Capstone Project 3: Prediction of NBA Game Winners

1. Problem statement

The goal of this project is to build a model to predict a game's winner using statistics of previous game played. All the information was imported from NBA's official website, and the time span of the study is in the season of 2019-2020.

Deliverables: code, a written report, and a slide deck.

2. Background

NBA is the second most popular sport in North America, and the third most popular sport in the world. Being able to predict game winner is essential and it has been something that analysts trying to do in the last few decades.

General managers and coaches of each team really would like to predict game's result when it's getting closer to the end of a season. They rely on that information to decide if they have a shot to the playoffs and decide whether they will continue fight for a playoff spot or rest all their major player to avoid any possible injuries. NBA players have protected contracts so they will get full salary even if they get injured and couldn't contribute at all.

Also, more than half of the teams (16 out of 30) can make it to playoffs so it's very important for them to predict game result to see which seeds they are likely to be. They can use the result to decide if they can try new strategies which they can use in the playoffs or they will need to stick to their old plans to fight or keep a spot.

Furthermore, as one of the most popular sports, NBA attracts tens and millions of people to bet on the games. The money involved is enormous. All gambling websites and gamblers are scratching their heads to find out a way to precisely predict the game result.

Home team has slight advantages due to several reasons like players are more familiar with the stadium, audience support, etc. In the season I picked to do prediction, home team won 55% of the time. And therefore, the target of the project was to beat that number.

3. Data inspection and cleaning

I retrieve two parts of data from NBA official website, traditional and advanced.

Traditional data includes the following columns for each game: Team, Match up, Game date, W/L, MIN (Minutes), PTS (Points), FGM (Field goal made), FGA (Field goal attempts), FG% (Field goal %), 3PM (3-pointers made), 3PA (3-pointers attempts), 3P% (3-pointers %), FTM (Free throw made), FTA (Free throw attempt), FT% (Free throw %), OREB (Offensive rebounds), DREB (Defensive rebounds), REB (Rebounds), AST (Assists), STL (Steals), BLK (Blocks), TOV (turnovers), PF (Personal fouls), and +/- (Points differential).

The url of the traditional data source is the following:
https://www.nba.com/stats/teams/boxscores/?Season=2019-20&SeasonType=Regular%20Season&sort=gdate&dir=-1

Advanced data includes the following columns for each game: Team, Match up, Game date, W/L, MIN (Minutes), OFFRTG (points scored per 100 possessions), DEFRTG (points allowed per 100 possessions), NETRTG (point differential per 100 possessions), AST% (% of assistance), AST_Ratio (Assists made per 100 possessions), OREB% (Offensive rebounds %), DREB% (Defensive rebounds %), REB% (Rebound %), TOV% (Turnovers %), EFTG% (Field goals % factoring in the points a shot can make), TS% (Shooting percentage factoring in the points a shot can make), PACE (Possessions per 48 minutes), PIE (PIE yields results which are comparable to other advanced statistics (e.g. PER) using a simple formula and the following is the formula: PTS + FGM + FTM - FGA - FTA + DREB + (.5 * OREB) + AST + STL + (.5 * BLK) - PF - TO) / (GmPTS + GmFGM + GmFTM - GmFGA - GmFTA + GmDREB + (.5 * GmOREB) + GmAST + GmSTL + (.5 * GmBLK) - GmPF - GmTO))

The url of the avanced data source is the following:

https://www.nba.com/stats/teams/boxscores-advanced/?Season=2019-20&SeasonType=Regular%20Season&sort=gdate&dir=-1

Steps to clean up the data:

1. Columns “Match Up”, “AST Ratio”, and “Game Date” had a special character in between the words so I replaced that with a “_”.
2. I removed duplicate columns in the two data sources.
3. I checked if there was any null data, and there was none.

