Prediction of Scoring Innings in Baseball

Springboard Data Science Intensive Capstone Project

Kevin Wang

September 2017

Table of Contents

Introduction

Sports analytics is a common and well-known application of data science. The prevalence of data analysis and statistics in baseball stands out. The main goal of sports is winning, and the front offices of many teams employ analytics to increase their team's chances of doing so. For instance, analytics can help project prospects in the minors and identify trends in player performance.

A common complaint of casual baseball fans is that the games are boring. Often in baseball, there are long periods of little action and excitement. A good measure of excitement for casual fans is run scoring; batting is typically considered more interesting than fielding (offense vs. defense). Perhaps if these casual fans could determine which innings are more likely to feature scoring, they would get more interested in baseball.

In this project, the goal is to create a model to predict whether a half-inning will feature run scoring based only on information available at the beginning of the inning. Potential clients include fans who have limited time to watch games, casual fans, and advertising companies interested in seeing when the most exciting parts of the game occur for maximum viewership. In addition, baseball teams and managers may be interested in predicting the likelihood of scoring, and thus making changes to improve their chances of scoring and thus increase their chances of winning.

Data Story

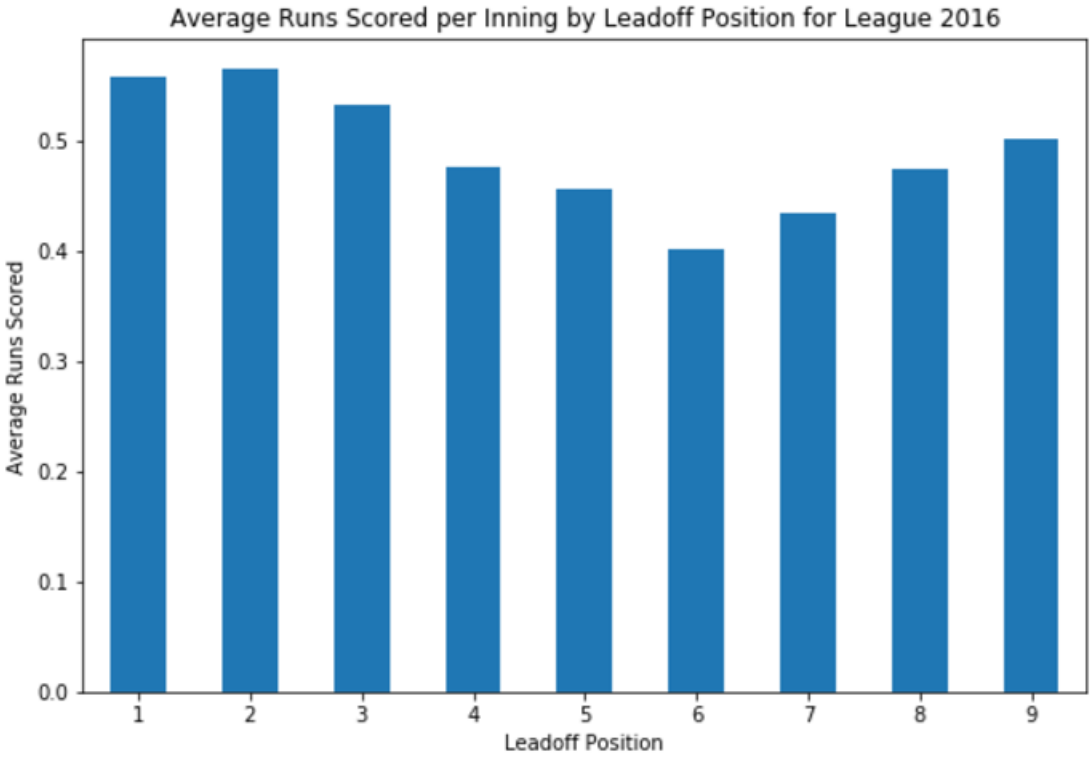
Play-by-play data for every game and every season is available through Retrosheet.org. The available data includes type of hit, out, every player on the field, positions, etc.

