HW2_1

February 17, 2021

```
[1]: import pandas as pd
import numpy as np
from scipy.special import logit, expit
import statsmodels.formula.api as smf
from sklearn.metrics import brier_score_loss
import warnings
warnings.filterwarnings('ignore')
rng = np.random.default_rng(seed = 456)
```

1 1a. For each game compute the historical average goal differentials for each team

```
[2]: df = pd.read_csv('soccer18.csv', parse_dates = ['Date'])
df = df.replace('Evian Thonon Gaillard', 'Evian')
df['GameID'] = df.index
df['PD_H'] = df.FTHG - df.FTAG
df['PD_A'] = df.FTAG - df.FTHG
df = df.sort_values('Date')
```

```
[3]: df['homeWin'] = 1*(df.FTHG > df.FTAG)
```

```
[4]: df_melt = pd.melt(df, id_vars='GameID', value_vars=['HomeTeam', 'AwayTeam'],␣
↪var_name='isHome', value_name='Team')
df_melt['isHome'] = np.where(df_melt.isHome =='HomeTeam', 'H', 'A')
```

```
[5]: df_melt2 = pd.melt(df, id_vars='GameID', value_vars=['PD_H', 'PD_A'],␣
↪var_name='isHome', value_name='PD')
df_melt2['isHome'] = np.where(df_melt2.isHome =='PD_H', 'H', 'A')
```

```
[6]: df_merge = df_melt.merge(df_melt2, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date']], on='GameID').sort_values('Date')
df_merge['hAGD'] = df_merge.groupby('Team').PD.transform(lambda x : x.
↪expanding().mean().shift(1, fill_value = 0))
df_merge['GP'] = df_merge.groupby('Team').PD.transform(lambda x : x.expanding().
↪count().shift(1, fill_value = 0))
```

[7]:
```python
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot['goalDisp'] = np.abs(df_pivot.hAGD_H - df_pivot.hAGD_A)
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].merge(
    df_pivot[['GameID', 'hAGD_H', 'hAGD_A','GP_H', 'GP_A', 'goalDisp']],␣
↪on='GameID')
```

[8]:
```python
train = df_pivot[df_pivot.Y < 18]
test = df_pivot[df_pivot.Y == 18]
```

1.1 i. Give a table containing the 7 games with the largest absolute disparity

[9]:
```python
train.sort_values('goalDisp', ascending=False).head(7).drop('GameID', 1).
↪reset_index(drop=True)
```

[9]:

	Div	Y	HomeTeam	AwayTeam	hAGD_H	hAGD_A	GP_H	GP_A	goalDisp
0	Serie_A	14	Sassuolo	Sampdoria	-3.5	1.000000	2	2	4.500000
1	Ligue_1	14	Evian	Paris SG	-3.5	1.000000	2	2	4.500000
2	Ligue_1	17	Strasbourg	Lille	-4.0	0.078261	1	115	4.078261
3	Serie_A	14	Palermo	Inter	-0.5	3.500000	2	2	4.000000
4	La_Liga	14	Cordoba	Celta	-2.0	2.000000	1	1	4.000000
5	Serie_A	14	Empoli	Roma	-2.0	2.000000	1	1	4.000000
6	La_Liga	14	Elche	Granada	-3.0	1.000000	1	1	4.000000

1.2 ii. Repeat the previous part restricted to games where each team had previously played at least 100 games in our dataset (that is, 100 or more)

[10]:
```python
train.loc[(train.GP_H >= 100) & (train.GP_A >= 100)].sort_values('goalDisp',␣
↪ascending=False).head(7).drop('GameID', 1).reset_index(drop=True)
```

[10]:

