model_1

November 29, 2023

[ ]:
```
from functions import *
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

import warnings
warnings.filterwarnings('ignore')
```

```
2023-11-28 22:21:21.347271: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1 ANALYSIS

[ ]:
```
N = 15
profits = np.zeros(N)
for j in range(N):
        opp_stats = pd.read_csv('DATA_SPORT/Opponent Stats Per 100 Poss.csv')
        team_stats = pd.read_csv('DATA_SPORT/Team Stats Per 100 Poss.csv')
        game = pd.read_csv('DATA/GAME.csv')
        odds = pd.read_csv('DATA/ODDS.csv')
        odds = odds[odds['GAME_ID']!=0]
        odds = odds[odds['HOMEML'].notna()]
        useful_cols =␣
 ↪['season','abbreviation','fg_per_100_poss','x3p_per_100_poss','x2p_per_100_poss','ft_per_10
        team_stats = team_stats[useful_cols]
        for i in range(2,len(useful_cols)):
                useful_cols[i] = 'opp_' + useful_cols[i]
        opp_stats = opp_stats[useful_cols]
        stats = pd.merge(team_stats,opp_stats,on=['season','abbreviation'])
```

```
        useful_cols =␣
↪['GAME_ID','HOMETEAM','AWAYTEAM','WL_HOME','PLUS_MINUS_HOME','HOMEML']
        odds = odds[useful_cols]
        game_dates = game[['GAME_ID','GAME_DATE_HOME']].copy()
        game_dates['YEAR'] = pd.to_datetime(game_dates['GAME_DATE_HOME']).dt.
↪year
        game_dates = game_dates.drop('GAME_DATE_HOME',axis=1)
        odds = pd.merge(odds,game_dates,on='GAME_ID',how='left')
        odds['YEAR-1'] = odds['YEAR'] - 1
        odds = pd.merge(odds, stats, left_on=['HOMETEAM', 'YEAR-1'],␣
↪right_on=['abbreviation', 'season'], suffixes=('', '_Home'))
        odds = pd.merge(odds, stats, left_on=['AWAYTEAM', 'YEAR-1'],␣
↪right_on=['abbreviation', 'season'], suffixes=('', '_Away'))
        drop_cols =␣
↪['YEAR-1','season','abbreviation','season_Away','abbreviation_Away','YEAR']
        odds = odds.drop(drop_cols,axis=1)
        data = odds.copy()
        state = random.randint(1,999)


        data = odds.copy()
        X = data.loc[:, 'fg_per_100_poss':]
        y = data['PLUS_MINUS_HOME']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2,random_state=state)

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        model = Sequential([
                Dense(64, activation='relu', input_shape=(X_train_scaled.
↪shape[1],)),
                Dense(32, activation='relu'),
                Dense(1)  # Single output node for regression
        ])

        model.compile(optimizer='adam', loss='mean_squared_error')
        model.fit(X_train_scaled, y_train, epochs=5, batch_size=4,verbose=0)
        loss = model.evaluate(X_test_scaled, y_test,verbose=0)
        print(f"\nTest Loss: {loss}")

        predictions = model.predict(X_test_scaled,verbose=0)[:,0]
        bin_truth = np.array(y_test) > 0
        bin_pred = predictions > 0
        count = 0
```

```
        for i in range(len(bin_truth)):
                if bin_truth[i] == bin_pred[i]:
                        count +=1
        print("Accuracy: {:<6.2f}%\n".format(count/len(predictions)*100))

        results = pd.DataFrame(y_test)
        results['GAME_ID'] = results.index.map(data['GAME_ID'])
        results['pred'] = predictions
        results = pd.merge(results, pd.read_csv('DATA/ODDS.csv'),on='GAME_ID')
        relevant_cols =␣
↪['GAME_ID','pred','HOMEML','AWAYML','WL_HOME','HOMESPREAD_ATCLOSE']
        results = results[relevant_cols]
        results['PRED_SPREAD'] = results['pred']*-1
        results['HWAGER']=0.0
        results['AWAGER']=0.0
        for i in range(len(results)):
                pred = results['PRED_SPREAD'].iloc[i]
                spread = results['HOMESPREAD_ATCLOSE'].iloc[i]
                awayml = results['AWAYML'].iloc[i]
                homeml = results['HOMEML'].iloc[i]
                # if spread <= 0:
                #         if pred <= 0:
                #                 results.iloc[i,results.columns.
↪get_loc('AWAGER')] = (abs(spread)+abs(pred))
                #         elif pred >= -1*spread:
                #                 results.iloc[i,results.columns.
↪get_loc('HWAGER')] = (abs(spread+pred))
                # elif spread > 0:
                #         if pred > 0:
                #                 results.iloc[i,results.columns.
↪get_loc('HWAGER')] = (abs(spread)+abs(pred))
                #         elif pred < -1*spread:
                #                 results.iloc[i,results.columns.
↪get_loc('AWAGER')] = (abs(spread+pred))
                if abs(spread-pred) >= 2:
                        if (spread > 0 and pred > 0) or (spread < 0 and pred <␣
↪0):
                                if awayml < 2:
                                        results.iloc[i,results.columns.
↪get_loc('AWAGER')] = 1
                        if (spread > 0 and pred < 0) or (spread < 0 and pred >␣
↪0):
                                if homeml < 2:
                                        results.iloc[i,results.columns.
↪get_loc('HWAGER')] = 1
        reg_results = results.copy()
```

```
        data = odds.copy()
        X = data.loc[:, 'fg_per_100_poss':]
        y = data['WL_HOME'].map({'W': 1, 'L': 0})
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2,random_state=state)

        model = Sequential([
                Dense(64, activation='relu', input_shape=(X_train_scaled.
↪shape[1],)),
                Dense(32, activation='relu'),
                Dense(1, activation='sigmoid')  # Single output node for␣
↪regression
        ])

        model.compile(optimizer='adam', loss='binary_crossentropy')
        model.fit(X_train_scaled, y_train, epochs=5, batch_size=2,verbose=0)
        loss = model.evaluate(X_test_scaled, y_test,verbose=0)
        print(f"\nTest Loss: {loss}")

        predictions = model.predict(X_test_scaled,verbose=0)[:,0]
        bin_truth = np.array(y_test) > 0
        bin_pred = predictions > 0
        count = 0
        for i in range(len(bin_truth)):
                if bin_truth[i] == bin_pred[i]:
                        count +=1
        print("Accuracy: {:<6.2f}%\n".format(count/len(predictions)*100))

        results = pd.DataFrame(y_test)
        results['GAME_ID'] = results.index.map(data['GAME_ID'])
        results['pred'] = predictions
        results = pd.merge(results, pd.read_csv('DATA/ODDS.csv'),on='GAME_ID')
        results['WL_HOME'] = results['WL_HOME_y']
        relevant_cols =␣
↪['GAME_ID','pred','HOMEML','AWAYML','WL_HOME','HOMESPREAD_ATCLOSE']
        results = results[relevant_cols]
        results['PRED_SPREAD'] = results['pred']*-1
        results['HWAGER']=0.0
        results['AWAGER']=0.0
        for i in range(len(results)):
                homep = results['pred'].iloc[i]
                awayp = 1-homep
                homeml = results['HOMEML'].iloc[i]
                awayml = results['AWAYML'].iloc[i]
```

```
                if homep*homeml>1.1:
                        results.iloc[i,results.columns.get_loc('HWAGER')] = 1
                if awayp*awayml>1.1:
                        results.iloc[i,results.columns.get_loc('AWAGER')] = 1
        cla_results = results.copy()


        output = reg_results.copy().drop('pred',axis=1)
        print(len(output[output['HWAGER']==1]))
        print(len(output[output['AWAGER']==1]))
        output['PRED_ML'] = cla_results['pred']
        for i in range(len(output)):
                if output['HWAGER'].iloc[i] == 1:
                        if output['PRED_ML'].iloc[i]*output['HOMEML'].iloc[i] <␣
↪.98:
                                output.iloc[i,output.columns.get_loc('HWAGER')]␣
↪= 0
                if output['AWAGER'].iloc[i] == 1:
                        if (1-output['PRED_ML'].iloc[i])*output['AWAYML'].
↪iloc[i] < .98:
                                output.iloc[i,output.columns.get_loc('AWAGER')]␣
↪= 0
        print(len(output[output['HWAGER']==1]))
        print(len(output[output['AWAGER']==1]))
        output.to_csv('DATA/WAGERS.csv')
        profit = calc_results("DATA/WAGERS.csv","DATA/RESULTS.csv")
        profits[j] = profit
        print(j)
```

```
Test Loss: 166.8861083984375
Accuracy: 62.84 %


Test Loss: 0.6441013813018799
Accuracy: 58.53 %

191
297
17
22
                                        $+7.36       on $     39.00:     18.86%
profit.
                                        28     of     39 games won:      71.79%
accuracy.
0
```

```
Test Loss: 172.11061096191406
Accuracy: 63.00 %


Test Loss: 0.6554831862449646
Accuracy: 58.47 %

276
372
15
65
                                    $+4.36       on $     80.00:      5.45%
profit.
                                    54     of     80 games won:      67.50%
accuracy.
1

Test Loss: 167.46975708007812
Accuracy: 64.83 %


Test Loss: 0.6402705907821655
Accuracy: 59.31 %

268
327
51
90
                                    $+4.23       on $    141.00:      3.00%
profit.
                                    93     of    141 games won:      65.96%
accuracy.
2

Test Loss: 173.0765838623047
Accuracy: 62.84 %


Test Loss: 0.643120288848877
Accuracy: 58.63 %

206
325
25
69
                                    $+4.31       on $     94.00:      4.58%
profit.
                                    64     of     94 games won:      68.09%
```

accuracy.
3

Test Loss: 177.88597106933594
Accuracy: 62.94 %

Test Loss: 0.645719051361084
Accuracy: 58.98 %

358
340
39
36
$+0.53 on $ 75.00: 0.71%
profit.
44 of 75 games won: 58.67%
accuracy.
4

Test Loss: 175.73666381835938
Accuracy: 64.19 %

Test Loss: 0.6423584222793579
Accuracy: 59.66 %

176
296
28
38
$+1.75 on $ 66.00: 2.65%
profit.
42 of 66 games won: 63.64%
accuracy.
5

Test Loss: 174.5936737060547
Accuracy: 62.91 %

Test Loss: 0.6509394645690918
Accuracy: 58.28 %

308
348
51
75

```
                                    $+6.09       on $    126.00:      4.83%
profit.
                                    81     of    126 games won:      64.29%
accuracy.
6

Test Loss: 173.71559143066406
Accuracy: 63.10 %


Test Loss: 0.6564046144485474
Accuracy: 59.18 %

251
341
38
61
                                    $+12.12      on $     99.00:     12.24%
profit.
                                    67     of     99 games won:      67.68%
accuracy.
7

Test Loss: 172.40609741210938
Accuracy: 62.81 %


Test Loss: 0.6471518874168396
Accuracy: 59.40 %

408
360
49
58
                                    $+6.13       on $    107.00:      5.73%
profit.
                                    68     of    107 games won:      63.55%
accuracy.
8

Test Loss: 169.87428283691406
Accuracy: 63.07 %


Test Loss: 0.6428994536399841
Accuracy: 59.43 %

322
```

```
329
47
56
                                        $+2.07       on $    103.00:      2.01%
profit.
                                        61     of    103 games won:      59.22%
accuracy.
9

Test Loss: 174.13623046875
Accuracy: 63.39 %


Test Loss: 0.6497077345848083
Accuracy: 58.05 %

228
344
26
57
                                        $+1.88       on $     83.00:      2.27%
profit.
                                        53     of     83 games won:      63.86%
accuracy.
10

Test Loss: 166.4166259765625
Accuracy: 62.17 %


Test Loss: 0.6542637944221497
Accuracy: 58.76 %

372
366
16
71
                                        $+6.59       on $     87.00:      7.58%
profit.
                                        57     of     87 games won:      65.52%
accuracy.
11

Test Loss: 163.20538330078125
Accuracy: 63.13 %


Test Loss: 0.6415192484855652
```

```
Accuracy: 59.98 %

446
351
37
58
                                        $+3.14       on $     95.00:      3.31%
profit.
                                        60     of     95 games won:      63.16%
accuracy.
12

Test Loss: 172.1002960205078
Accuracy: 64.13 %


Test Loss: 0.6436160802841187
Accuracy: 60.66 %

260
310
42
55
                                        $+11.34      on $     97.00:     11.69%
profit.
                                        66     of     97 games won:      68.04%
accuracy.
13

Test Loss: 172.03858947753906
Accuracy: 64.55 %


Test Loss: 0.6440562605857849
Accuracy: 59.37 %

239
339
38
38
                                        $+2.46       on $     76.00:      3.24%
profit.
                                        47     of     76 games won:      61.84%
accuracy.
14
```

[ ]:
```
profits
```

[ ]:
```
array([18.86314088,  5.45030041,  2.99741124,  4.5825057 ,  0.71145733,
        2.64777878,  4.83179334, 12.24315533,  5.72989093,  2.01016597,
        2.26817448,  7.57944266,  3.30734322, 11.69014367,  3.2402458 ])
```

2 previous

[ ]:
```
data = odds.copy()
X = data.loc[:, 'fg_per_100_poss':]
y = data['PLUS_MINUS_HOME']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=state)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dense(32, activation='relu'),
        Dense(1)  # Single output node for regression
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train_scaled, y_train, epochs=5, batch_size=4,verbose=1)
loss = model.evaluate(X_test_scaled, y_test,verbose=0)
print(f"Test Loss: {loss}\n")

predictions = model.predict(X_test_scaled,verbose=0)[:,0]
bin_truth = np.array(y_test) > 0
bin_pred = predictions > 0
count = 0
for i in range(len(bin_truth)):
        if bin_truth[i] == bin_pred[i]:
                count +=1
print("Accuracy: {:<6.2f}%".format(count/len(predictions)*100))

results = pd.DataFrame(y_test)
results['GAME_ID'] = results.index.map(data['GAME_ID'])
results['pred'] = predictions
results = pd.merge(results, pd.read_csv('DATA/ODDS.csv'),on='GAME_ID')
relevant_cols =␣
 ↪['GAME_ID','pred','HOMEML','AWAYML','WL_HOME','HOMESPREAD_ATCLOSE']
results = results[relevant_cols]
results['PRED_SPREAD'] = results['pred']*-1
results['HWAGER']=0.0
results['AWAGER']=0.0
```

```
for i in range(len(results)):
        pred = results['PRED_SPREAD'].iloc[i]
        spread = results['HOMESPREAD_ATCLOSE'].iloc[i]
        awayml = results['AWAYML'].iloc[i]
        homeml = results['HOMEML'].iloc[i]
        # if spread <= 0:
        #         if pred <= 0:
        #                 results.iloc[i,results.columns.get_loc('AWAGER')] =
 ↪(abs(spread)+abs(pred))
        #         elif pred >= -1*spread:
        #                 results.iloc[i,results.columns.get_loc('HWAGER')] =
 ↪(abs(spread+pred))
        # elif spread > 0:
        #         if pred > 0:
        #                 results.iloc[i,results.columns.get_loc('HWAGER')] =
 ↪(abs(spread)+abs(pred))
        #         elif pred < -1*spread:
        #                 results.iloc[i,results.columns.get_loc('AWAGER')] =
 ↪(abs(spread+pred))
        if abs(spread-pred) >= 1:
                if (spread > 0 and pred > 0) or (spread < 0 and pred < 0):
                        if awayml < 2:
                                results.iloc[i,results.columns.
 ↪get_loc('AWAGER')] = 1
                if (spread > 0 and pred < 0) or (spread < 0 and pred > 0):
                        if homeml < 2:
                                results.iloc[i,results.columns.
 ↪get_loc('HWAGER')] = 1

reg_results = results.copy()
```

```
Epoch 1/5
3111/3111 [==============================] - 6s 2ms/step - loss: 175.9701
Epoch 2/5
3111/3111 [==============================] - 5s 2ms/step - loss: 171.6916
Epoch 3/5
3111/3111 [==============================] - 5s 2ms/step - loss: 168.9295
Epoch 4/5
3111/3111 [==============================] - 5s 2ms/step - loss: 166.2634
Epoch 5/5
3111/3111 [==============================] - 5s 1ms/step - loss: 164.0597
Test Loss: 169.03414916992188

Accuracy: 64.03 %
```

[ ]:
```
data = odds.copy()
X = data.loc[:, 'fg_per_100_poss':]
```

```
y = data['WL_HOME'].map({'W': 1, 'L': 0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=state)

model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')  # Single output node for regression
])

model.compile(optimizer='adam', loss='binary_crossentropy')
model.fit(X_train_scaled, y_train, epochs=5, batch_size=2,verbose=1)
loss = model.evaluate(X_test_scaled, y_test,verbose=0)
print(f"Test Loss: {loss}\n")

predictions = model.predict(X_test_scaled,verbose=0)[:,0]
bin_truth = np.array(y_test) > 0
bin_pred = predictions > 0
count = 0
for i in range(len(bin_truth)):
        if bin_truth[i] == bin_pred[i]:
                count +=1
print("Accuracy: {:<6.2f}%".format(count/len(predictions)*100))

results = pd.DataFrame(y_test)
results['GAME_ID'] = results.index.map(data['GAME_ID'])
results['pred'] = predictions
results = pd.merge(results, pd.read_csv('DATA/ODDS.csv'),on='GAME_ID')
results['WL_HOME'] = results['WL_HOME_y']
relevant_cols =␣
 ↪['GAME_ID','pred','HOMEML','AWAYML','WL_HOME','HOMESPREAD_ATCLOSE']
results = results[relevant_cols]
results['PRED_SPREAD'] = results['pred']*-1
results['HWAGER']=0.0
results['AWAGER']=0.0
for i in range(len(results)):
        homep = results['pred'].iloc[i]
        awayp = 1-homep
        homeml = results['HOMEML'].iloc[i]
        awayml = results['AWAYML'].iloc[i]

        if homep*homeml>1.1:
                results.iloc[i,results.columns.get_loc('HWAGER')] = 1
        if awayp*awayml>1.1:
                results.iloc[i,results.columns.get_loc('AWAGER')] = 1

cla_results = results.copy()
```

```
Epoch 1/5
6221/6221 [==============================] - 12s 2ms/step - loss: 0.6575
Epoch 2/5
6221/6221 [==============================] - 11s 2ms/step - loss: 0.6463
Epoch 3/5
6221/6221 [==============================] - 10s 2ms/step - loss: 0.6399
Epoch 4/5
6221/6221 [==============================] - 10s 2ms/step - loss: 0.6338
Epoch 5/5
6221/6221 [==============================] - 10s 2ms/step - loss: 0.6272
Test Loss: 0.6407946944236755

Accuracy: 59.79 %
```

[ ]:
```
output = reg_results.copy().drop('pred',axis=1)
print(len(output[output['HWAGER']==1]))
print(len(output[output['AWAGER']==1]))
output['PRED_ML'] = cla_results['pred']
for i in range(len(output)):
        if output['HWAGER'].iloc[i] == 1:
                if output['PRED_ML'].iloc[i]*output['HOMEML'].iloc[i] < 1:
                        output.iloc[i,output.columns.get_loc('HWAGER')] = 0
        if output['AWAGER'].iloc[i] == 1:
                if (1-output['PRED_ML'].iloc[i])*output['AWAYML'].iloc[i] < 1:
                        output.iloc[i,output.columns.get_loc('AWAGER')] = 0
print(len(output[output['HWAGER']==1]))
print(len(output[output['AWAGER']==1]))
output.to_csv('DATA/WAGERS.csv')
profit = calc_results("DATA/WAGERS.csv","DATA/RESULTS.csv")
```

```
381
500
48
98
                                        $+23.06      on $    146.00:     15.79%
profit.
                                        103    of    146 games won:      70.55%
accuracy.
```