The play-by-play data is available in event file format from Retrosheet.org. The downloads may be parsed into .csv files using programs they provide. An alternative is to use R and a set of files known as Chadwick to parse a season's play-by-play data into a single .csv file, as detailed at http://isaacmiller.co/how-to-create-a-single-season-csv-of-retrosheet-play-by-play-data-pc/.

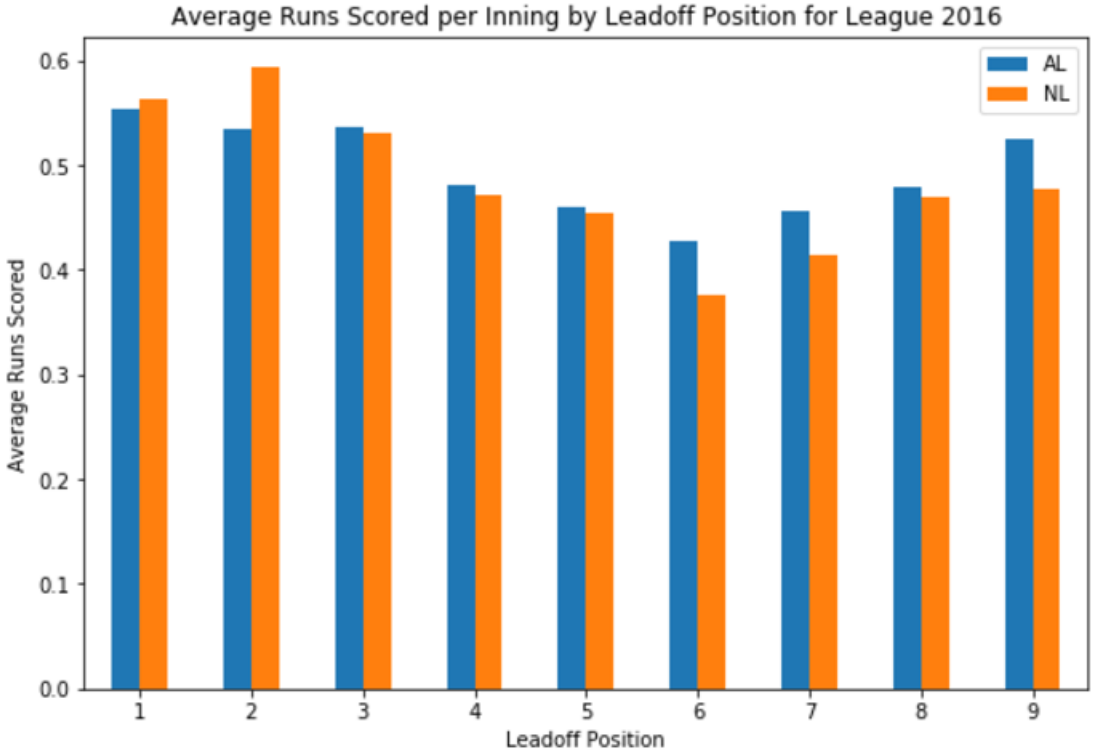
Since there are over 90 fields available, a subset of features is selected to make the data easier to work with.

One field that could be of interest in predicting whether there will be scoring in an inning is the batting order position of the batter leading off. Managers often place their best hitters in the 3-5 spots, with the worst hitters in the 8th position in the lineup. However, the order players come up to bat is not the same for every inning; only the 1st inning is guaranteed to have batters come up in the original 1-9 order. Do innings where the listed leadoff hitter actually leads off contribute to more runs and a better chance at winning? Is it possible to predict the number of runs scored in an inning based on features such as batting position leading off, inning, etc.? A prospective viewer may, depending on what kind of hitter is leading off, pay more attention in innings with higher scoring-chance.

