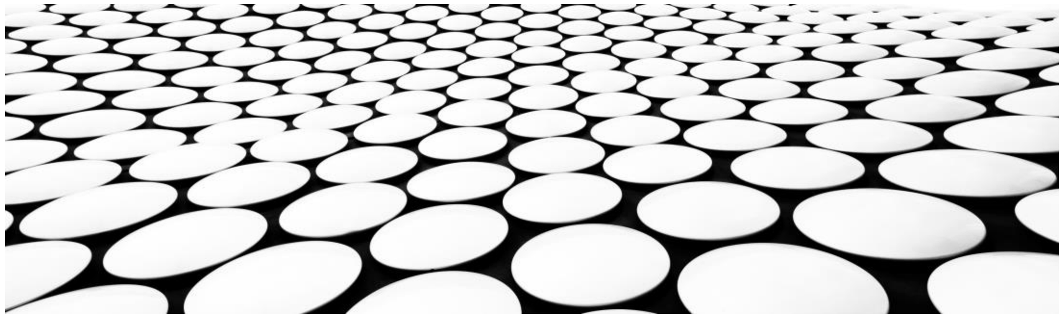
ATP Player Hand Analysis And Prediction

CAPSTONE PROJECT 1



Overview

- Problem
- The data
- Data Wrangling and preparation
- Exploratory Data Analysis
- Statistical Inference
- Machine Learning

Problem

- Do ATP left hand players have advantage against ATP right hand players?
- Can we predict if an ATP player is righty or lefty based on their match results?

The Data

- The data used for this project comes from Github.
- The dataset includes ATP match results from 2000 to 2019

```
['data/tennis_atp/match_00_19/atp_matches_2000.csv',
 'data/tennis_atp/match_00_19/atp_matches_2001.csv',
 'data/tennis_atp/match_00_19/atp_matches_2002.csv',
 'data/tennis_atp/match_00_19/atp_matches_2003.csv',
 'data/tennis_atp/match_00_19/atp_matches_2004.csv',
 'data/tennis_atp/match_00_19/atp_matches_2005.csv',
 'data/tennis_atp/match_00_19/atp_matches_2006.csv',
 'data/tennis_atp/match_00_19/atp_matches_2007.csv',
 'data/tennis_atp/match_00_19/atp_matches_2008.csv',
 'data/tennis_atp/match_00_19/atp_matches_2009.csv',
 'data/tennis_atp/match_00_19/atp_matches_2010.csv',
 'data/tennis_atp/match_00_19/atp_matches_2011.csv',
 'data/tennis_atp/match_00_19/atp_matches_2012.csv',
 'data/tennis_atp/match_00_19/atp_matches_2013.csv',
 'data/tennis_atp/match_00_19/atp_matches_2014.csv',
 'data/tennis_atp/match_00_19/atp_matches_2015.csv',
 'data/tennis_atp/match_00_19/atp_matches_2016.csv',
 'data/tennis_atp/match_00_19/atp_matches_2017.csv',
 'data/tennis_atp/match_00_19/atp_matches_2018.csv',
 'data/tennis_atp/match_00_19/atp_matches_2019.csv']
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61560 entries, 0 to 2780
Data columns (total 49 columns):
tourney_id            61560 non-null object
tourney_name          61560 non-null object
surface               61442 non-null object
draw_size             2781 non-null float64
tourney_level         61560 non-null object
tourney_date          61560 non-null int64
match_num             61560 non-null int64
winner_id             61560 non-null int64
winner_seed           25567 non-null object
winner_entry          7346 non-null object
winner_name           61560 non-null object
winner_hand           61542 non-null object
winner_ht             56229 non-null float64
winner_ioc            61560 non-null object
winner_age            61545 non-null float64
loser_id              61560 non-null int64
loser_seed            13973 non-null object
loser_entry           12107 non-null object
loser_name            61560 non-null object
loser_hand            61514 non-null object
loser_ht              53389 non-null float64
loser_ioc             61560 non-null object
loser_age             61529 non-null float64
score                 61559 non-null object
best_of               61560 non-null int64
round                 61560 non-null object
minutes               54478 non-null float64
w_ace                 55781 non-null float64
w_df                  55781 non-null float64
w_svpt                55781 non-null float64
w_1stIn               55781 non-null float64
w_1stWon              55781 non-null float64
w_2ndWon              55781 non-null float64
w_SvGms               55781 non-null float64
w_bpSaved             55781 non-null float64
w_bpFaced             55781 non-null float64
l_ace                 55781 non-null float64
l_df                  55781 non-null float64
l_svpt                55781 non-null float64
l_1stIn               55781 non-null float64
l_1stWon              55781 non-null float64
l_2ndWon              55781 non-null float64
l_SvGms               55781 non-null float64
l_bpSaved             55781 non-null float64
l_bpFaced             55781 non-null float64
winner_rank           61057 non-null float64
winner_rank_points    61057 non-null float64
loser_rank            60263 non-null float64
loser_rank_points     60263 non-null float64
dtypes: float64(28), int64(5), object(16)
memory usage: 23.5+ MB
```

Data Wrangling and Preparation

- Removing extraneous data – i.e.,
  - Removing rows with 100% missing metric values
  - Removing some categorical columns not needed
- Feature engineering—i.e.,
  - computing and adding columns for data exploration and machine learning
  - Prepared data set of ATP matches between righties and lefties

```
df.loc[df.player_hand=='R', 'player_hand_flag'] = 1
df.loc[df.player_hand=='L', 'player_hand_flag'] = 0
```