	Div	Y	HomeTeam	AwayTeam	hAGD_H	hAGD_A	GP_H	GP_A	\
0	La_Liga	16	Granada	Barcelona	-0.875000	2.192308	104	104	
1	La_Liga	17	Levante	Barcelona	-0.705357	2.140000	112	150	
2	La_Liga	16	Granada	Real Madrid	-0.936937	1.900000	111	110	
3	La_Liga	17	Las Palmas	Barcelona	-0.623762	2.208633	101	139	
4	La_Liga	17	La Coruna	Barcelona	-0.621622	2.142857	148	147	
5	La_Liga	16	La Coruna	Barcelona	-0.519608	2.225490	102	102	
6	La_Liga	17	Barcelona	La Coruna	2.186047	-0.527132	129	129	

	goalDisp
0	3.067308
1	2.845357
2	2.836937

```
3   2.832395
4   2.764479
5   2.745098
6   2.713178
```

1.3 iii. Almost all games in the solution to part (i) come from the 2014 season (the first season in our dataset), but one comes from the 2017 season. In a few words, explain what is special about it

For the game in 2017, Strasbourg played it's first game since being relegated in the 2007-08 season. Since the data we have starts in 2014, it is considered their first game in the data set, thus why the average point differential is so high.

2 1b. Fit a logit model to predict the probability of the home team winning (draws count as non-wins) using only an intercept term.

[11]:
```python
df = pd.merge(df, df_pivot[['GameID', 'hAGD_H', 'hAGD_A']], how='left')
train = df[df.Y < 18]
test = df[df.Y == 18]
```

[12]:
```python
result = smf.logit('homeWin ~ 1', data = train).fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.689679
         Iterations 3
```

[12]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                homeWin   No. Observations:                 7304
Model:                          Logit   Df Residuals:                     7303
Method:                           MLE   Df Model:                            0
Date:                Wed, 17 Feb 2021   Pseudo R-squ.:               4.042e-12
Time:                        18:42:47   Log-Likelihood:                -5037.4
converged:                       True   LL-Null:                       -5037.4
Covariance Type:            nonrobust   LLR p-value:                       nan
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.1669      0.023     -7.106      0.000      -0.213      -0.121
==============================================================================
"""
```

2.1 i. Report your coefficient value.

[13]:
```
print('The coefficient is:', result.params.values[0])
```

```
The coefficient is: -0.16687026113323677
```

2.2 ii. Report the Brier score of your out-of-sample predictions on 2018 (Y=18)

[14]:
```
y_pred = result.predict(test)
print('Brier Score:', brier_score_loss(test['homeWin'], y_pred))
```

```
Brier Score: 0.2473559477379797
```

3 1c. The intercept coefficient from the previous part is negative. Does this imply there is no home field advantage? In other words, if home teams are favored, shouldn't the intercept be positive?

The intercept and the logit model as a whole cannot be used to interpret probability. To do that we have to implement the expit function in order to translate the probabilities and whether or not there is home field advantage. Since the expit function exponentiates the values given to create a probability the resulting value will always be between 0 and 1.

4 1d. Repeat part (b) using the intercept, and the historical average goal differentials from each team as features (three features in total)

[15]:
```
result = smf.logit('homeWin ~ hAGD_H + hAGD_A + 1', data = train).fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.630677
         Iterations 5
```

[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                homeWin   No. Observations:                 7304
Model:                          Logit   Df Residuals:                     7301
Method:                           MLE   Df Model:                            2
```

```
Date:                Wed, 17 Feb 2021   Pseudo R-squ.:                 0.08555
Time:                        18:42:47   Log-Likelihood:                -4606.5
converged:                       True   LL-Null:                       -5037.4
Covariance Type:            nonrobust   LLR p-value:                6.933e-188
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.1791      0.025     -7.183      0.000      -0.228      -0.130
hAGD_H         0.7853      0.039     20.128      0.000       0.709       0.862
hAGD_A        -0.7619      0.040    -19.082      0.000      -0.840      -0.684
==============================================================================
"""
```

[16]:

```python
print('The coefficients are:', result.params.values)
```

```
The coefficients are: [-0.17910355  0.78534468 -0.76193982]
```

[17]:

```python
y_pred = result.predict(test)
print('Brier Score:', brier_score_loss(test['homeWin'], y_pred))
```

```
Brier Score: 0.21726101075298782
```