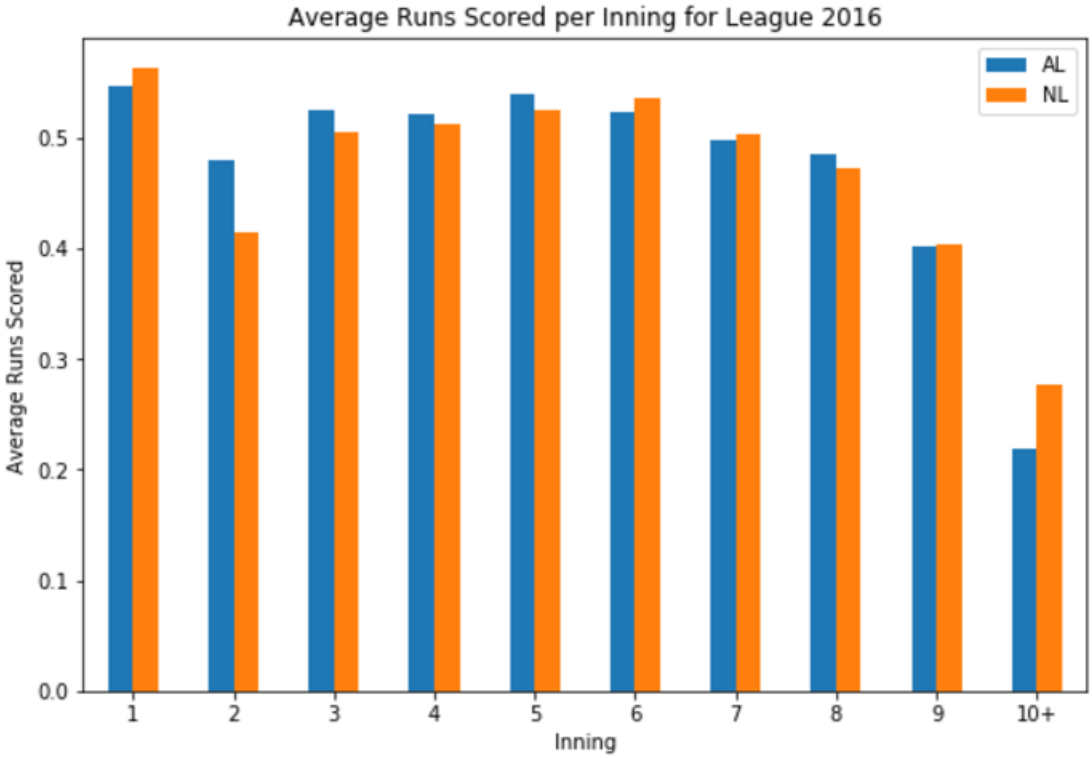
The following plot shows the average runs scored per inning depending on which order position is leading off for the 2016 MLB season:



Since the American League (AL) uses the designated hitter (DH) in place of the poor-hitting pitchers while the National League (NL) does not, it may be beneficial to consider the league of the team batting:



