Swing and Miss Prediction, and Optimal Pitcher/Pitch Combination

Boston Red Sox

Lina Nguyen

This two part project's focus is to predict which pitches will result in a swing and miss, and the top three optimal pitcher and pitch combination that result in a swing and miss. The low performance of the model was due to technological limitations, although computational efficiency was improved using PCA. Model performance may be improved next time by incoorporating parallel processing, which in turn will allow for better model tuning. Model performance may also improve next time by allocating more time into feature engineering.

This project was approached with different angles before settling on one. Due to technological limitations, dimensionality reduction played a huge role in computational efficiency. PCA was used on all features to reduce the dimensions of features down to 12. At first, I thought that grouping features with high cardinality in the dataset together, running four different pca's, then combining the dataset after would help with performance. I did this so that the features wouldn't lose their importance. I also used undersampling because I knew how computationally intensive this would be on my computer. This resulted in all metrics (accuracy, precision, f-1 score, recall) being around 60% for the best performing model. I then decided to perform PCA on the entire dataset at once, with the explained variance at 95%, and reran my models. This resulted in faster performance of the model but still about the same results in the metrics. I knew that the low performance was because of undersampling, so I decided to try oversampling instead. Although this made my models take a lot longer to run, xgBoost was able to classify swing and miss with around 80% for all metrics. However, I realized that I shouldn't treating the imbalance on my testing set, so I went back to split the data before performing SMOTE. This lowered performance so I also tested undersampling, and not balancing my dataset before classification, but those overall performed worse at classifying the minority group. The model that performed best was random forest, although having a lower accuracy than xgBoost, performed better at classifying Swing and Misses based off of the recall and f1-score. Also, it was a lot less computionally intensive to run compared to xgBoost. I originally wanted to use tpot to find the best algorithm possible, but was computationlly limited.

After selecting the best model, from the limited amount of models I was able to run, I tried to use GridSearchCV to optimize parameters, however, due to computational limitations and time restraints, I was unable to. I decided to run my original model with the entire testing dataset, and then use it to classify the holdout set.

From the holdout set, I identified the top 3 combinations that produced a swing and miss. From there, I ran random forest models to pull feature importances to identify which features played the biggest role in a swing and miss for that partical pitcher and pitch combination.

Import Packages

In [143…
```
import pandas as pd
from pandas_profiling import ProfileReport
import matplotlib.pyplot as plt

# Feature Engineering
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

# modeling
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Model Optimization
from sklearn.model_selection import GridSearchCV

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

In [49]:
```
# import dataset
train = pd.read_csv("Application_Train_01.csv")
hold = pd.read_csv("Application_Holdout_01.csv")
hold1 = pd.read_csv("Application_Holdout_01.csv")
```

```
<ipython-input-49-1546066c737b>:2: DtypeWarning: Columns (41) have mixed types. Specify dtype option on import or set low_
memory=False.
  train = pd.read_csv("Application_Train_01.csv")
```

EDA

In [ ]:
```
profile = ProfileReport(train)
profile.to_file("output.html")
```

Feature Engineering

In [50]:
```
# drop all columns not included in the holdout set, except for PitchResult
train1 = train.drop(['Venue', 'HomeTeam', 'AwayTeam', 'PitcherName', 'BatterID', 'BatterName', 'SwingFL', 'VideoLink'], a

# drop GameDate, GameNumber, PitcherID because too many nunique
# balls - there are values equal to 4, change it to 3
# drop Season
train1 = train1.drop(['GameDate', 'GameNumber', 'PitcherID', 'Season'], axis = 1)

# covert DayNight, TopInning, PitcherHand, and BatterHand into binary
# convert PitchType into numerical representations.
# convert PitchResult to 1 = "Swinging strike", else = 0
train1['DayNight'] = train1["DayNight"].replace("Day", 1)
train1['DayNight'] = train1["DayNight"].replace("Night", 0)

train1['TopInning'] = train1["TopInning"].replace("TOP", 1)
train1['TopInning'] = train1["TopInning"].replace("BOTTOM", 0)

train1['PitcherHand'] = train1["PitcherHand"].replace("R", 1)
train1['PitcherHand'] = train1["PitcherHand"].replace("L", 0)

train1['BatterHand'] = train1["BatterHand"].replace("R", 1)
train1['BatterHand'] = train1["BatterHand"].replace("L", 0)

train1["PitchResult"] = [1 if i == "Swinging strike" else 0 for i in train1["PitchResult"]]

train1["PitchType"] = pd.Series(train1["PitchType"]).replace({'FB' : 0, 'CH': 1, 'SL': 2, 'SI' :3, 'CB': 4, 'SF': 5, 'CF'
```

In [51]:
```
# scale everything but PitchResult
y = train1['PitchResult']
X = train1.drop(['PitchResult'], axis = 1)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(X)
scaled = pd.DataFrame(scaled, columns = X.columns)
```

In [52]:
```
# PCA For all
pca = PCA(n_components = 0.95)
pca.fit(scaled)
reduced = pca.transform(scaled)
```

In [53]:
```
# train test split
X_train, X_test, y_train, y_test = train_test_split(reduced, y, train_size=0.75, test_size=0.25)
```

In [54]:
```
# Oversampling to address class imbalance
sm = SMOTE(random_state=42)
X_train, y_train = sm.fit_resample(X_train, y_train)
```

Modeling

In [55]:
```
rf = RandomForestClassifier(max_depth = 8)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.93      0.55      0.69    286885
           1       0.15      0.65      0.24     35134

    accuracy                           0.56    322019
   macro avg       0.54      0.60      0.46    322019
weighted avg       0.84      0.56      0.64    322019
```

In [20]:
```
xgb = XGBClassifier(max_depth = 12, n_estimators = 200)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
print('Accuracy Score -', accuracy_score(y_test, y_pred_xgb))
print(classification_report(y_test, y_pred_xgb))
```

```
Accuracy Score - 0.6832112390883768
              precision    recall  f1-score   support

           0       0.91      0.72      0.80    286911
           1       0.15      0.41      0.22     35108

    accuracy                           0.68    322019
   macro avg       0.53      0.56      0.51    322019
weighted avg       0.83      0.68      0.74    322019
```

Model Optimization

In [21]:
```
# SMOTE again on entire dataset to train
sm = SMOTE(random_state=42)
X, y = sm.fit_resample(reduced, y)
```

In [ ]:
```
# param_grid = {
#     'n_estimators': [200, 500],
#     'max_features': ['auto', 'sqrt', 'log2'],
#     'max_depth' : [4,5,6,7,8],
#     'criterion' :['gini', 'entropy']
# }

# CV_rfc = GridSearchCV(estimator=rf, param_grid=param_grid, cv= 5)
# CV_rfc.fit(X, y)
# CV_rfc.best_params_
```

In [25]:
```
# rf.fit(X, y)
```

Prediction

In [56]:
```
# feature engineering for holdout set

# drop GameDate, GameNumber, PitcherID because too many nunique
# balls - there are values equal to 4, change it to 3
# drop Season
hold = hold.drop(['GameDate', 'GameNumber', 'PitcherID', 'Season', 'PitcherName'], axis = 1)

# covert DayNight, TopInning, PitcherHand, and BatterHand into binary
# convert PitchType into numerical representations.
# convert PitchResult to 1 = "Swinging strike", else = 0
hold['DayNight'] = hold["DayNight"].replace("Day", 1)
hold['DayNight'] = hold["DayNight"].replace("Night", 0)

hold['TopInning'] = hold["TopInning"].replace("TOP", 1)
hold['TopInning'] = hold["TopInning"].replace("BOTTOM", 0)

hold['PitcherHand'] = hold["PitcherHand"].replace("R", 1)
hold['PitcherHand'] = hold["PitcherHand"].replace("L", 0)

hold['BatterHand'] = hold["BatterHand"].replace("R", 1)
hold['BatterHand'] = hold["BatterHand"].replace("L", 0)

hold["PitchType"] = pd.Series(hold["PitchType"]).replace({'FB' : 0, 'CH': 1, 'SL': 2, 'SI' :3, 'CB': 4, 'SF': 5, 'CF': 6}
```

In [57]:
```
# scale
scaler = MinMaxScaler()
scaled = scaler.fit_transform(hold)
scaled = pd.DataFrame(scaled, columns = hold.columns)

# pca on holdoutset
holdtrans = pca.transform(scaled)
```

In [58]:
```
# predict data
predictions = rf.predict(holdtrans)
```

In [70]:
```
hold1['Swing&Miss'] = predictions
hold1.to_csv("Hold.csv")
```

Pitcher/Pitch Combination

In [71]:
```
swingmiss = hold1[(hold1['Swing&Miss'] == 1)]
```

In [72]:
```
swingmiss
```

Out[72]:

	GameDate	Season	GameNumber	GameSeqNum	DayNight	Inning	TopInning	PAOfInning	PitchOfPA	Balls	...	ReleaseVelocityY	ReleaseVelocity
2	2019-03-28	2019	565220	6	Day	1	TOP	2	2	0	...	-129.968994	-4.79439
6	2019-03-28	2019	565220	10	Day	1	TOP	3	2	1	...	-130.542999	-1.19275
7	2019-03-28	2019	565220	11	Day	1	TOP	3	3	1	...	-129.990005	1.20233
11	2019-03-28	2019	565220	15	Day	1	TOP	4	2	0	...	-129.802994	-1.04965
12	2019-03-28	2019	565220	16	Day	1	TOP	4	3	0	...	-130.445999	-3.00325
...	...	...	...	...	...	...	...	...	...	...	...	...	...
71055	2019-09-29	2019	567343	171	Day	6	BOTTOM	1	6	2	...	-137.272995	-3.27201
71061	2019-09-29	2019	567343	181	Day	6	BOTTOM	3	2	0	...	-136.574005	-1.70711
71062	2019-09-29	2019	567343	182	Day	6	BOTTOM	3	3	1	...	-124.529999	0.20238
71063	2019-09-29	2019	567343	183	Day	6	BOTTOM	3	4	1	...	-138.149994	-2.18091
71064	2019-09-29	2019	567343	184	Day	6	BOTTOM	4	1	0	...	-118.365997	2.97070

26412 rows × 35 columns

The top three pitcher-pitch combinations that result in a swing and miss are:

In [80]:
```
combo = swingmiss.groupby(['PitcherName','PitchType']).size().reset_index().rename(columns={0:'count'})
combo.sort_values('count', ascending=False).head(3)
```

Out[80]:

	PitcherName	PitchType	count
90	Pitcher 352040	SL	854
205	Pitcher 371811	CH	671
132	Pitcher 363739	CB	643

In [144…
```
# Breaking datasets into respective pitcher/pitch combo, where pitcher/pitch = 1, and else = 0
PitcherSL = swingmiss[(swingmiss['PitcherName'] == "Pitcher 352040") & (swingmiss['PitchType'] == "SL")]
swingmiss1 = swingmiss.drop(PitcherSL.index)
swingmiss1['Result'] = 0
PitcherSL['Result'] = 1
PitcherSL = pd.concat([swingmiss1, PitcherSL])

PitcherCH = swingmiss[(swingmiss['PitcherName'] == "Pitcher 371811") & (swingmiss['PitchType'] == "CH")]
swingmiss1 = swingmiss.drop(PitcherCH.index)
swingmiss1['Result'] = 0
PitcherCH['Result'] = 1
PitcherCH = pd.concat([swingmiss1, PitcherCH])

PitcherCB = swingmiss[(swingmiss['PitcherName'] == "Pitcher 363739") & (swingmiss['PitchType'] == "CB")]
swingmiss1 = swingmiss.drop(PitcherCB.index)
swingmiss1['Result'] = 0
PitcherCB['Result'] = 1
PitcherCB = pd.concat([swingmiss1, PitcherCB])
```

In [127…
```
# Pitcher 352040 SL

# drop GameDate, GameNumber, PitcherID because too many nunique
# drop PitchType, PitcherName
# balls - there are values equal to 4, change it to 3
# drop Season
PitcherSL = PitcherSL.drop(['GameDate', 'GameNumber', 'PitcherID', 'Season', 'PitchType', 'PitcherName'], axis = 1)

# covert DayNight, TopInning, PitcherHand, and BatterHand into binary
# convert PitchType into numerical representations.
# convert PitchResult to 1 = "Swinging strike", else = 0
PitcherSL['DayNight'] = PitcherSL["DayNight"].replace("Day", 1)
PitcherSL['DayNight'] = PitcherSL["DayNight"].replace("Night", 0)

PitcherSL['TopInning'] = PitcherSL["TopInning"].replace("TOP", 1)
PitcherSL['TopInning'] = PitcherSL["TopInning"].replace("BOTTOM", 0)

PitcherSL['PitcherHand'] = PitcherSL["PitcherHand"].replace("R", 1)
PitcherSL['PitcherHand'] = PitcherSL["PitcherHand"].replace("L", 0)

PitcherSL['BatterHand'] = PitcherSL["BatterHand"].replace("R", 1)
PitcherSL['BatterHand'] = PitcherSL["BatterHand"].replace("L", 0)

# scale everything but PitchResult
y = PitcherSL['Result']
X = PitcherSL.drop(['Result'], axis = 1)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(X)
scaled = pd.DataFrame(scaled, columns = X.columns)

# Oversampling to address class imbalance
sm = SMOTE(random_state=42)
X,y = sm.fit_resample(scaled, y)

rf = RandomForestClassifier(max_depth = 8)
rf.fit(X, y)
print(classification_report(y_test, y_pred_rf))
```

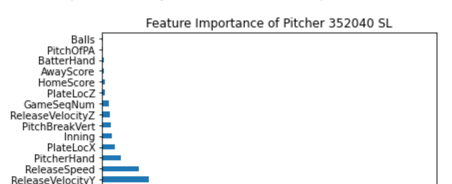
```
              precision    recall  f1-score   support

           0       0.93      0.55      0.69    286885
           1       0.15      0.65      0.24     35134

    accuracy                           0.56    322019
   macro avg       0.54      0.60      0.46    322019
weighted avg       0.84      0.56      0.64    322019
```

In [147…
```
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh', title = 'Feature Importance of Pitcher 352040 SL')
```

Out[147…
```
<AxesSubplot:title={'center':'Feature Importance of Pitcher 352040 SL'}>
```



Pitcher 352040 SL produces a swing and miss, and its effectiveness is at least partially driven by its ReleaseLocX.

In [135…
```
# Pitcher 371811 CH

# drop GameDate, GameNumber, PitcherID because too many nunique
# drop PitchType, PitcherName
# balls - there are values equal to 4, change it to 3
# drop Season
PitcherCH = PitcherCH.drop(['GameDate', 'GameNumber', 'PitcherID', 'Season', 'PitchType', 'PitcherName'], axis = 1)

# covert DayNight, TopInning, PitcherHand, and BatterHand into binary
# convert PitchType into numerical representations.
# convert PitchResult to 1 = "Swinging strike", else = 0
PitcherCH['DayNight'] = PitcherCH["DayNight"].replace("Day", 1)
PitcherCH['DayNight'] = PitcherCH["DayNight"].replace("Night", 0)

PitcherCH['TopInning'] = PitcherCH["TopInning"].replace("TOP", 1)
PitcherCH['TopInning'] = PitcherCH["TopInning"].replace("BOTTOM", 0)

PitcherCH['PitcherHand'] = PitcherCH["PitcherHand"].replace("R", 1)
PitcherCH['PitcherHand'] = PitcherCH["PitcherHand"].replace("L", 0)

PitcherCH['BatterHand'] = PitcherCH["BatterHand"].replace("R", 1)
PitcherCH['BatterHand'] = PitcherCH["BatterHand"].replace("L", 0)

# scale everything but PitchResult
y = PitcherCH['Result']
X = PitcherCH.drop(['Result'], axis = 1)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(X)
scaled = pd.DataFrame(scaled, columns = X.columns)

# Oversampling to address class imbalance
sm = SMOTE(random_state=42)
X,y = sm.fit_resample(scaled, y)

rf = RandomForestClassifier(max_depth = 8)
rf.fit(X, y)
print(classification_report(y_test, y_pred_rf))
```

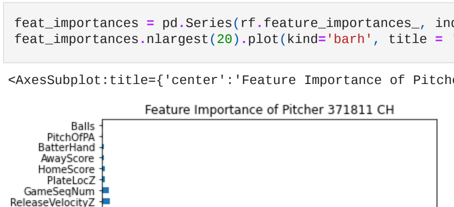
```
              precision    recall  f1-score   support

           0       0.93      0.55      0.69    286885
           1       0.15      0.65      0.24     35134

    accuracy                           0.56    322019
   macro avg       0.54      0.60      0.46    322019
weighted avg       0.84      0.56      0.64    322019
```

In [146…
```
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh', title = 'Feature Importance of Pitcher 371811 CH')
```

Out[146…
```
<AxesSubplot:title={'center':'Feature Importance of Pitcher 371811 CH'}>
```



Pitcher 371811's CH produces a swing and miss, and its effectiveness is at least partially driven by its ReleaseLocZ.

In [140…
```
# Pitcher 363739 CB

# drop GameDate, GameNumber, PitcherID because too many nunique
# drop PitchType, PitcherName
# balls - there are values equal to 4, change it to 3
# drop Season
PitcherCB = PitcherCH.drop(['GameDate', 'GameNumber', 'PitcherID', 'Season', 'PitchType', 'PitcherName'], axis = 1)

# covert DayNight, TopInning, PitcherHand, and BatterHand into binary
# convert PitchType into numerical representations.
# convert PitchResult to 1 = "Swinging strike", else = 0
PitcherCB['DayNight'] = PitcherCB["DayNight"].replace("Day", 1)
PitcherCB['DayNight'] = PitcherCB["DayNight"].replace("Night", 0)

PitcherCB['TopInning'] = PitcherCB["TopInning"].replace("TOP", 1)
PitcherCB['TopInning'] = PitcherCB["TopInning"].replace("BOTTOM", 0)

PitcherCB['PitcherHand'] = PitcherCB["PitcherHand"].replace("R", 1)
PitcherCB['PitcherHand'] = PitcherCB["PitcherHand"].replace("L", 0)

PitcherCB['BatterHand'] = PitcherCB["BatterHand"].replace("R", 1)
PitcherCB['BatterHand'] = PitcherCB["BatterHand"].replace("L", 0)

# scale everything but PitchResult
y = PitcherCB['Result']
X = PitcherCB.drop(['Result'], axis = 1)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(X)
scaled = pd.DataFrame(scaled, columns = X.columns)

# Oversampling to address class imbalance
sm = SMOTE(random_state=42)
X,y = sm.fit_resample(scaled, y)

rf = RandomForestClassifier(max_depth = 8)
rf.fit(X, y)
print(classification_report(y_test, y_pred_rf))
```

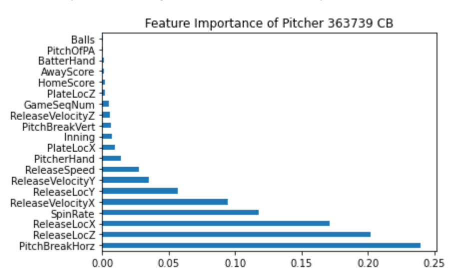
```
              precision    recall  f1-score   support

           0       0.93      0.55      0.69    286885
           1       0.15      0.65      0.24     35134

    accuracy                           0.56    322019
   macro avg       0.54      0.60      0.46    322019
weighted avg       0.84      0.56      0.64    322019
```

In [145…
```
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh', title = 'Feature Importance of Pitcher 363739 CB')
```

Out[145…
```
<AxesSubplot:title={'center':'Feature Importance of Pitcher 363739 CB'}>
```



Pitcher 363739 CB's produces a swing and miss, and its effectiveness is at least partially driven by its PitchBreakHorz.