HW2_2

February 17, 2021

[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import statsmodels.formula.api as smf
import statsmodels.api as sm
from scipy.special import logit, expit
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import brier_score_loss

import warnings
warnings.filterwarnings('ignore')
rng = np.random.default_rng(seed = 456)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[2]:

```
def forward_selection(data, target, significance_level=0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.
 ↪add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<significance_level):
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features

def backward_elimination(data, target,significance_level = 0.05):
```

```
    features = data.columns.tolist()
    while(len(features)>0):
        features_with_constant = sm.add_constant(data[features])
        p_values = sm.OLS(target, features_with_constant).fit().pvalues[1:]
        max_p_value = p_values.max()
        if(max_p_value >= significance_level):
            excluded_feature = p_values.idxmax()
            features.remove(excluded_feature)
        else:
            break
    return features
```

[3]:
```
def to_df(X, y):
  df = X.copy()
  df['homeWin'] = y
  return df

def logit(X_train, y_train, X_test, y_test):
  f = y_train.name + ' ~ ' + ' + '.join([col for col in X_train.columns])
  result = smf.logit(f, data=to_df(X_train, y_train)).fit()
  y_pred = result.predict(X_test)
  return brier_score_loss(y_test, y_pred)

def randomForest(X_train, y_train, X_test, y_test):
  rf = RandomForestClassifier()
  rf.fit(X_train, y_train)
  y_pred = rf.predict_proba(X_test)[:, 1]
  return brier_score_loss(y_test, y_pred)
```

[4]:
```
df = pd.read_csv('soccer18.csv', parse_dates = ['Date'])
df = df.replace('Evian Thonon Gaillard', 'Evian')
df['GameID'] = df.index
df['PD_H'] = df.FTHG - df.FTAG
df['PD_A'] = df.FTAG - df.FTHG
df = df.sort_values('Date')
```

We can add a few metrics based off data we have to add dimensionality to the dataset.

From Q1 * h/a historical goal differential * absolute disparity

[5]:
```
df_melt = pd.melt(df, id_vars='GameID', value_vars=['HomeTeam', 'AwayTeam'],␣
 ↪var_name='isHome', value_name='Team')
df_melt['isHome'] = np.where(df_melt.isHome =='HomeTeam', 'H', 'A')
df_melt2 = pd.melt(df, id_vars='GameID', value_vars=['PD_H', 'PD_A'],␣
 ↪var_name='isHome', value_name='PD')
df_melt2['isHome'] = np.where(df_melt2.isHome =='PD_H', 'H', 'A')
```

```
df_merge = df_melt.merge(df_melt2, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date']], on='GameID').sort_values('Date')
df_merge['hAGD'] = df_merge.groupby(['Team']).PD.transform(lambda x : x.
↪expanding().mean().shift(1, fill_value = 0))
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].merge(
    df_pivot[['GameID', 'hAGD_H', 'hAGD_A']], on='GameID')
df = pd.merge(df, df_pivot[['GameID', 'hAGD_H', 'hAGD_A']], how='left')
df['homeWin'] = 1*(df.FTHG > df.FTAG)
```

[6]:
```
train = df[df.Y < 18]
test = df[df.Y == 18]
```

Using Question 1d Intercept + historical goal differential

[7]:
```
result = smf.logit('homeWin ~ 1+hAGD_H+hAGD_A', data=train).fit()
y_pred = result.predict(test)
brier_score_loss(test['homeWin'], y_pred)
```

```
Optimization terminated successfully.
         Current function value: 0.630677
         Iterations 5
```

[7]: 0.21726101075298782

Here we have the baseline model with a brier score of **0.21727**

[8]:
```
X_train, y_train = df[df.Y < 18][['hAGD_H', 'hAGD_A']], df[df.Y < 18].homeWin
X_test, y_test = df[df.Y == 18][['hAGD_H', 'hAGD_A']], df[df.Y == 18].homeWin
```

[9]:
```
bsl = randomForest(X_train, y_train, X_test, y_test)
print('Random forest brier score:', bsl)
```

```
Random forest brier score: 0.26095649642657215
```

Now we have the baseline model, we can add more features with the other available columns of data (HS/AS, HST/AST, home_xG/away_xG). Since these stats are only calculated during/after the game, we can convert this data into historical features like we did the goal differentials.

