main

May 8, 2024

[1]:
```
import os
import pathlib
import pandas as pd
import numpy as np
```

```
/Users/catherine_gai/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/computation/expressions.py:20: UserWarning: Pandas requires
version '2.7.3' or newer of 'numexpr' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

[2]:
```
import sklearn.linear_model as lm
import sklearn.preprocessing as preprocess
import sklearn.neural_network as nn
import sklearn.ensemble as ensemble
```

[3]:
```
import warnings
warnings.filterwarnings('ignore')
```

[4]:
```
data_folder = pathlib.Path("E0")
```

[5]:
```
train_files = ["E0_13.csv", "E0_14.csv", "E0_15.csv", "E0_16.csv", "E0_17.csv",␣
↪"E0_18.csv"]
val_files = ["E0_19.csv", "E0_20.csv", "E0_21.csv"]
test_files =["E0_22.csv", "E0_23.csv"]
```

[6]:
```
train_dfs = [pd.read_csv(data_folder / f) for f in train_files]
```

[7]:
```
val_dfs = [pd.read_csv(data_folder / f) for f in val_files]
```

[8]:
```
test_dfs = [pd.read_csv(data_folder / f) for f in test_files]
```

[9]:
```
print([df.shape for df in train_dfs])
print([df.shape for df in val_dfs])
print([df.shape for df in test_dfs])
```