Steps to merge traditional and advanced data frames:

1. I added the following columns to prepare for data frames join:
    a. “Home”: home team
    b. “Visitor”: visitor team
    c. “Home_Win”: if home team win
    d. “Team_Game”: Game # for each team
2. Two data frames were ready to join: created a data frame that has both traditional and advanced data

Steps to create features:

1. Created a dictionary to put each team as a key and team’s game stat as values
2. Created 2 set of data:
    a. Rolling data (window = 5) to show last 5 games stat
    b. Season data to show all the games played previously in the season
3. Combined 2 data sets
4. Turned the dictionary into a data frame: each row represented a game and it had home and visitor team’s rolling and season data

Looking at Dallas Mavericks as an example:

	TEAM	MATCH_UP	GAME_DATE	Game#	W/L	Opponent	Home_Game	rolling_PTS	season_PTS	rolling_FGM	season_FGM	rolling_FGA	season_FGA
0	DAL	DAL vs. WAS	2019-10-23	2	1	WAS	1	108.000000	108.000000	35.000000	35.000000	76.000000	76.000000
1	DAL	DAL @ NOP	2019-10-25	3	1	NOP	0	118.000000	115.500000	41.666667	40.000000	87.333333	84.500000
2	DAL	DAL vs. POR	2019-10-27	4	0	POR	1	118.500000	116.666667	39.333333	39.000000	90.666667	87.666667
3	DAL	DAL @ DEN	2019-10-29	5	1	DEN	0	114.700000	114.750000	40.000000	39.500000	88.000000	86.750000
4	DAL	DAL vs. LAL	2019-11-01	6	0	LAL	1	113.133333	113.800000	40.000000	39.600000	92.666667	89.800000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
70	DAL	DAL vs. LAC	2020-08-06	72	0	LAC	1	117.800000	116.690141	38.866667	41.507042	91.733333	90.169014
71	DAL	DAL vs. MIL	2020-08-08	73	1	MIL	1	123.000000	116.958333	41.400000	41.597222	98.933333	90.513889
72	DAL	DAL @ UTA	2020-08-10	74	1	UTA	0	122.000000	117.027397	43.533333	41.684932	94.600000	90.438356
73	DAL	DAL vs. POR	2020-08-11	75	0	POR	1	125.800000	117.216216	43.800000	41.689189	90.266667	90.310811
74	DAL	DAL @ PHX	2020-08-13	76	0	PHX	0	118.866667	117.013333	42.800000	41.653333	89.400000	90.293333

75 rows × 77 columns

Fig 1. Dallas Mavericks Data in each game (2019-2020 season)

Except for informational columns, all the other columns are numerical. I did StandardScaler() on the data and did some data visualization.