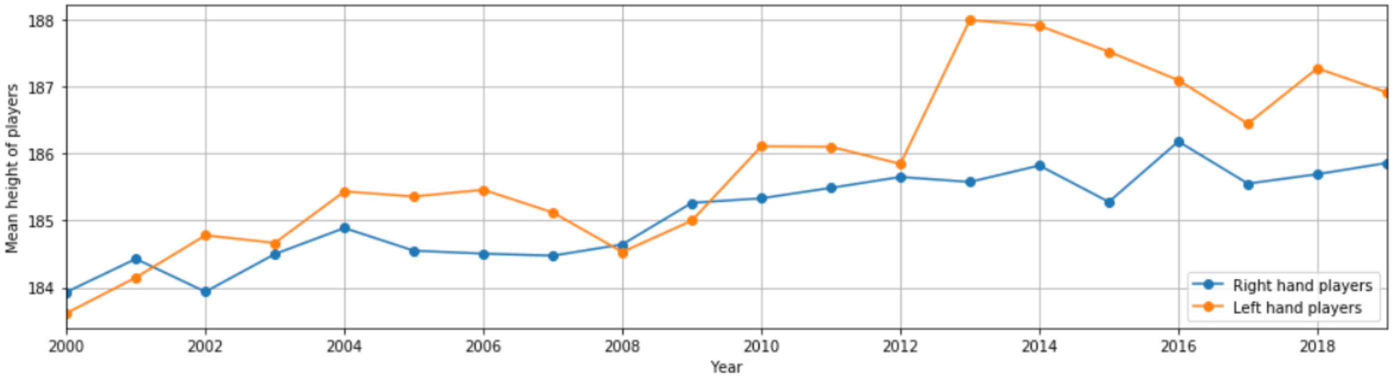
```
df_all['svpt_won_pct']= np.around((df_all.sv1stWon + df_all.sv2ndWon)/df_all.svpt,2)
df_all['svpt_std_var']=df_all.svpt - df_all.SvGms*4
df_all['bpSaved_pct'] = np.around(df_all.bpSaved/df_all.bpFaced,2)
df_all.bpSaved_pct.fillna(0, inplace=True)
```