[10]:
```
df['STD_H'] = df.HST - df.AST
df['STD_A'] = df.AST - df.HST
df['xGD_H'] = df.home_xG - df.away_xG
df['xGD_A'] = df.away_xG - df.home_xG
```

[11]:
```
df_melt3 = pd.melt(df, id_vars='GameID', value_vars=['xGD_H', 'xGD_A'],␣
↪var_name='isHome', value_name='xGD')
```

```
df_melt3['isHome'] = np.where(df_melt3.isHome =='xGD_H', 'H', 'A')
df_merge = df_melt.merge(df_melt3, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date', 'Y']], on='GameID').sort_values('Date')
df_merge['hxGD'] = df_merge.groupby(['Team', 'Y']).xGD.transform(lambda x : x.
↪expanding().mean().shift(1, fill_value = 0))
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].
↪merge(df_pivot[['GameID', 'hxGD_H', 'hxGD_A']], on='GameID')
df = pd.merge(df, df_pivot[['GameID', 'hxGD_H', 'hxGD_A']], how='left')
```

[12]:
```
df_melt4 = pd.melt(df, id_vars='GameID', value_vars=['STD_H', 'STD_A'],␣
↪var_name='isHome', value_name='STD')
df_melt4['isHome'] = np.where(df_melt4.isHome =='STD_H', 'H', 'A')
df_merge = df_melt.merge(df_melt4, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date', 'Y']], on='GameID').sort_values('Date')
df_merge['hST'] = df_merge.groupby(['Team', 'Y']).STD.transform(lambda x : x.
↪expanding().mean().shift(1, fill_value = 0))
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].
↪merge(df_pivot[['GameID', 'hST_H', 'hST_A']], on='GameID')
df = pd.merge(df, df_pivot[['GameID', 'hST_H', 'hST_A']], how='left')
```

[13]:
```
# Drop all game and post game stats
pgs = ['FTHG', 'FTAG', 'HTHG', 'HTAG', 'HS', 'AS', 'HST', 'AST', 'home_xG',␣
↪'away_xG', 'PD_H', 'PD_A', 'STD_H', 'STD_A', 'xGD_H', 'xGD_A']
df_hs = df.drop(pgs, 1)
```

[14]:
```
# Drop GameID and Y after train/test splits
X_train, y_train = df_hs[df.Y < 18].drop('homeWin', 1), df_hs[df.Y < 18].homeWin
X_test, y_test = df_hs[df.Y == 18].drop('homeWin', 1), df_hs[df.Y == 18].homeWin
X_train = X_train.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
X_test = X_test.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
```

[15]:
```
randomForest(X_train, y_train, X_test, y_test)
```

[15]:
```
0.22452999892117875
```

[16]:
```
logit(X_train, y_train, X_test, y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.623677
         Iterations 5
```

[16]: 0.21519929870245788

Let's add a few more metrics using available data * Home Shot Quantity = FTHG / HS * Home Shot Quality = FTHG / HST * Away Shot Quantity = FTAG / AS * Away Shot Quality = FTAG / AST * Home Win Percentage = $\frac{\sum I \cdot \text{homeWin}}{totalgames}$

Because these are fractions, there will be NaN values. If there are zero total shots and/or shots on target but a goal was scored for home/away respectively, we can interpret that as self goals. Therefore the shot quantity/quality percentage can be 0%

[17]:
```
# Create Shot Quantity and Shot Quality columns for Home/Away
df['shotQntD_H'] = df.FTHG / df.HS
df['shotQntD_A'] = df.FTAG / df.AS
df['shotQltD_H'] = df.FTHG / df.HST
df['shotQltD_A'] = df.FTAG / df.AST

# Fill the percentages with 0 if Shots / Shots on Target are equal to zero
df = df.fillna(0)
```

[18]:
```
# Replace the columns values with the difference of home/away
df['shotQntD_H'] = df.shotQntD_H - df.shotQntD_A
df['shotQntD_A'] = df.shotQntD_A - df.shotQntD_H
df['shotQltD_H'] = df.shotQltD_H - df.shotQltD_A
df['shotQltD_A'] = df.shotQltD_A - df.shotQltD_H
```

[19]:
```
df_melt5 = pd.melt(df, id_vars='GameID', value_vars=['shotQntD_H',␣
↪'shotQntD_A'], var_name='isHome', value_name='sQntD')
df_melt5['isHome'] = np.where(df_melt5.isHome =='shotQntD_H', 'H', 'A')
df_merge = df_melt.merge(df_melt5, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date', 'Y']], on='GameID').sort_values('Date')
df_merge['hsQntD'] = df_merge.groupby(['Team', 'Y']).sQntD.transform(lambda x :␣
↪x.expanding().mean().shift(1, fill_value = 0))
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].
↪merge(df_pivot[['GameID', 'hsQntD_H', 'hsQntD_A']], on='GameID')
df = pd.merge(df, df_pivot[['GameID', 'hsQntD_H', 'hsQntD_A']], how='left')
```

[20]:
```
df_melt6 = pd.melt(df, id_vars='GameID', value_vars=['shotQltD_H',␣
↪'shotQltD_A'], var_name='isHome', value_name='sQltD')
df_melt6['isHome'] = np.where(df_melt6.isHome =='shotQltD_H', 'H', 'A')
df_merge = df_melt.merge(df_melt6, on=['GameID', 'isHome']).merge(df[['GameID',␣
↪'Date', 'Y']], on='GameID').sort_values('Date')
df_merge['hsQltD'] = df_merge.groupby(['Team', 'Y']).sQltD.transform(lambda x :␣
↪x.expanding().mean().shift(1, fill_value = 0))
df_pivot = df_merge.pivot(index='GameID', columns='isHome')
```

```
df_pivot.columns = [f'{i}_{j}' for i, j in df_pivot.columns]
df_pivot = df_pivot.reset_index()
df_pivot = df[['GameID', 'Div', 'Y', 'HomeTeam', 'AwayTeam']].
 ↪merge(df_pivot[['GameID', 'hsQltD_H', 'hsQltD_A']], on='GameID')
df = pd.merge(df, df_pivot[['GameID', 'hsQltD_H', 'hsQltD_A']], how='left')
```

[21]:
```
df_hs = df_hs.merge(df[['GameID', 'hsQntD_H', 'hsQntD_A', 'hsQltD_H',␣
 ↪'hsQltD_A']], how='left')
# Create Home Win Percentage Column
hw = df_hs.groupby(['HomeTeam', 'Y']).homeWin.transform(lambda x : x.
 ↪expanding().sum().shift(1, fill_value = 0))
hg = df_hs.groupby(['HomeTeam', 'Y']).homeWin.transform(lambda x : x.
 ↪expanding().count().shift(1, fill_value = 0))

df_hs['home_win_pct'] = hw / hg
df_hs = df_hs.fillna(0)
df_hs
```

[21]:
```
          Div        Date   Y  …  hsQltD_H  hsQltD_A  home_win_pct
0     Ligue_1 2014-08-08  14  …  0.000000  0.000000      0.000000
1     Ligue_1 2014-08-09  14  …  0.000000  0.000000      0.000000
2     Ligue_1 2014-08-09  14  …  0.000000  0.000000      0.000000
3     Ligue_1 2014-08-09  14  …  0.000000  0.000000      0.000000
4     Ligue_1 2014-08-09  14  …  0.000000  0.000000      0.000000
…         …          …  ..  …       …         …            …
9125  Serie_A 2019-05-26  18  …  0.138975  0.093488      0.277778
9126  Serie_A 2019-05-26  18  …  0.223951  0.027068      0.555556
9127  Serie_A 2019-05-26  18  …  0.136268  0.202230      0.611111
9128  Serie_A 2019-05-26  18  …  0.199632  0.282999      0.500000
9129  Serie_A 2019-05-26  18  …  0.119181  0.082793      0.277778

[9130 rows x 18 columns]
```

[22]:
```
# Drop GameID and Y after train/test splits
X_train, y_train = df_hs[df.Y < 18].drop('homeWin', 1), df_hs[df.Y < 18].homeWin
X_test, y_test = df_hs[df.Y == 18].drop('homeWin', 1), df_hs[df.Y == 18].homeWin
X_train = X_train.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
X_test = X_test.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
```

[23]:
```
lr_bsl = logit(X_train, y_train, X_test, y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.623616
         Iterations 5
```

[24]:
```
rf_bsl = randomForest(X_train, y_train, X_test, y_test)
```

Even after all the operations we went through, the random forest model without hyperparameter tuning does not perform better than the logit. From here, I will create a validation set using Y=17 from the training data and perform a Grid Search to optimize the hyperparameters in order to boost performance.

[25]:
```
# Create validation splits along with training and testing
X_train, y_train = df_hs[df.Y < 17].drop('homeWin', 1), df_hs[df.Y < 17].homeWin
X_val, y_val = df_hs[df.Y == 17].drop('homeWin', 1), df_hs[df.Y == 17].homeWin
X_test, y_test = df_hs[df.Y == 18].drop('homeWin', 1), df_hs[df.Y == 18].homeWin

X_train = X_train.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
X_val = X_val.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
X_test = X_test.drop(['GameID', 'Y'], 1).select_dtypes(include=np.number)
```

[26]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth' : [1, 3, 5, 7]
}

rf_CV = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid,␣
 ↪scoring = 'neg_brier_score', cv=4)
rf_CV.fit(X_val, y_val)
rf_CV.best_params_
```

[26]:
```
{'max_depth': 5, 'n_estimators': 500}
```

[27]:
```
rf = RandomForestClassifier(max_depth=5, n_estimators=200)
rf.fit(X_train, y_train)
y_pred = rf.predict_proba(X_test)[:, 1]
rf_GS_bsl = brier_score_loss(y_test, y_pred)
```

[28]:
```
# Implementing forward and backward selection
fs = forward_selection(X_train, y_train)
bs = backward_elimination(X_train, y_train)
```

[29]:
```
lr_bsl_fs = logit(X_train[fs], y_train, X_test[fs], y_test)
rf_bsl_fs = randomForest(X_train[fs], y_train, X_test[fs], y_test)
lr_bsl_bs = logit(X_train[bs], y_train, X_test[bs], y_test)
rf_bsl_bs = randomForest(X_train[bs], y_train, X_test[bs], y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.628785
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.628785
         Iterations 5
```

[30]:
```
lr_bsl_fs, rf_bsl_fs
```

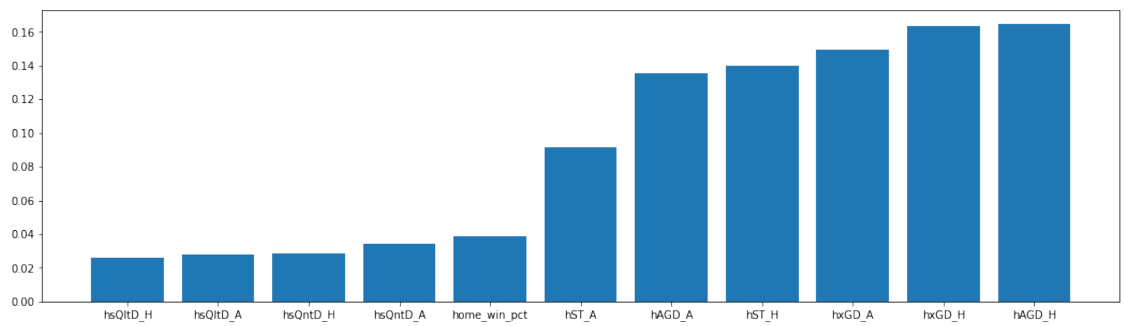
[30]: (0.21540507985187357, 0.2279123860912145)

[31]:
```
lr_bsl_bs, rf_bsl_bs
```

[31]: (0.21540507985187352, 0.2304679629872942)

[32]:
```
importances = pd.DataFrame(list(zip(X_train.columns, rf.feature_importances_)),␣
↪columns = ['attribute', 'importance'])
sorted_imp = importances.sort_values('importance')
plt.figure(figsize=(18,5))
plt.bar(sorted_imp['attribute'], sorted_imp['importance'])
```

[32]: <BarContainer object of 11 artists>



[33]:
```
bf = sorted_imp[5:].attribute.values
lr_bsl_bf = logit(X_train[bf], y_train, X_test[bf], y_test)
rf_bsl_bf = randomForest(X_train[bf], y_train, X_test[bf], y_test)
```