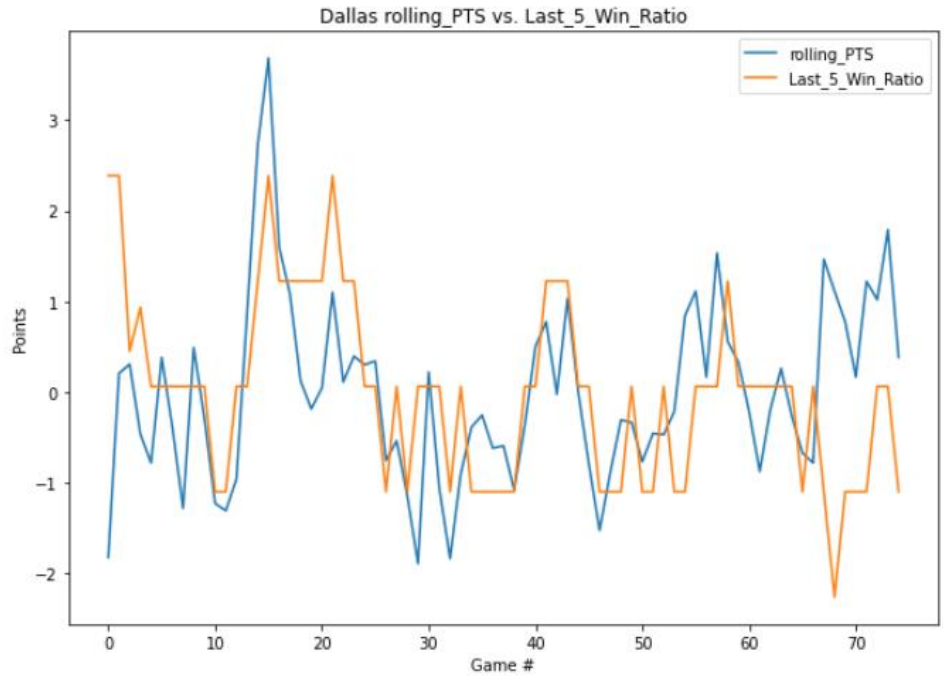

Fig 2. Dallas rolling points vs. last 5 game win ratio

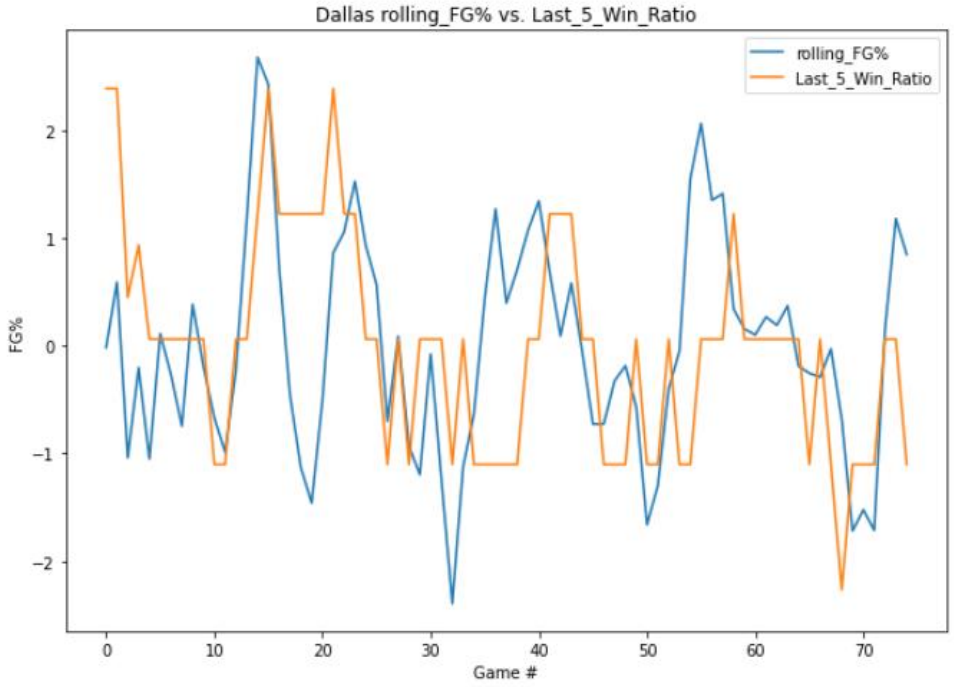

Fig 3. Dallas rolling field goals % vs. last 5 game win ratio

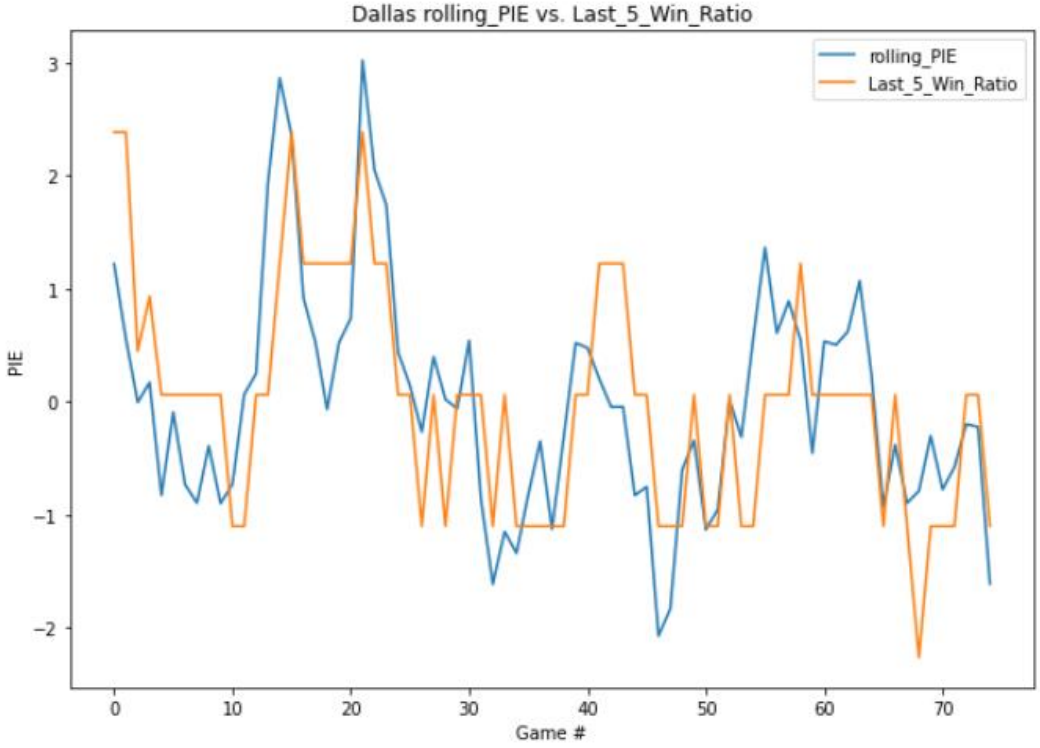

Fig 4. Dallas rolling PIE vs. last 5 game win ratio

Looking at figure 2-4, I picked rolling points, rolling field goals percentage, and rolling ration to plot with last 5 game win ratio. We could see that comparing with the first two feature, rolling PIE seemed to have the same trend as last 5 game win ratio. PIE value was an aggregated value created by NBA and they claimed it can predict game result well. I then started to build models to test if that's true.

4. Model Building

Since home team won 55%, the data is balanced so I picked classification accuracy as my metric to test how good a model is. I selected 6 models to predict home win:
SVC, Random Forest, Logistic Regression, Decision Tree, Gradient Boosting, and Naïve Bayes.

Using the default models and split train and test data as 80% to 20 %, the following was the result:

	accuracy_train	accuracy_test
LogisticRegression	0.715827	0.684211
DecisionTreeClassifier	0.839329	0.555024
RandomForestClassifier	1.0	0.631579
svc	0.854916	0.684211
GradientBoostingClassifier	0.979616	0.617225
NaiveBayes	0.652278	0.62201

Fig 5. Train accuracy and test accuracy of 6 models

SVC, Random Forest, Logistic Regression seemed to perform better. So I picked them to do further hyperparameter tuning.

SVC:

```
# GridSerach to get the best params for SCV model

params = [{'svc__kernel': ['rbf','poly','sigmoid'],
           'svc__gamma': [0.0001, 0.001, 0.01, 0.1],
           'svc__C':[0.1,1,10,100,1000],
           'svc__degree':[0,1,2,3]}]
svc_I = Pipeline([('Scaler', StandardScaler()), ('svc', SVC())])
svc_CV = GridSearchCV(svc_I, params, scoring = 'accuracy', cv=5, verbose = 1, n_jobs=-1)
svc_CV.fit(X_train, y_train)
print(svc_CV.best_params_)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
{'svc__C': 1, 'svc__degree': 0, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
```

Fig 6. SVC_I: fitting using gridsearch and using the best params from the gridsearch result to fit again

```
svc_I = Pipeline([('Scaler', StandardScaler()), ('svc', SVC(C= 1, degree= 0, gamma= 0.01, kernel= 'rbf'))])
svc_I.fit(X_train, y_train)
```

The accuracy rate of the testing was 67.4% comparing with default model at 68.4%.

Then I selected the top 10 important features using permutation importance method to retrain the model to see if there is any improvement:

```
# Now using the top 10 features to retrain the model

svc_new_features = features[sorted_idx][-10:]
X_train_LR = X_train[svc_new_features]
X_test_LR  = X_test[svc_new_features]

params = [{'svc__kernel': ['rbf','poly','sigmoid'],
           'svc__gamma': [0.0001, 0.001, 0.01, 0.1],
           'svc__C':[0.1,1,10,100,1000],
           'svc__degree':[0,1,2,3]}]
svc_II = Pipeline([('Scaler', StandardScaler()), ('svc', SVC())])
svc_CV_II = GridSearchCV(svc_II, params, scoring = 'accuracy', cv=5, verbose = 1, n_jobs=-1)
svc_CV_II.fit(X_train_LR, y_train)
print(svc_CV_II.best_params_)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
{'svc__C': 1, 'svc__degree': 0, 'svc__gamma': 0.01, 'svc__kernel': 'rbf'}
```

```
# Update model with the best params

svc_II = Pipeline([('Scaler', StandardScaler()), ('svc', SVC( C= 1, degree= 0, gamma= 0.01, kernel= 'rbf'))])
svc_II.fit(X_train_LR, y_train)
```

Fig 7. SVC_II: Using top 10 important (permutation important) to retrain the model

The hyperparameters were the same as previous model, and the accuracy rate of was slightly lower at 66.5%. The accuracy has not improved in the last 2 modeling.

Random Forest:

```
# GridSerach to get the best params for Random Forest model

params = {'RF__n_estimators': [100, 300, 1000],
               'RF__max_features': ['auto', 'sqrt'],
               'RF__max_depth': [10, 40, 70, 100],
               'RF__min_samples_split': [2, 5, 10],
               'RF__min_samples_leaf': [6, 8, 10, 12],
               'RF__bootstrap': [True]}
RF_I = Pipeline([('Scaler', StandardScaler()), ('RF', RandomForestClassifier(class_weight='balanced'))])
RF_CV = GridSearchCV(RF_I, params, scoring = 'accuracy', cv=5, verbose =1, n_jobs=-1)
RF_CV.fit(X_train, y_train)
print(RF_CV.best_params_)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
{'RF__bootstrap': True, 'RF__max_depth': 40, 'RF__max_features': 'sqrt', 'RF__min_samples_leaf': 10, 'RF__min_samples_split': 1
0, 'RF__n_estimators': 100}
```

```
# Update model with the best params

RF_I = Pipeline([('Scaler', StandardScaler()),
               ('RF', RandomForestClassifier(bootstrap= 'True', max_depth=40, max_features= 'sqrt', min_samples_leaf=10,
                                             min_samples_split=10, n_estimators= 100, class_weight='balanced'))])
RF_I.fit(X_train, y_train)
```

Fig 8. RF_I fitting using gridsearch and using the best params from the gridsearch result to fit again

The accuracy rate of the testing was 65.1% comparing with default model at 63.2%.

```
# Now using the top 10 features to retrain the model

RF_new_features = features[sorted_idx][-10:]
X_train_RF = X_train[RF_new_features]
X_test_RF  = X_test[RF_new_features]

params = {'RF__n_estimators': [100, 300, 1000],
               'RF__max_features': ['auto', 'sqrt'],
               'RF__max_depth': [10, 40, 70, 100],
               'RF__min_samples_split': [2, 5, 10],
               'RF__min_samples_leaf': [6, 8, 10, 12],
               'RF__bootstrap': [True]}
RF_II = Pipeline([('Scaler', StandardScaler()), ('RF', RandomForestClassifier(class_weight='balanced'))])
RF_CV_II = GridSearchCV(RF_II, params, scoring = 'accuracy', cv=5, verbose =1, n_jobs=-1)
RF_CV_II.fit(X_train_RF, y_train)
print(RF_CV_II.best_params_)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
{'RF__bootstrap': True, 'RF__max_depth': 70, 'RF__max_features': 'auto', 'RF__min_samples_leaf': 8, 'RF__min_samples_split': 5,
'RF__n_estimators': 300}
```

```
# Update model with the best params

RF_II = Pipeline([('Scaler', StandardScaler()),
               ('RF', RandomForestClassifier(bootstrap= 'True', max_depth= 70, max_features= 'auto', min_samples_leaf= 8,
                                             min_samples_split= 5, n_estimators= 300, class_weight= 'balanced'))])
RF_II.fit(X_train_RF, y_train)
```

Fig 9. RF_II: Using top 10 important (permutation important) to retrain the model

The accuracy rate of the testing was 66.0%. The accuracy rate increased slight during in the last 2 modeling.

Logistic Regression:

```
# GridSerach to get the best params for Random Forest model

params = {'LR__penalty' : ['l1','l2'],
          'LR__C' : [100, 10, 1.0, 0.1, 0.01],
          'LR__max_iter' : [100,150,200,300],
          'LR__solver':['liblinear','saga']
         }
LR_I = Pipeline([('Scaler', StandardScaler()), ('LR', LogisticRegression(max_iter=10000))])

LR_CV = GridSearchCV(LR_I, params, scoring = 'accuracy', cv=5, verbose =1, n_jobs=-1)
LR_CV.fit(X_train, y_train)
print(LR_CV.best_params_)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
{'LR__C': 0.1, 'LR__max_iter': 100, 'LR__penalty': 'l1', 'LR__solver': 'liblinear'}
```

```
LR_I = Pipeline([('Scaler', StandardScaler()),
                 ('LR', LogisticRegression(C=0.1,  max_iter= 100, penalty='l1', solver = 'liblinear'))])
LR_I.fit(X_train, y_train)
```

Fig 10. LR_I fitting using gridsearch and using the best params from the gridsearch result to fit again

The accuracy rate of the testing was 66.5% comparing with default model at 68.4%.

```
# Now using the top 10 features to retrain the model

LR_new_features = features[sorted_idx][-10:]
X_train_LR = X_train[LR_new_features]
X_test_LR  = X_test[LR_new_features]

params = {'LR__penalty' : ['l1','l2'],
          'LR__C' : [100, 10, 1.0, 0.1, 0.01],
          'LR__max_iter' : [100,150,200,300],
          'LR__solver':['liblinear','saga']
         }
LR_II = Pipeline([('Scaler', StandardScaler()), ('LR', LogisticRegression(max_iter=10000))])
LR_CV_II = GridSearchCV(LR_II, params, scoring = 'accuracy', cv=5, verbose =1, n_jobs=-1)
LR_CV_II.fit(X_train_LR, y_train)
print(LR_CV_II.best_params_)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
{'LR__C': 0.1, 'LR__max_iter': 100, 'LR__penalty': 'l2', 'LR__solver': 'liblinear'}
```

```
LR_II = Pipeline([('Scaler', StandardScaler()),
                 ('LR', LogisticRegression(C=0.1,  max_iter= 100, penalty='l2', solver = 'liblinear'))])
LR_II.fit(X_train_LR, y_train)
```

Fig 11. LR_II: Using top 10 important (permutation important) to retrain the model

Hyperparameter penalty has changed from L1 to L2. The accuracy rate of the testing was 67.4%. The accuracy rate didn't improve in the last 2 modeling.

Feature importance:

Since the permutation importance is universal available in the 3 models, I selected it to compare between 3 models. As an example, the following figure shows the top 10 important features of logistic regression model (LR_I):

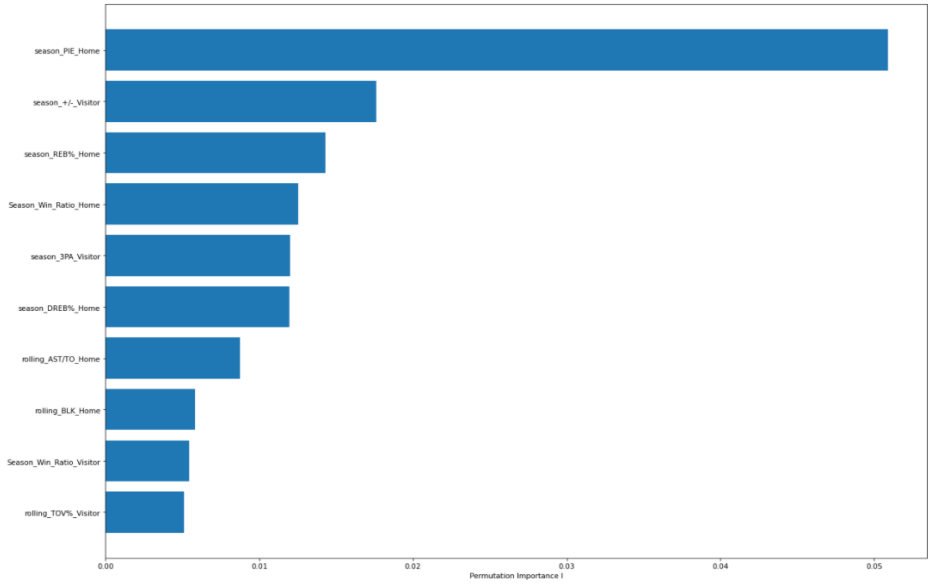

Fig 12. LR_I top 10 important feautres (using permutation important)

We can see that season_PIE_Home was significantly important than other features. However, in random forest, season_PIE_Home was only the number 2 features as the following figure:

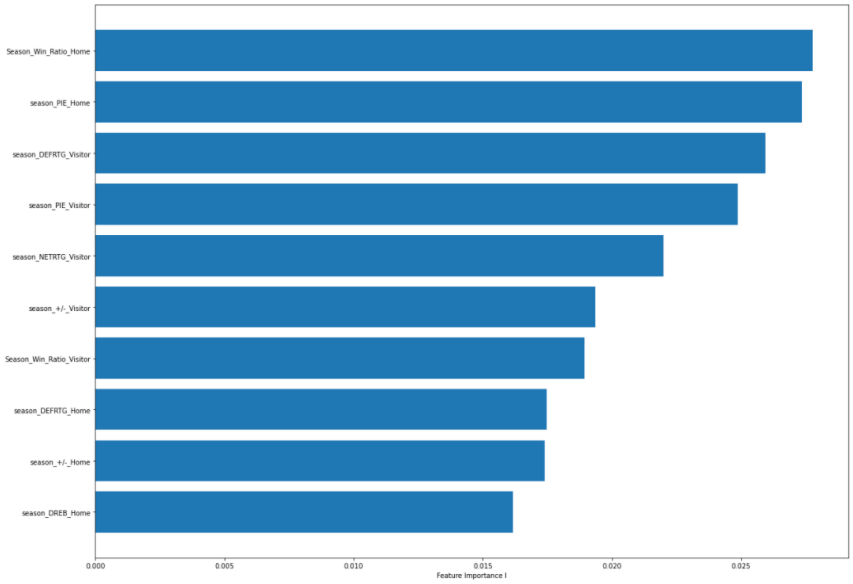

Fig 13. RF_I top 10 important feautres (using permutation important)

svc_I:

	features	permutation_importance
0	season_DREB_Home	0.0204067
1	season_NETRTG_Home	0.0176555
2	season_PIE_Home	0.0174163
3	season_+/-_Home	0.0171292
4	season_REB%_Home	0.0165789
5	rolling_FTA_Home	0.0162919
6	Season_Win_Ratio_Home	0.0156459
7	season_AST_Home	0.0154545
8	rolling_OREB%_Home	0.0152153
9	season_DEFRTG_Home	0.0148804

svc_II:

	features	permutation_importance
0	season_DREB_Home	0.0256699
1	rolling_OREB%_Home	0.0220574
2	Season_Win_Ratio_Home	0.0211483
3	season_PIE_Home	0.0172727
4	season_REB%_Home	0.0144737
5	season_NETRTG_Home	0.0138756
6	season_AST_Home	0.0134689
7	season_+/-_Home	0.0129665
8	season_DEFRTG_Home	0.0100718
9	rolling_FTA_Home	0.00930622

Random_Forest_I

	features	permutation_importance
0	season_PIE_Home	0.0188995
1	Season_Win_Ratio_Home	0.0186124
2	season_DREB_Home	0.0180622
3	season_NETRTG_Home	0.0136842
4	season_+/-_Home	0.0135646
5	rolling_OREB_Home	0.0103589
6	season_DEFRTG_Home	0.00997608
7	season_EFG%_Home	0.00791866
8	season_OFFRTG_Visitor	0.0077512
9	season_3PA_Visitor	0.00772727

Random_Forest_II

	features	permutation_importance
0	Season_Win_Ratio_Home	0.0278947
1	season_OFFRTG_Visitor	0.0236364
2	season_PIE_Home	0.0198565
3	season_DREB_Home	0.0195694
4	season_DEFRTG_Home	0.0192344
5	season_+/-_Home	0.0186124
6	rolling_OREB_Home	0.017512
7	season_EFG%_Home	0.0158134
8	season_3PA_Visitor	0.00964115
9	season_NETRTG_Home	0.00650718

Logistic_Regression_I

	features	permutation_importance
0	season_PIE_Home	0.0509091
1	season_+/-_Visitor	0.0176316
2	season_REB%_Home	0.0142823
3	Season_Win_Ratio_Home	0.0125359
4	season_3PA_Visitor	0.0120096
5	season_DREB%_Home	0.0119378
6	rolling_AST/TO_Home	0.00873206
7	rolling_BLK_Home	0.0058134
8	Season_Win_Ratio_Visitor	0.00545455
9	rolling_TOV%_Visitor	0.00511962

Logistic_Regression_II

	features	permutation_importance
0	season_PIE_Home	0.0666986
1	rolling_TOV%_Visitor	0.0147847
2	Season_Win_Ratio_Home	0.0101675
3	rolling_AST/TO_Home	0.00923445
4	season_+/-_Visitor	0.00892344
5	Season_Win_Ratio_Visitor	0.0077512
6	season_DREB%_Home	0.00732057
7	rolling_BLK_Home	0.00669856
8	season_3PA_Visitor	0.00578947
9	season_REB%_Home	-0.00255981

Fig 14. svc_I, svcII, RF_I, RF_II, LR_I, and LR_II top 10 important feautres (using permutation important)

As you can see in figure 14, I listed the top 10 important features in each model. We can see that in total "home" : "visitor" ratio was 4:1; "season" : "rolling" also have the same ratio.

5. Summary:

As you can see in figure 15, the highest accuracy of my model was at 68.4% comparing with the baseline at 55.0%. Both SVC and logistic regression models tied at that percentage. Hyperparameter tuning didn't help on accuracy improvement in SVC and logistic regression models. Also, selecting more import features to build a new model didn't help either.

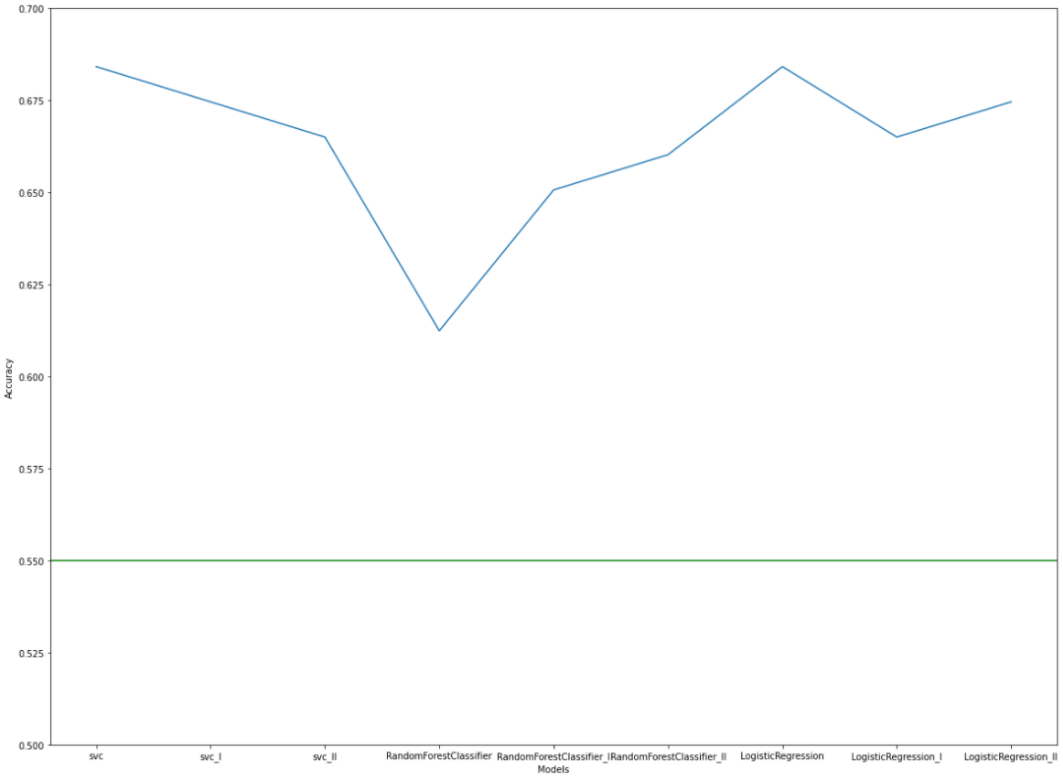

Fig 15: Model Accuracy: The x-label names ended with "_1" represented the result of hyperparameter tuning using grid search method. The x-label names ended wtih "_2" represented the result of fitting with top 10 most important features, selected through the method of permuation importance in "_1".

As for which features are more important, there are three features that worth noticing:

1. **season_PIE_Home**: It is significantly more important than other features in logistic regression model and it is also top 5 in svc and random forest method

2. **season_Win_Ratio_Home:** The most important feature in random forest model and it is also top 7 in 2 other methods.
3. **season_DREB_Home**: The most important feature in svc and it is also top 4 in random forest method.

Furthermore, “home” stats are significantly more important than “visitor” stats, and “season” stats are significantly more important than “rolling” stat.

6. Recommendations for clients to use the findings:

1. Season's stats are more important comparing with rolling stats, so for a team to predict their game result, they should look at the whole season's stats instead of recent stats.
2. Home team's stats are more important than visitor team's stats. And therefore, when a team play a home game, whether they win or not are more dependent on their own stats instead of their opponent's.
3. The best accuracy I could get was still less than 70%, which means historical data still couldn't explain game result more than 30% of the time. Players and coaches should get frustrated knowing the odds are against them because there are still rooms to turn things around. And people who put wager on game should pay closely attention on the first half of the game, they may consider bet on a different team at half time because if players are playing significantly different from their previous games, they are likely to change the game result.

7. Ideas for further studies:

Future studies should look at each player's stat as well. Rooster can vary from game to game due to injury or other reason and that should be put into consideration. And players combination seems to matter as well, different players combination can contribute significantly different depend on whether they have good or bad atmosphere. Furthermore, officials matter sometimes because the way they officiate games may lead to different game result.