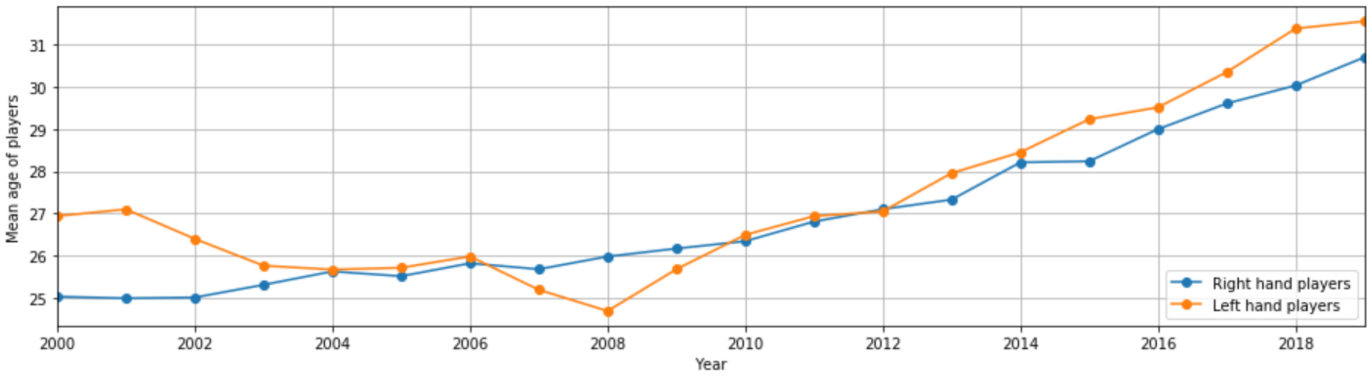
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20132 entries, 0 to 20131
Data columns (total 48 columns):
Unnamed: 0            20132 non-null int64
tourney_id            20132 non-null object
tourney_name          20132 non-null object
surface               20132 non-null object
tourney_level         20132 non-null object
tourney_date          20132 non-null int64
match_num             20132 non-null int64
player_id             20132 non-null int64
player_name           20132 non-null object
player_hand           20132 non-null object
player_ht             20132 non-null float64
player_ioc            20132 non-null object
player_age            20132 non-null float64
score                 20132 non-null object
best_of               20132 non-null int64
round                 20132 non-null object
minutes               20132 non-null float64
ace                   20132 non-null float64
df                    20132 non-null float64
svpt                  20132 non-null float64
sv1stIn               20132 non-null float64
sv1stWon              20132 non-null float64
sv2ndWon              20132 non-null float64
SvGms                 20132 non-null float64
bpSaved               20132 non-null float64
bpFaced               20132 non-null float64
player_rank           20132 non-null float64
player_rank_points    20132 non-null float64
ace_pct               20132 non-null float64
df_pct                20132 non-null float64
sv1stIn_pct           20132 non-null float64
sv2ndIn_pct           20132 non-null float64
sv1stWon_pct          20132 non-null float64
sv2ndWon_pct          20132 non-null float64
GmsWon                20132 non-null float64
GmsLoss               20132 non-null float64
year                  20132 non-null int64
opponent_id           20132 non-null int64
opponent_name         20132 non-null object
won_flag              20132 non-null int64
player_age_bucket     20132 non-null object
player_hand_flag      20132 non-null float64
surface_id            20132 non-null float64
tourney_level_id      20132 non-null float64
player_rank_group     20132 non-null float64
svpt_won_pct          20132 non-null float64
svpt_std_var          20132 non-null float64
bpSaved_pct           20132 non-null float64
dtypes: float64(29), int64(8), object(11)
memory usage: 7.4+ MB
```

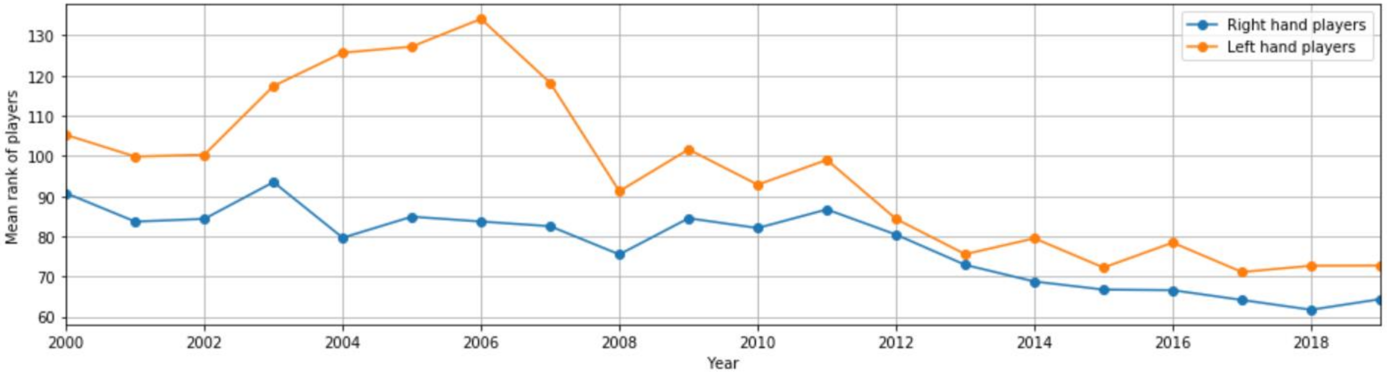
Exploratory Data Analysis

Some observations:

- Average height of lefties are taller than the righties played in the same year except year 2001, 2008 and 2009
- Except 2007 - 2009, the average age of lefties are older than righties
- The average of rankings of righties are higher than lefties



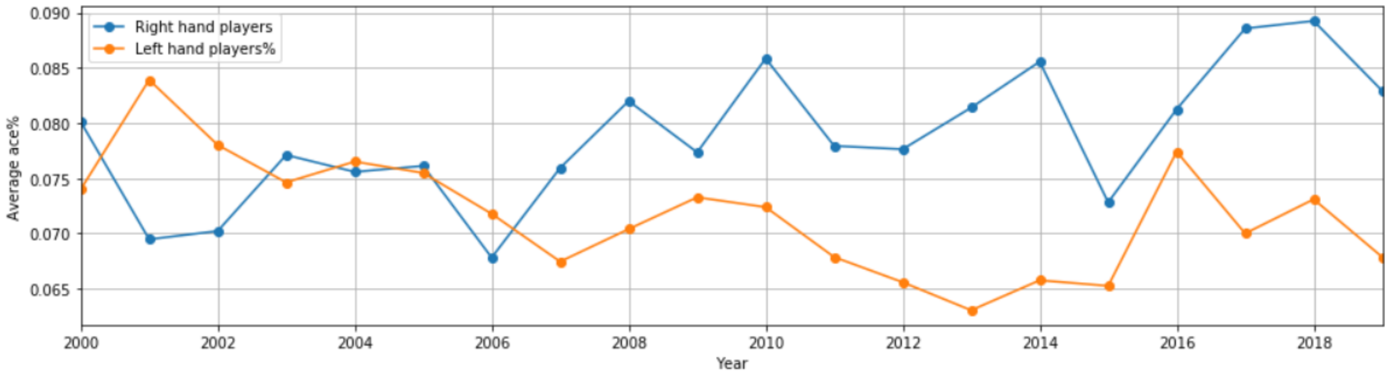


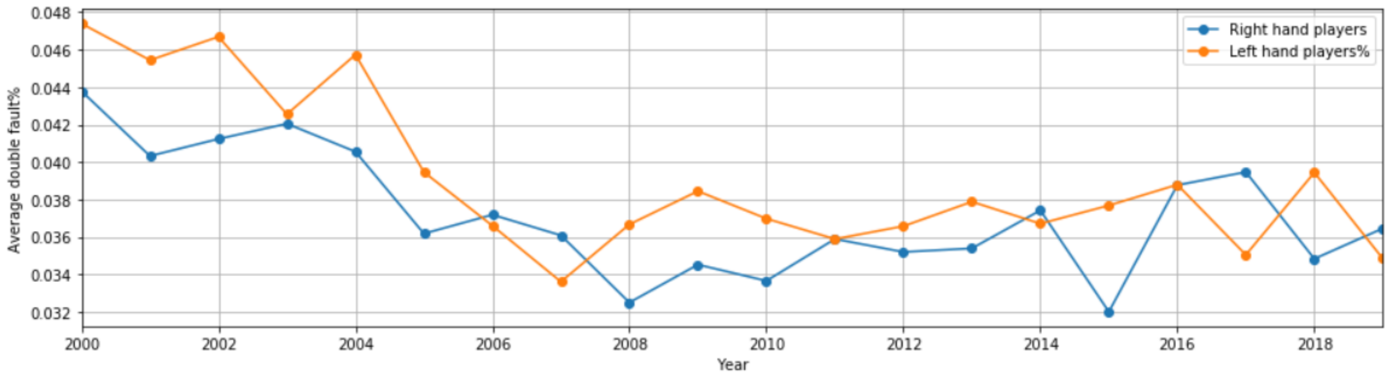


Exploratory Data Analysis

Some observations:

- Compared 20 year mean ace percentage for righties and lefties, the righties had higher ace percentage than lefties
- Lefties has a little higher double fault percentage than righties
- Compared 20 year matches, lefties do have higher first in server percentage
- The first serve won percentage seems no significant difference between righties and lefties
- Righties also have higher second serve in percentage than lefties





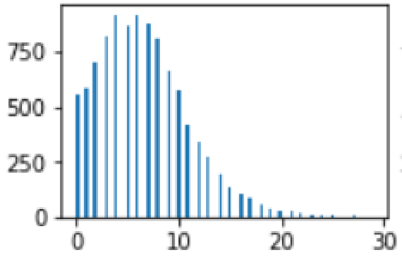
Statistical Inference

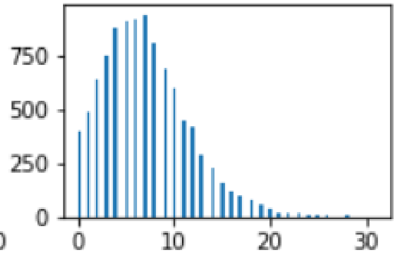
Some observations:

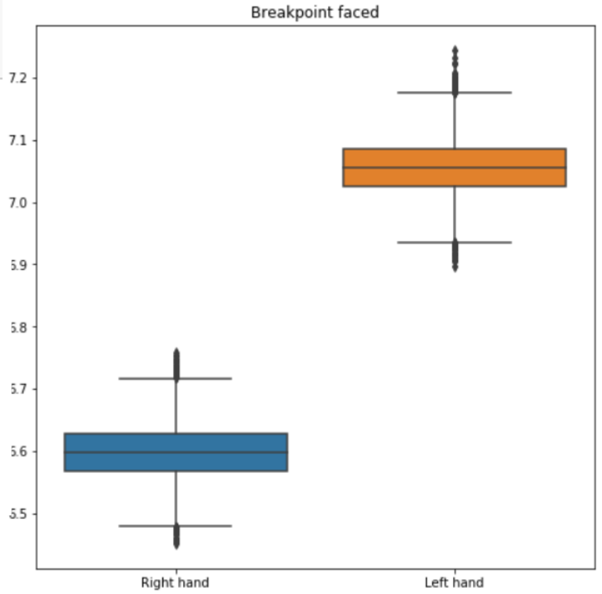
- Performed t-test to check if there are significant difference on serve stats (ace%, 1stIn%, 1stWon%, 2ndIn%, 2ndWon%, bpFaced, games won and game loss between righties and lefties. Only 1stWon% doesn't have significant difference between righties and lefties. The rest null hypothesis are rejected.
- Then, performed bootstrap to generate sample mean distribution for righties and lefties. Calculated mean, 95% confidence interval and plot boxplot. Lefties have better 1stIn% than righties. Righties have advantages of the rest.

