank #4 :
Estimator: Random Forest Classifier
Test set accuracy score for default params: 0.539
```

[ ]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay

for idx, (name,model) in enumerate(models.items()):
    fig,ax = plt.subplots(figsize=(10,10))
    yPred = model.predict(xTest)
    fig.suptitle('Estimator: %s' % name)
    ConfusionMatrixDisplay.from_predictions(yTest, yPred, ax=ax, normalize=␣
 ↪'true')
```




Estimator: SVC
0.59
0.027
0.38
0.36
0.016
0.63
0.16
0.013
0.82
Away
Draw
Home
True label
Away
Draw
Home
Predicted label
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Estimator: Logistic Regression
0.65
0.053
0.3
0.35
0.074
0.57
0.22
0.045
0.73
Away
Draw
Home
True label
Away
Draw
Home
Predicted label
0.7
0.6
0.5
0.4
0.3
0.2
0.1



6.2 Métriques et résultats

CrossEntropyLoss

7 Trucs à faire

`Boucle sur Compétitions pour créer dataframe plus gros`

7.1 Créer csv avec dataframe

Description des variables :

Visualiaztion : Choix + Explication des Boxplots, Correlation matrix,

Modèles : XGBoost/RandomForest/LogisticRegression

Métriques : accuracy_score, balanced_accuracy

TUning : Gridsearch/RandomizedSearch sur params

[ ]:
```
matchStats['target'].value_counts()
```

8 test

[31]:
```
fullData.shape
```

[31]:
```
(3316, 59)
```

[58]:
```
fullDataNoDraw = fullData[fullData.Result != 'Draw']
```

[59]:
```
xTrainNoDraw, xTestNoDraw, yTrainNoDraw, yTestNoDraw =␣
 ↪train_test_split(fullDataNoDraw.iloc[:,:-1], fullDataNoDraw['Result'],␣
 ↪stratify= fullDataNoDraw['Result'], random_state=42)
```

[60]:
```
###RandomForest Pipeline

from sklearn.ensemble import RandomForestClassifier

pipeRandomForestNoDraw = pipeline.Pipeline([
    ('scl', StandardScaler()),
    ('clf', RandomForestClassifier())
])
```

[69]:
```
pipeGradientBoosting.fit(xTrainNoDraw, yTrainNoDraw)
```

[69]:
```
Pipeline(steps=[('scl', StandardScaler()),
                ('clf', GradientBoostingClassifier())])
```

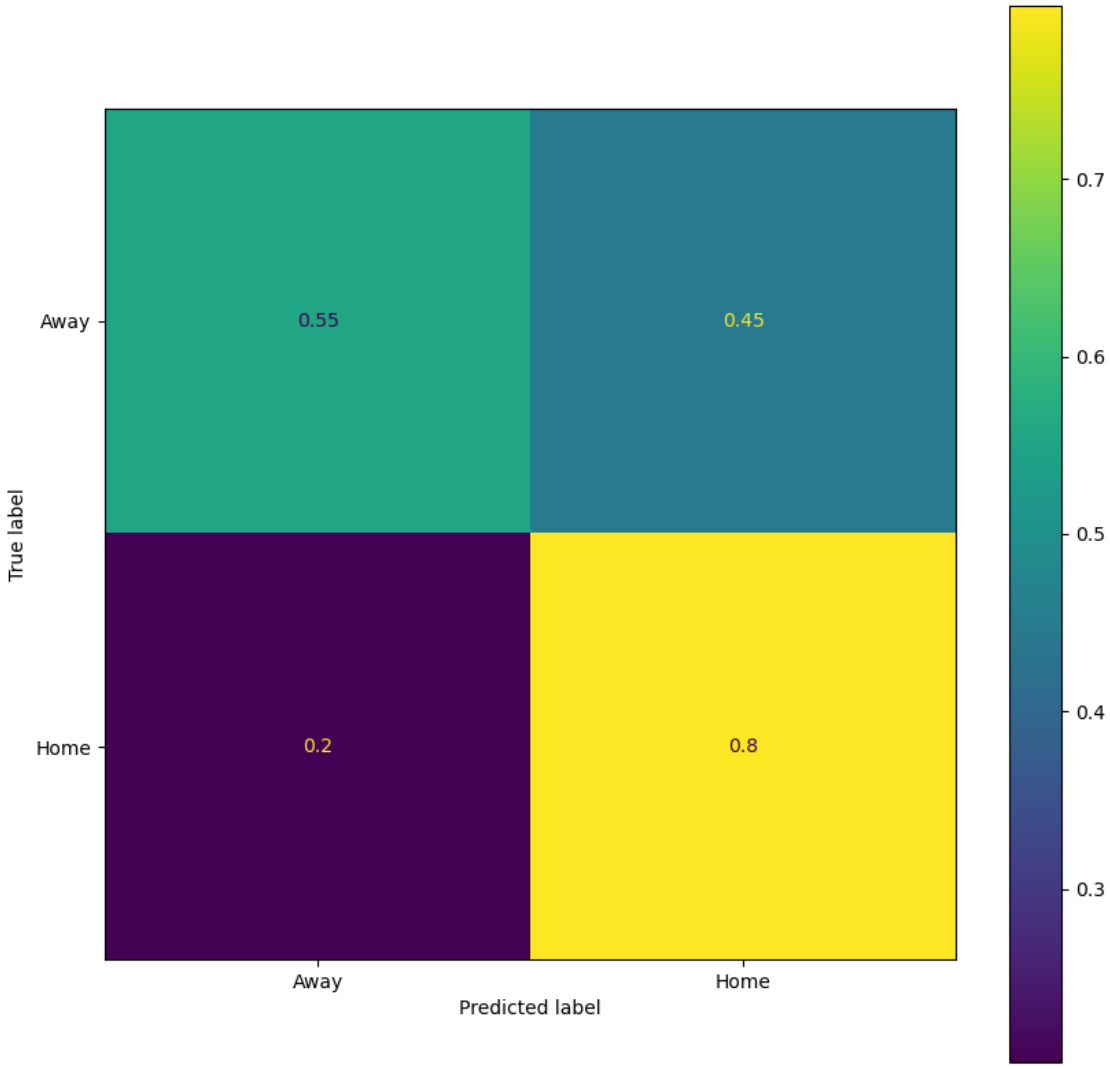
[70]:
```
yPredNoDraw = pipeRandomForestNoDraw.predict(xTestNoDraw)
score = metrics.accuracy_score(yPredNoDraw, yTestNoDraw)
```

[71]:
```
score
```

[71]:
```
0.6964006259780907
```

[72]:
```
from sklearn.metrics import ConfusionMatrixDisplay

fig,ax = plt.subplots(figsize=(10,10))
ConfusionMatrixDisplay.from_predictions(yTestNoDraw, yPredNoDraw, ax=ax,␣
 ↪normalize= 'true')
```

[72]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fa651c03c10>



[ ]: