Predicting Pitch Type Based on Situational Inputs Using Random Forest Classification Model

Originally Published: 10/05/2022
Author: Roman Smith

Introduction

The primary mechanics of the game of baseball involve a sequence of at-bats, with each at bat consisting of a pitcher from one team throwing a sequence of pitches to a batter from the other team. To make it difficult for batters to predict where a pitch will cross the plate and how fast it will travel, pitchers throw different types of pitches. Thus, while it is illegal to steal the signs that pitchers and catchers use to communicate which type of pitch they will throw, it would be advantageous to an offense to be able to anticipate which type of pitch will be thrown.

This investigation aims to develop a statistical model to predict the type of the next pitch to be thrown in any scenario. A random forest classification model will be trained on historical pitching data and then the accuracy of the model will be calculated and compared to the accuracy of the baseline prediction strategy to evaluate its performance.

Problem Definition

The objective of this investigation is to answer the following question:

> *What predictive power does a random forest classification model have for the type of a pitch, given a specific pitcher and series of situational inputs, compared to a baseline?*

Model

The model used will be the *random forest classifier* from the Python package *sklearn*. A random forest model is a feature-based, supervised machine learning model that trains labeled, tabular data to predict the class of a target feature, using the values of other predictor features. A random forest classifier is an ensemble method that learns multiple independent decision tree classifiers based on random subsets of training data, and then makes its final class prediction the class predicted by the most decision trees in the forest.

Two models will be generated in this investigation. The first will be an "out-of-the-box" sklearn random forest classifier that uses the default hyperparameter values to build the forest. The second will be a random forest classifier with optimized hyperparameter values found through the process of cross-fold validation.

Inputs

The input for the model is a tabular data set for an individual pitcher where each data point is a single pitch, consisting of the following predictor features (with the set of possible values):

- Batter stands right {0,1}
- Batter stands left {0,1}
- Pitcher throws right {0,1}
- Pitcher throws left {0,1}
- Balls {0,1,2,3}
- Strikes {0,1,2}
- 3rd base occupied {0,1}
- 2nd base occupied {0,1}
- 1st base occupied {0,1}
- Outs {0,1,2}
- Pitch number of at-bat {an integer > 0}

And the following target feature:

- Pitch type {AB, AS, CH, CU, EP, FC, FF, FO, FS, FT, GY, IN, KC, KN, NP, PO, SC, SI, SL, UN}

Descriptions for these pitch types can be found here:
(https://www.daktronics.com/en-us/support/kb/DD3312647)

Once the model has been trained, the input for testing will be a data set only with the predictor features.

For this investigation, the data set is pitch-level data from Adam Wainwright in the 2022 MLB regular season.

Outputs

A trained model given input data of predictor features will output a predicted class membership (pitch type).

The output of the overall investigation will be a model that predicts a class in the target feature given values for the input features.

Procedure

The complete source code for the experiment can be found on my GitHub (https://github.com/roman-smith/pitch_prediction)

1: Necessary imports

```
from pybaseball import cache, statcast, playerid_lookup
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
```

2: Get data from pybaseball.statcast*

```
data = statcast(start_dt='2022-04-07', end_dt='2022-10-05')
```

*pybaseball must be installed locally from (https://pypi.org/project/pybaseball/)

3. Select Pitcher

```
    LAST_NAME = 'Wainwright'
    FIRST_NAME = 'Adam'
    pitcher_id = playerid_lookup(last=LAST_NAME, first=FIRST_NAME).loc[0,
'key_mlbam']
```

Pitch arsenals are unique to each pitcher, so predicting pitches must be done for individual pitchers as opposed to the entire league.

4. Filter data by selected pitcher

```
    data = data[data['pitcher'] == pitcher_id]
```

5. Subset the data into desired predictor and target data

```
    target = ['pitch_type']
    features = ['stand', 'p_throws', 'balls', 'strikes', 'on_3b', 'on_2b',
'on_1b', 'outs_when_up', 'pitch_number']

    y = data[target]
    x = data[features]
```

6. Transform baserunner data

```
    x['on_3b'] = x['on_3b'].apply(lambda x: 0 if pd.isna(x) else 1)
    x['on_2b'] = x['on_2b'].apply(lambda x: 0 if pd.isna(x) else 1)
    x['on_1b'] = x['on_1b'].apply(lambda x: 0 if pd.isna(x) else 1)
```

pybaseball.statcast returns base occupation data in the form of the player id that occupies the base, or nothing if no player occupies the base. The random forest classifier requires numeric data for every instance of all predictor features, and will assume that all values are ordinal instead of categorical. Thus, to provide the baserunner features as input to a classifier, the occupation status of the base needs to be numerically identified with a 1 or 0 respectively.

7. One-hot encode *stand* and *p_throw* data

```
    # Generate one-hot encoder for 'stand' and 'p_throws'
    transformer = make_column_transformer(
        (OneHotEncoder(), ['stand', 'p_throws']),
        remainder='passthrough'
    )

    # Fit transformer to data
    transformed = transformer.fit_transform(x)

    # Reconstruct the input data set using the transformed data and column
names
    x = pd.DataFrame(
        transformed,
        columns=transformer.get_feature_names_out()
    )
```

The *stand* (batter handedness) and *p_throw* (pitcher handedness) fields have categorical data in the set {'R','L'}. The random forest classifier requires numeric data for all instances of all feature predictors, and will assume that all values are ordinal instead of categorical. Thus, the features need to be "one-hot" encoded by splitting *stand* and *p_throw* into two columns each, *stand_R* and *stand_L* and *p_throw_R* and *p_throw_L* each with a 1 if the original column was a 'R' or 'L' respectively and 0 if it was the opposite.

The transformer is specified as one-hot encoding and for the two specified fields, fitted to the data set, and then the data set is reformed with the new fields.

8. Split data into training and testing sets

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=0)
```

The test-size is set to 0.25 which is a standard value for random forest classifiers. The random state is set to 0 for reproducibility.

9. Calculate the baseline accuracy

```
df = y.value_counts(normalize=True).to_frame('counts')
highest_proportion = df['counts'].max()
```

The baseline accuracy is the proportion of the entire data set represented by the most frequently occurring class in the target feature. Predicting membership of this class for every input requires no further analysis and thus gives a baseline accuracy to measure the accuracy of the models against.

10. Train the random forest with default values

```
clf_def = RandomForestClassifier(random_state=0)
clf_def.fit(x_train, y_train.values.ravel())
```

Random state is set to 0 for reproducibility.

11. Test values and calculate accuracy

```
pred_def = clf_def.predict(x_test)
acc_def = accuracy_score(y_test, pred_def)
```

12. Generate random grid for cross-fold validation

```
random_grid = {
    'n_estimators': [int(x) for x in np.linspace(start = 100, stop = 
2000, num = 20)],
    'max_depth': [int(x) for x in np.linspace(5, 50, num = 10)],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [2, 4, 8],
    'bootstrap': [True, False]
}
```

Now that the “out-of-the-box” model has been trained and tested, the optimized model can be trained. The keys of this dictionary represent the most important hyperparameters to be optimized, and their values are the possible values to be passed to the model. The values are somewhat arbitrary, but represent common values for their respective hyperparameters in random forest models.

13. Fit the cross-fold validator to the training data

```
    clf_opt = RandomForestClassifier()
    clfs_opt = RandomizedSearchCV(estimator=clf_opt,
param_distributions=random_grid, n_iter=200, cv=4, random_state=0,
n_jobs=-1)
    clfs_opt.fit(x_train, y_train.values.ravel())
    clf_opt = clfs_opt.best_estimator_
```

A default random forest classifier is passed to the *RandomizedSearchCV()* function from sklearn. The number of iterations is 200 and the number of folds is 4, so the function trains 200 different random forest models, each trained 4 times, and each with a random combination of possible random forest hyperparameters. From each of the 200 models, the cross-validation function will identify which is the most accurate, which is saved for testing. The iteration number and fold number could be increased for a greater number of total forests generated and thus a higher probability of a more accurate model, but this would become very costly in terms of computing time and power. Random state is set to 0 for reproducibility, but the random nature of hyperparameter combination makes it inherently unreproducible. However, due to the high number of iterations, the difference in accuracy between trials would be small.

14. Test values and calculate accuracy

```
    pred_opt = clf_opt.predict(x_test)
    acc_opt = accuracy_score(y_test, pred_opt)
```

Results

In the investigation, three predictive models were measured

1. The baseline strategy of predicting the most common pitch each time
2. The random forest classifier with default hyperparameters
3. The random forest classifier with hyperparameters optimized by cross-fold validation

The accuracy of a model is defined by the proportion of times the correct class is predicted. There are other common metrics of evaluation for feature-based supervised models like random forest classifiers such as precision, recall, and F1 score that account for differences in performance across different classes. However, the assumption is made that each class is equally important in terms of precision and recall, so the comprehensive accuracy is a fine metric to evaluate a model.

The following are the accuracies of the three predictors:

Baseline:

Accuracy: 0.30324909747292417

Default hyperparameters:

Accuracy: 0.33727034120734906

Percent increase over baseline: 11%

Optimized hyperparameters:

Accuracy: 0.3766404199475066

Percent increase over baseline: 24%

Percent increase over default parameters: 12%

Conclusion

The objective of the investigation was to determine what predictive power a random forest classification model has over a baseline to predict the classification of a pitch given the pitcher and a series of situational variables.

As seen in the Results section, both the random forest classifier with the default hyperparameters and the classifier with the optimized hyperparameters offer improvements (11% and 24% respectively) over the baseline strategy of predicting the most common pitch each time. Thus, it is concluded that a random forest classification model offers a small improvement over a baseline strategy.

In modern-day baseball, however, the standard approach to batting is not to simply guess what the next pitch will be, but to watch the pitcher's delivery and to be ready to react to the speed and movement of the pitch. Thus, if guessing the next pitch with approximately 30% accuracy is not the preferred method, an increase to 37% accuracy is not likely to make a significant difference in overall hitting approach. In this sense, though using a random forest classifier to predict pitches is more accurate than the baseline, it is not a feasible strategy.

Evaluation and Improvements

As neither of the random forest classifiers trained in the investigation produced significantly improved accuracy over the baseline, the results of the investigation were not as productive as was hoped. However, this does not necessarily mean that it is not possible to predict pitches to a productive degree using statistical modeling.

For future models, the following possible improvements could be tested:

1. Evaluate the model on multiple different pitchers to evaluate the accuracy of the random forest classifier across the league as a whole
2. Apply different classification models and compare their accuracies to the random forest classifier. Some potential models to try include:
    a. Support vector machine (SVM) classifier
    b. K-nearest-neighbors classifier
    c. Naive Bayes classifier
3. Include more predictor features. Specifically, a major shortcoming of the data used in this investigation is that it uses only situational data points such as the current count, handedness of the pitcher and batter, etc., and does not account for the types of pitches previously thrown in the at-bat. Incorporating such pitch sequence data could likely lead to a much more accurate model, however, sequence-based data does not lend well to feature-based models, so a different paradigm of modeling such as a neural network would likely be required.