```
lefties = df[df.player_hand=='L'].bpFaced
righties = df[df.player_hand=='R'].bpFaced
s, p = stats.ttest_ind(lefties, righties, equal_var = False)
print('T test statistic = ' + str(s))
print('T test p-value = ' + str(p))
```

```
T test statistic = 7.326820695264647
T test p-value = 2.445761838339517e-13
```








```
bs_rep_r = draw_bs_reps(righties, np.mean, N_rep)
bs_rep_l = draw_bs_reps(lefties, np.mean, N_rep)
r_int_low, r_int_high = np.percentile(bs_rep_r, [2.5, 97.5])
l_int_low, l_int_high = np.percentile(bs_rep_l, [2.5, 97.5])


print('righties mean ='+str(bs_rep_r.mean()), 'lefties mean = '+ str(bs_rep_l.mean()))
print('righties 95% confidence interval: '+ '[' + str(r_int_low) + ',' + str(r_int_high)+']')
print('lefties 95% confidence interval: '+ '[' + str(l_int_low) + ',' + str(l_int_high)+']')
```

```
righties mean =6.597391366977946 lefties mean = 7.055982286906419
righties 95% confidence interval: [6.512318696602424,6.683491456387841]
lefties 95% confidence interval: [6.969103914166501,7.141965527518379]
```

Machine Learning – Algorithms, Predictors and Metrics used

Algorithm
KNeighborsClassifier
RandomForestClassifier
GradientBoostingClasifier

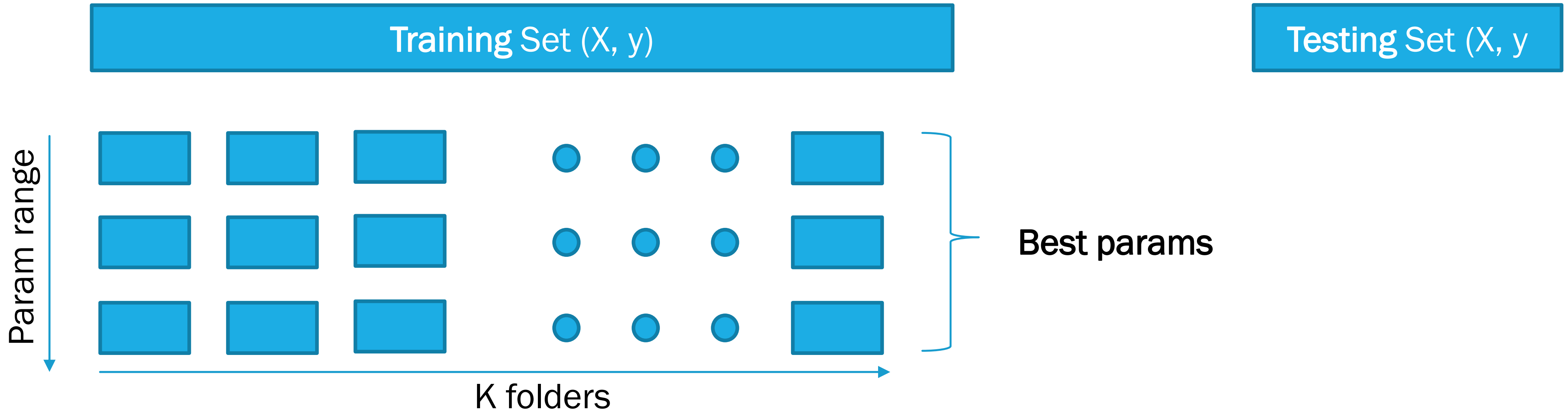
- Response variable: player_hand_flag
- Binary classification: righty – 1; lefty - 0
- In the data set, the records of player hand (lefties and righties) are balanced
- Metrics measure the model performance:
  - AUC
  - Precise
  - Recall
  - F1

Predictor Set 1
'sv1stIn_pct'
'sv1stWon_pct'
'svpt_won_pct'
'sv2ndWon_pct'
'ace_pct'
'df_pct'
'bpFaced'
'bpSaved'
'player_age'
'player_rank'
'svpt_std_var'
'bpSaved_pct'

Predictor Set 2
'sv1stIn'
'sv1stWon'
'svpt'
'sv2ndWon'
'ace'
'df'
'SvGms'
'sv1stIn_pct'
'sv1stWon_pct'
'svpt_won_pct'
'sv2ndWon_pct'
'ace_pct'
'df_pct'
'bpSaved_pct'
'bpFaced'
'bpSaved'
'player_age'
'player_rank'
'player_rank_points'
'player_ht'
'svpt_std_var'

Predictor Set 3
sv1stIn_pct'
'sv1stWon_pct'
'svpt_won_pct'
'sv2ndWon_pct'
'ace_pct'
'df_pct'
'bpFaced'
'bpSaved'
'player_age'
'player_rank'
'svpt_std_var'
'bpSaved_pct'
'player_ht'
'player_rank_points'

Fine Tuning Hyperparameters




```python
def knn_cross_val_k_search(X,y,cv=5):
    k_range = range(1, 31)
    k_scores = []

    #loop through reasonable values of k
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=cv, scoring='roc_auc')
        k_scores.append(scores.mean())
    print(k_scores)

    plt.plot(k_range, k_scores, marker='o')
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()

    return k_range, k_scores
```

- Techniques used
  - GridSearchCV
  - cross_val_scores

K Neighbors Classifier

- Run KNN against two data sets
- For data using predictor set1, the best n_neighbors = 26
- For data using predictor set2, the best n_neighbors = 7
- Neither models perform well

Metrics of test of predictor set 1:

```
classification report:
              precision    recall  f1-score   support

         0.0       0.57      0.59      0.58      1994
         1.0       0.58      0.55      0.57      2033

    accuracy                           0.57      4027
   macro avg       0.57      0.57      0.57      4027
weighted avg       0.57      0.57      0.57      4027

confusion matrix:
[[1184  810]
 [ 908 1125]]
```

```
ROC AUC Score:
0.6004416347912404
```

Metrics of test of predictor set 2:

```
classification report:
              precision    recall  f1-score   support

         0.0       0.58      0.65      0.62      1994
         1.0       0.62      0.55      0.58      2033

    accuracy                           0.60      4027
   macro avg       0.60      0.60      0.60      4027
weighted avg       0.60      0.60      0.60      4027

confusion matrix:
[[1301  693]
 [ 924 1109]]
```

```
ROC AUC Score:
0.6351447357320362
```

Random Forest Classifier – test 1

- Data of predictor set 1
- Run RandomForestClassifier using default parameters

```
classification report:
              precision    recall  f1-score   support

         0.0       0.56      0.68      0.62      2014
         1.0       0.60      0.47      0.53      2013

    accuracy                           0.58      4027
   macro avg       0.58      0.58      0.57      4027
weighted avg       0.58      0.58      0.57      4027

confusion matrix:
[[1371  643]
 [1064  949]]
```

```
roc auc: 0.576085262082462
```

```
model_columns=['sv1stIn_pct','sv1stWon_pct','svpt_won_pct','sv2ndWon_pct','ace_pct','df_pct',
               'bpFaced','bpSaved', 'player_age', 'player_rank', 'svpt_std_var', 'bpSaved_pct']
names= df[model_columns].columns