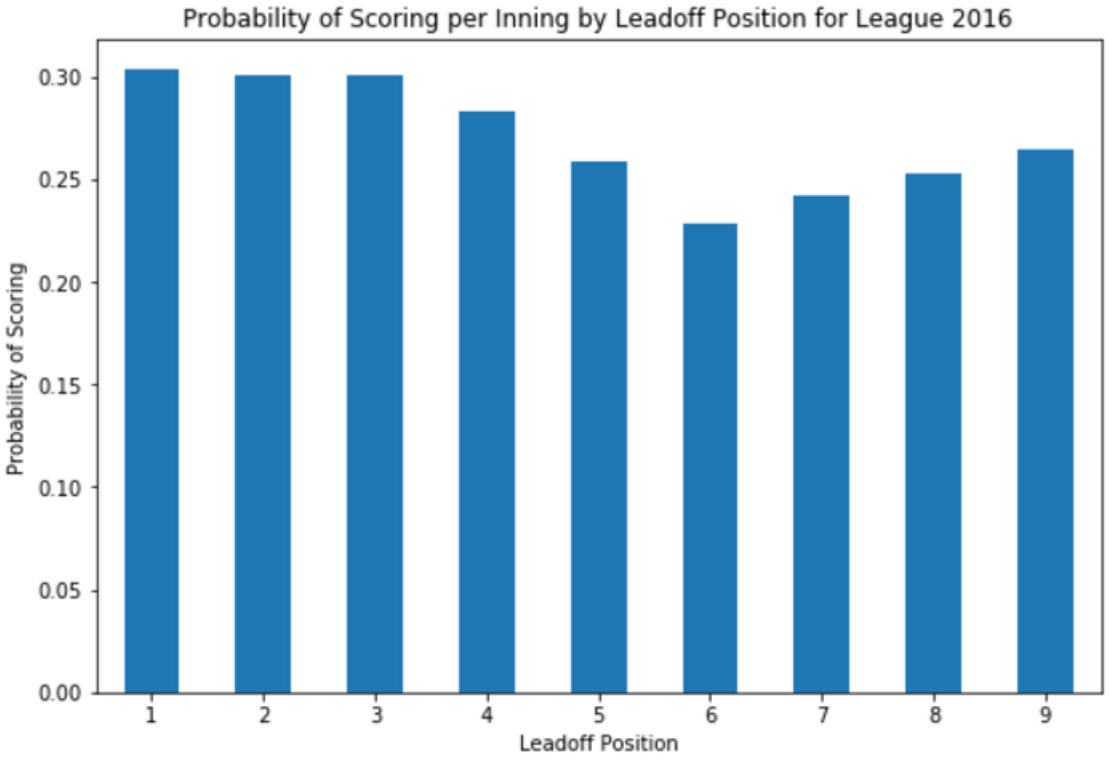
Another potential interesting feature is the inning number itself. For the purposes of the following plot, extra innings are grouped into the same bin in order to make a clearer plot:

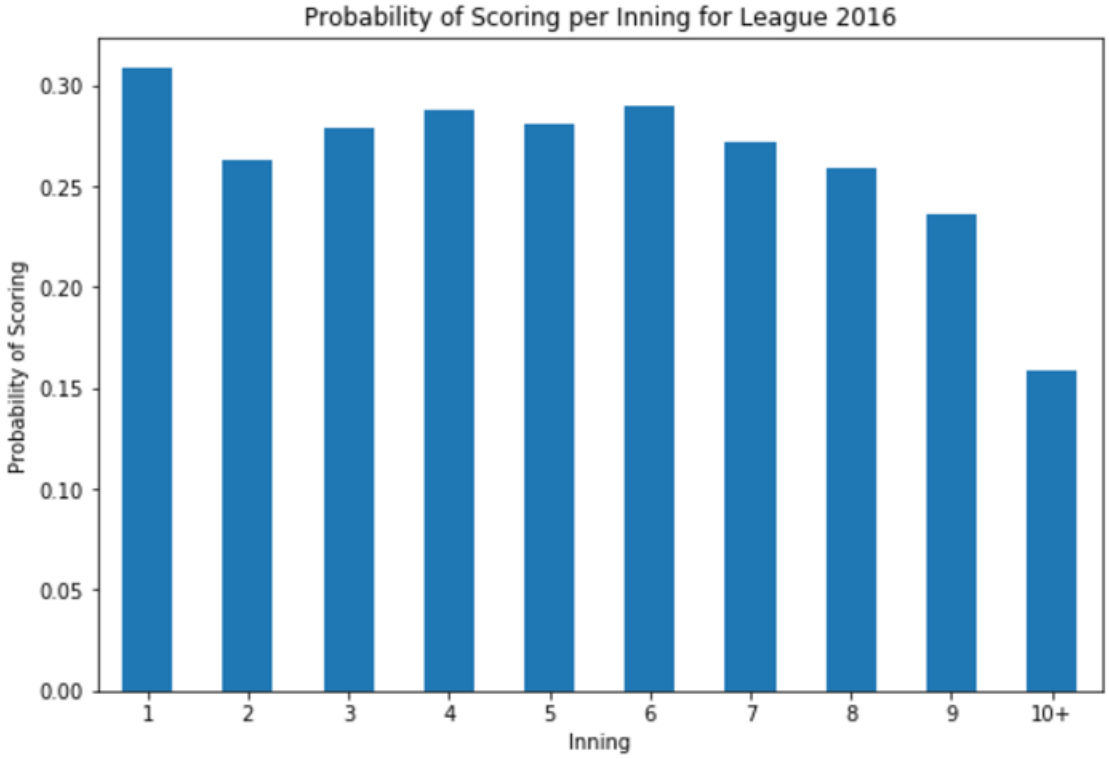


From the visualizations, for hits per inning, there is a slight drop for the second inning before going up a little into the mid-innings, then trending down from there. The same could be said for runs per inning. Unsurprisingly, the presence of the DH allowed the AL teams to have slightly better offensive numbers.