```
Optimization terminated successfully.
         Current function value: 0.628655
         Iterations 5
```

[34]:
```
lr_bsl_bf, rf_bsl_bf
```

[34]: (0.2153547126025551, 0.22501157243688358)

1 2a. Your out-of-sample Brier score on 2018.

[35]:
```
print('Brier Scores ')
print('\033[1mLogit:', lr_bsl)
print('\033[0mRandom Forest without Hyperparameter tuning:', rf_bsl)
```

```
print('Random Forest with Hyperparameter tuning:', rf_GS_bsl)
```

```
Brier Scores
Logit: 0.21502632234832797

Random Forest without Hyperparameter tuning: 0.2209693651033026
Random Forest with Hyperparameter tuning: 0.21661814417100592
```

2 2b. The type of model you fit

I used **Logit** and **Random Forest Classifier**

3 2c. A very brief summary of the features used in your model. Be terse but precise so that the reader can figure out exactly how your features are computed.

Since much of the data is computed after the fact, I decided to use historical representations of the data based on the same stats just from previous games. I also decided to group these by year because stats do not usually translate from season to season.

I implemented historical representations of the following variables * Goal Differential (hAGD) * Shots on Target (hST) * Expected Goals (hxG)

I also created new variables based on the given data and also transformed them into historical representations * Shot Quantity (hsQntD) * Shot Quality (hsQltD) * Team Win Percentage (home_win_pct)

4 2d. A write-up of the process you used to build your model.

I wanted to start with the baseline model introduced in question 1 of the this assignment. Since it was close enough to market implied probabilities, I decided to see if implementing a simple Random Forest would improve the brier score. However, the result proved that Logit was the better initial baseline.

From there I worked to include some more features - all of which were outlined in question 2c. I thought that combining some of the features available to us could be useful in terms of creating new interactions between variables. Shot quantity was especially intriguing to me because, more opportunities generally lead to better results. Tangentially, shot quality should, ideally, be a better indicator because conversion rate of shots on target are much higher.

With a higher complexity data space available, I turned to feature selection in the form of forward and backward selection. After running brier scores on the data set with the selected features the scores got slightly worse. I then went to feature selection via a feature importance plot. In the plot, 6 features stood above the rest, so I used those “best features” on the train/test data. This again drew poorer results. The drawbacks from these two feature selection approaches could be attributed to the fact that 11 columns of feature variables may not be enough to warrant greedy selection, so I decided to keep all 11 columns.

Going into the final model selection, I was able to create a Logit model with a brier score of **0.21503** (which beats the initial baseline measure of **0.21727**), but wanted to bring back random forest with some hyperparameter tuning. To do that, I created a validation set with `Y=17` from the training set and applied a GridSearch (with variable `max_depth` and `num_estimators`). Even with this approach, the Logit model still beat out the hyperparameter tuned Random Forest.

Ultimately, the dataset that I created did not lend itself to creative feature selection techniques, cross-validation, or complex machine learning models. It could be due to the limited number of features that were available or due to not having more datapoints, but in the end the result just proves how powerful Logit models are with simple datasets.