X=df[model_columns]
y=df['player_hand_flag']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=p_test_size,  random_state= p_random_state, stratify=y)
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Random Forest Classifier - test 2

- Data of predictor set 1
- Hyperparameters:
  - n_estimators = 15
  - max_depth = 6

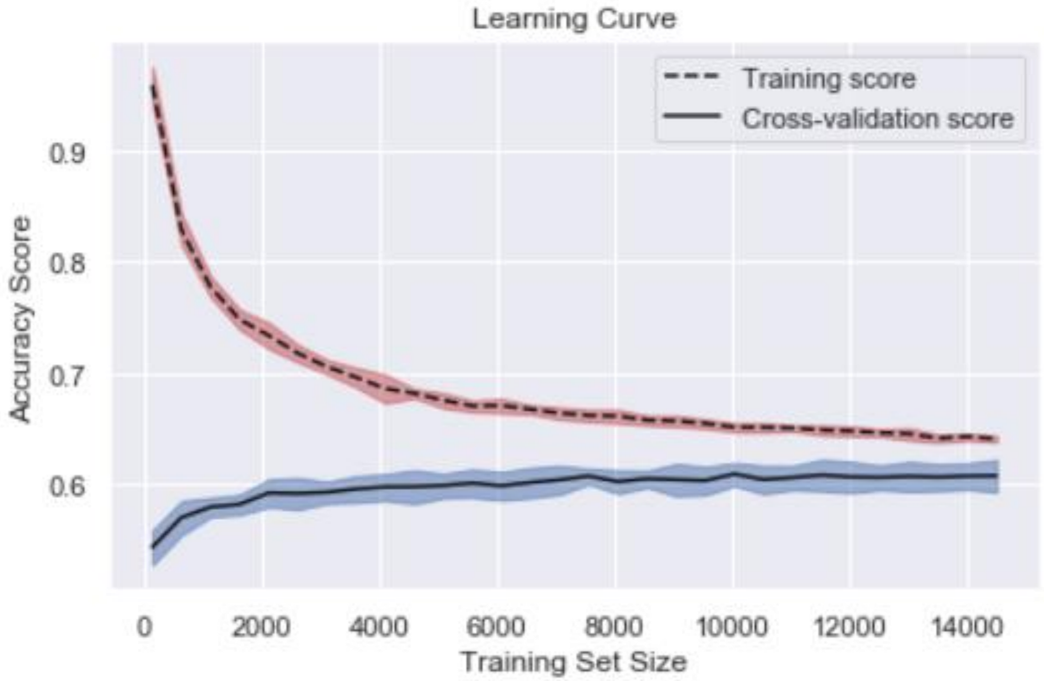
classification report:

	precision	recall	f1-score	support
0.0	0.57	0.74	0.64	2014
1.0	0.63	0.44	0.52	2013
accuracy			0.59	4027
macro avg	0.60	0.59	0.58	4027
weighted avg	0.60	0.59	0.58	4027

confusion matrix:

```
[[1486  528]
 [1119  894]]
```

ROC AUC Score:
0.6401381339071606



Random Forest Classifier - test 3

- Data of predictor set 2
- Hyperparameters:
  - n_estimators = 100
  - max_depth = 8

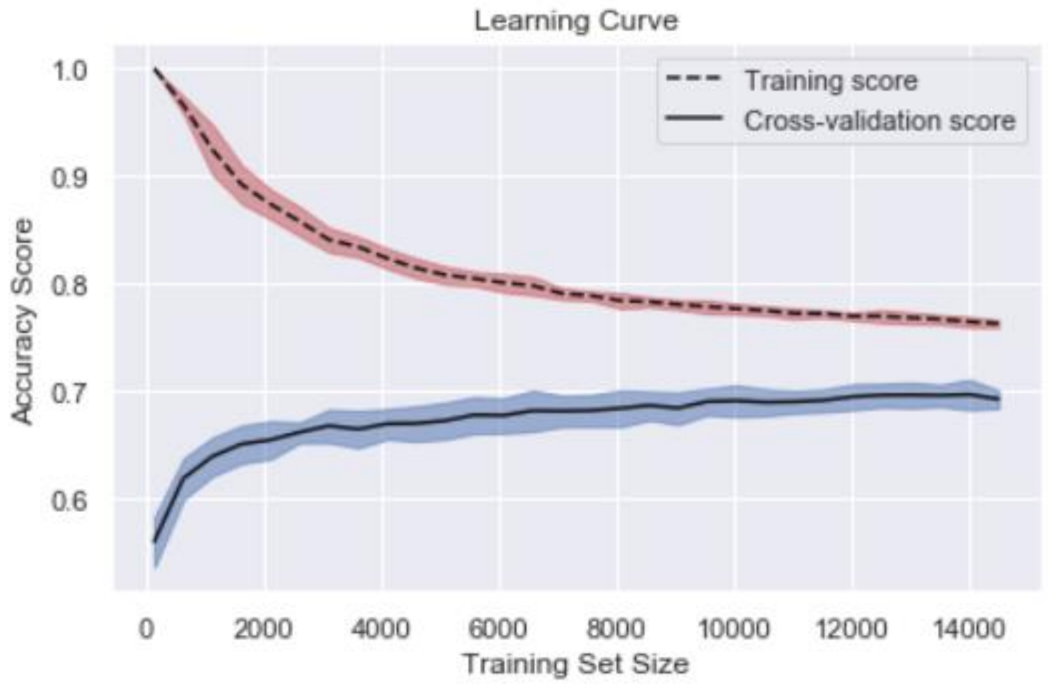
```
classification report:
              precision    recall  f1-score   support

         0.0       0.67      0.77      0.71      2014
         1.0       0.72      0.61      0.66      2013

    accuracy                           0.69      4027
   macro avg       0.69      0.69      0.69      4027
weighted avg       0.69      0.69      0.69      4027

confusion matrix:
[[1542  472]
 [ 776 1237]]
```

```
ROC AUC Score:
0.7671996471791351
```



Gradient Boosting Classifier - test 1

- Data of predictor set 1
- Hyperparameters:
  - learning_rate = 0.05
  - n_estimators = 30
  - max_depth = 5
  - Mam_features = 8

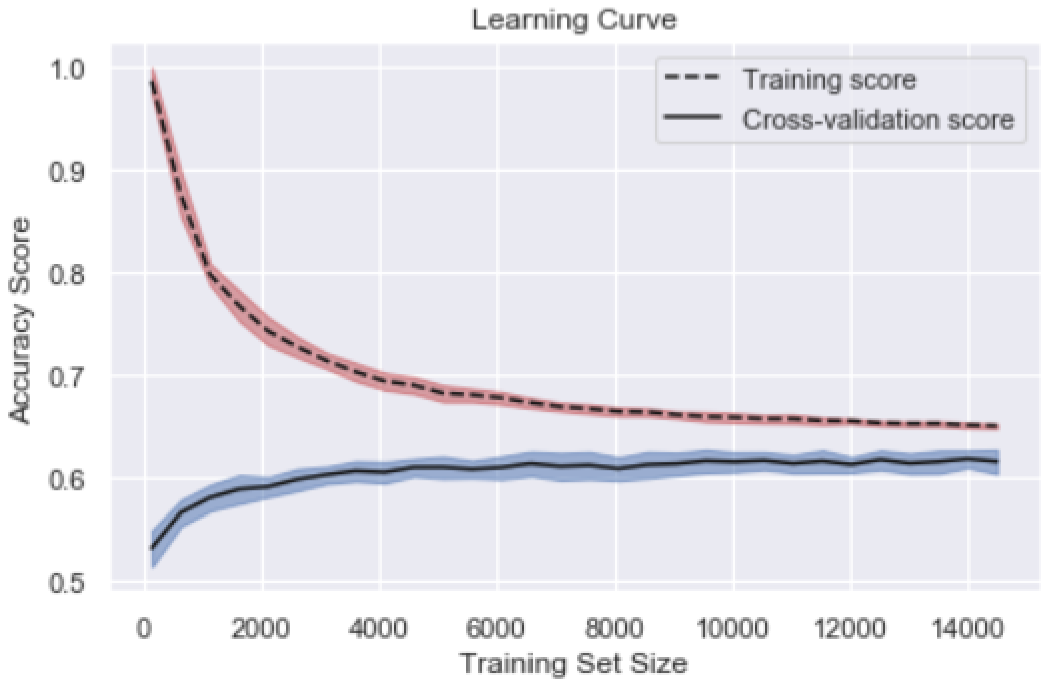
classification report:

	precision	recall	f1-score	support
0.0	0.59	0.75	0.66	1994
1.0	0.66	0.49	0.56	2033
accuracy			0.62	4027
macro avg	0.63	0.62	0.61	4027
weighted avg	0.63	0.62	0.61	4027

confusion matrix:

```
[[1490  504]
 [1040  993]]
```

ROC AUC Score:
0.6868755534286O9



Gradient Boosting Classifier - test 2

- Data of predictor set 2
- Hyperparameters:
  - learning_rate = 0.07
  - n_estimators = 20
  - max_depth = 15
  - Mam_features = 17

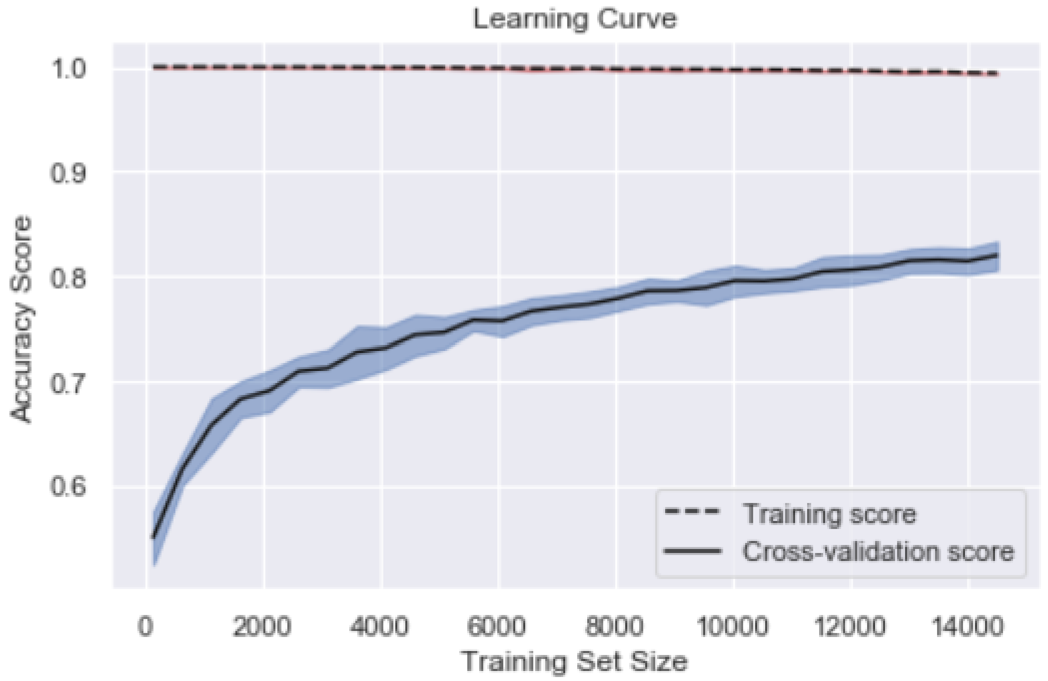
classification report:

	precision	recall	f1-score	support
0.0	0.78	0.87	0.82	2014
1.0	0.86	0.76	0.80	2013
accuracy			0.81	4027
macro avg	0.82	0.81	0.81	4027
weighted avg	0.82	0.81	0.81	4027

confusion matrix:

```
[[1756  258]
 [ 487 1526]]
```

ROC AUC Score:
0.9034054218582195



Gradient Boosting Classifier - test 3

- Data of predictor set 3
- Hyperparameters:
  - learning_rate = 0.07
  - n_estimators = 20
  - max_depth = 15
  - Mam_features = 10

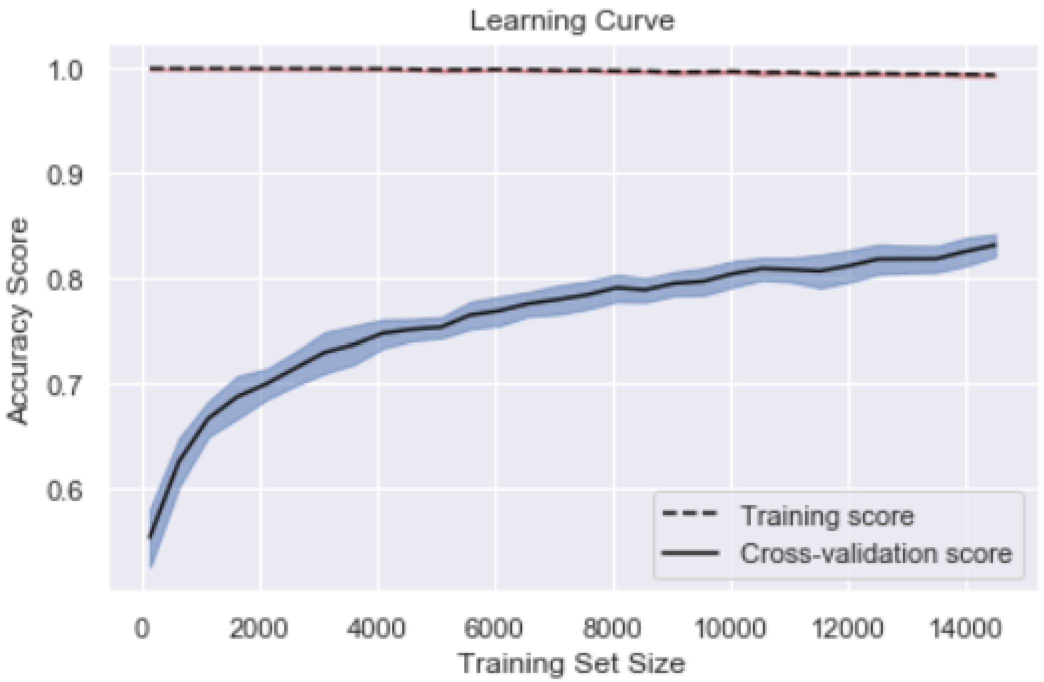
```
classification report:
              precision    recall  f1-score   support

         0.0       0.80      0.88      0.84      2014
         1.0       0.87      0.78      0.83      2013

    accuracy                           0.83      4027
   macro avg       0.84      0.83      0.83      4027
weighted avg       0.84      0.83      0.83      4027

confusion matrix:
[[1780  234]
 [ 435 1578]]
```

```
ROC AUC Score:
0.9155597849331876
```



Conclusion

- Compared the performance between KNeighborClassifier, RandomForestClassifier and Gradient Boosting. Gradient Boosting model performed best.
- It takes more than one features to predict player hand. As a bagging ensemble, Random Forest model performs better than classifier (KNN) and regressor (Logistic). To further reduce variance and bias, boosting ensemble Gradient Boosting (GBM) algorithm performs better than Random Forest.
- Gradient Boosting model has the ROC AUC score at 91%. Gradient Boosting learning curve shows the potential to be further tuned.
- Possible Next steps:
  - Reduce overfitting by In-depth fine hyperparameters tuning for Gradient Boosting algorithm or through feature selection (increase or decrease complexity)
  - Would like to try XGBoost