```
[(380, 74), (380, 68), (381, 68), (380, 65), (380, 65), (380, 65)]
[(380, 62), (380, 106), (380, 106)]
[(380, 106), (380, 106)]
```

[10]:
```
train_df = pd.concat(train_dfs, join="inner")
train_df.shape
```

[10]:
```
(2281, 65)
```

[11]:
```
val_df = pd.concat(val_dfs, join="inner")
val_df.shape
```

[11]:
```
(1140, 44)
```

[12]:
```
test_df = pd.concat(test_dfs, join="inner")
test_df.shape
```

[12]:
```
(760, 106)
```

[13]:
```
"""
HS = Home Team Shots
AS = Away Team Shots
HST = Home Team Shots on Target
AST = Away Team Shots on Target
HHW = Home Team Hit Woodwork
AHW = Away Team Hit Woodwork
HC = Home Team Corners
AC = Away Team Corners
HF = Home Team Fouls Committed
AF = Away Team Fouls Committed
HFKC = Home Team Free Kicks Conceded
AFKC = Away Team Free Kicks Conceded
HO = Home Team Offsides
AO = Away Team Offsides
HY = Home Team Yellow Cards
AY = Away Team Yellow Cards
HR = Home Team Red Cards
AR = Away Team Red Cards
HBP = Home Team Bookings Points (10 = yellow, 25 = red)
ABP = Away Team Bookings Points (10 = yellow, 25 = red)
"""
```

[13]:
```
'\nHS = Home Team Shots\nAS = Away Team Shots\nHST = Home Team Shots on
Target\nAST = Away Team Shots on Target\nHHW = Home Team Hit Woodwork\nAHW =
Away Team Hit Woodwork\nHC = Home Team Corners\nAC = Away Team Corners\nHF =
Home Team Fouls Committed\nAF = Away Team Fouls Committed\nHFKC = Home Team Free
Kicks Conceded\nAFKC = Away Team Free Kicks Conceded\nHO = Home Team
Offsides\nAO = Away Team Offsides\nHY = Home Team Yellow Cards\nAY = Away Team
Yellow Cards\nHR = Home Team Red Cards\nAR = Away Team Red Cards\nHBP = Home
Team Bookings Points (10 = yellow, 25 = red)\nABP = Away Team Bookings Points
(10 = yellow, 25 = red)\n'
```

[14]:
```
team_cols = ["HomeTeam", "AwayTeam"]
x_cols = ["HS" ,"AS", "HST", "AST", "HC", "AC", "HF", "AF", "HY", "AY", "HR",␣
↪"AR"]
y_col = ["FTR"]
```

[15]:
```
train_df = train_df[team_cols + x_cols + y_col].dropna()
val_df = val_df[team_cols + x_cols + y_col].dropna()
test_df = test_df[team_cols + x_cols + y_col].dropna()
```

[16]:
```
def drop_draw(df):
    return df[(df["FTR"] != "D") & (df["FTR"] != 2)]
```

[17]:
```
def remap(x):
    if x == "D":
        return 2
    elif x == "H":
        return 0
    elif x == "A":
        return 1
    else:
        raise ValuError
```

[18]:
```
# len(drop_draw(train_df))
```

[19]:
```
X_train = train_df[x_cols]
Y_train = train_df[y_col]
```

[20]:
```
X_train_normalized = preprocess.normalize(X_train, axis=0)
Y_train = Y_train["FTR"].apply(remap)
```

[21]:
```
lr_model = lm.LogisticRegression(random_state=115, multi_class="ovr")
lr_model.fit(X_train_normalized, Y_train)
```

[21]:
```
LogisticRegression(multi_class='ovr', random_state=115)
```

[22]:
```
lr_model.score(X_train_normalized, Y_train)
```

[22]:
```
0.4605263157894737
```

[23]:
```
mlp = nn.MLPClassifier(random_state=115, hidden_layer_sizes=[32, 64, 128, 64,␣
↪32], max_iter=5000, learning_rate_init=0.003)
mlp.fit(X_train_normalized, Y_train)
```

[23]:
```
MLPClassifier(hidden_layer_sizes=[32, 64, 128, 64, 32],
              learning_rate_init=0.003, max_iter=5000, random_state=115)
```

[24]:
```
mlp.score(X_train_normalized, Y_train)
```

[24]:
```
0.5828947368421052
```

[25]:
```
adaboost = ensemble.AdaBoostClassifier()
adaboost.fit(X_train, Y_train)
```

[25]:
```
AdaBoostClassifier()
```

[26]:
```
adaboost.score(X_train, Y_train)
```

[26]:
```
0.5894736842105263
```

[27]:
```
X_val = val_df[x_cols]
Y_val = val_df[y_col]
X_val_normalized = preprocess.normalize(X_val, axis=0)
Y_val = Y_val["FTR"].apply(remap)
```

[28]:
```
mlp.score(X_val_normalized, Y_val)
```

[28]:
```
0.6131578947368421
```

0.0.1 Start of Prediction using Historical Record

[29]:
```
x_cols_home = [c for c in x_cols if c.startswith("H")]
x_cols_away = [c for c in x_cols if c.startswith("A")]
```

[30]:
```
def get_record(history_df, team):
    filtered_history = np.concatenate([history_df[history_df["HomeTeam"] ==␣
 ↪team][x_cols_home].to_numpy(),
                                       history_df[history_df["AwayTeam"] ==␣
 ↪team][x_cols_away].to_numpy()])
    return filtered_history.mean(axis=0)
```

[31]:
```
records = []
for i in range(len(val_df)):
    record = {}
    record.update(dict(zip(x_cols_home, get_record(train_df, val_df["HomeTeam"].
 ↪to_numpy()[i]))))
    record.update(dict(zip(x_cols_away, get_record(train_df, val_df["AwayTeam"].
 ↪to_numpy()[i]))))
    record.update({"HomeTeam": val_df["HomeTeam"].to_numpy()[i], "AwayTeam":␣
 ↪val_df["AwayTeam"].to_numpy()[i]})
    record.update({"FTR": val_df["FTR"].to_numpy()[i]})
    records.append(record)
val_df_new = pd.DataFrame(records, columns=team_cols + x_cols + y_col)
```

[32]:
```
val_df_new = val_df_new.dropna()
```

[33]:
```
X_val = val_df_new[x_cols]
Y_val = val_df_new[y_col]
X_val_normalized = preprocess.normalize(X_val, axis=0)
Y_val = Y_val["FTR"].apply(remap)
```

[34]:
```
mlp.score(X_val_normalized, Y_val)
```

[34]: 0.5184782608695652

[35]:
```
train_val_df = pd.concat([train_df, val_df])
print(len(train_val_df.columns))
records = []
for i in range(len(val_df)):
    record = {}
    record.update(dict(zip(x_cols_home, get_record(train_val_df,␣
 ↪val_df["HomeTeam"].to_numpy()[i]))))
    record.update(dict(zip(x_cols_away, get_record(train_val_df,␣
 ↪val_df["AwayTeam"].to_numpy()[i]))))
    record.update({"HomeTeam": val_df["HomeTeam"].to_numpy()[i], "AwayTeam":␣
 ↪val_df["AwayTeam"].to_numpy()[i]})
    record.update({"FTR": val_df["FTR"].to_numpy()[i]})
    records.append(record)
test_df_new = pd.DataFrame(records, columns=team_cols + x_cols + y_col)
```

15

[36]:
```
X_test = test_df_new[x_cols]
Y_test = test_df_new[y_col]
X_test_normalized = preprocess.normalize(X_test, axis=0)
Y_test = Y_test["FTR"].apply(remap)
```

[37]:
```
mlp.score(X_test_normalized, Y_test)
```

[37]: 0.5201754385964912

[ ]:
```

```