When considering leadoff positions, there was a similar trend in that starting with the top of the order led to more runs, and leading off with the 6th spot led to fewer runs. Number of hits also followed this trend, although the effect is less obvious. Surprisingly, NL teams did not average the fewest amount of runs for when the 9 hitter, usually the pitcher, leads off. Even if the first batter is almost a free out, the top of the lineup is still able to produce runs better than if the bottom of the order was due up.

Another value to plot is the probability of actually scoring runs dependent on the leadoff position or inning. This value will later be used as the Y value for the models.





From these plots, it is fairly clear that the trends are similar to the average runs scored per inning plots from before.

The dataset does not provide detailed player data. For instance, there is no data available that references player batting average, which could easily help predict the likelihood of scoring depending on who will be coming up to bat in a particular inning. There are also game logs available on Retrosheet. This data can provide additional information on games that is typically fixed for each game, such as whether the game was a day or night game, who the umpires were, etc. The game log data could be used to obtain several additional features that can be considered for use in the modeling.

Preliminary Modeling

An initial attempt at a model requires first cleaning the data such that the rows are all innings instead of plays. The rows of the play-by-play data for the whole 2016 season are iterated through to count how many runs and hits that occurred in each inning. The counts are taken and then a True/False flag is generated for each inning to specify whether any runs are scored. Each inning thus has various features that are known at the beginning of the inning, along with the dependent variables of number of runs/hits that occurred and whether any runs/hits occurred. For the modeling, the problem is presented as a binary classification problem of either True/False for the "Runs scored?" flag.

Logistic Regression

The first model considered for use is Logistic Regression with the Y[i] value being either True/False, depending on whether runs scored for a particular inning "i." The dependent variable is thus binary. The initial features considered are inning, leadoff position, and score differential, which is how far ahead or behind the team batting is when their half of the inning to bat starts.

Since the inning number and the leadoff position are integers yet categorical variables, the model is first tested while encoding the variables using the One Hot Encoder. These two features are encoded and combined with the score differential to generate a model to predict whether or not runs are scored in a particular inning. This generates an accuracy of around 0.72. Then, the inning and leadoff position are allowed to remain as numerical variables, and the model is generated again. The accuracy is shown to not change much.

The accuracy of 0.72 seems decent, but when examining precision and recall, the scores are both 0. Taking a look at the confusion matrix, reproduced below, shows the problem:

Runs scored?	False	True
False	7904	2961
True	0	0

The model as is currently predicts every inning to be scoreless when it is fit to the test data portion of the train-test split.

Sklearn's GridSearchCV method is used to determine the best value for the regularization parameter C in an attempt to improve the model; the best C is found to be 0.001. However, the resulting accuracy, precision, and recall remain the same as before.

Random Forest

Due to the problems with using Logistic Regression, we decide to test an alternative modeling algorithm, the Random Forest Classifier. The algorithm is run on the same training and test data split as before, with the same features. The results for the test data are compared to Logistic Regression:

	Accuracy	Precision	Recall
Logistic Regression	0.727473538886	0.0	0.0
Random Forest	0.716244822826	0.031746031746	0.303225806452

Also, here is the confusion matrix for Random Forest:

Runs scored?	False	True
False	7700	2888
True	204	73

The Random Forest model shows that there are now predictions for True in the model, an improvement over Logistic Regression. However, the precision and accuracy scores are still low, so more tuning is needed.

Additional Features

For now, we will stick to using Logistic Regression while different parameters are tuned. The goal is to generate a model where precision and recall are not obscenely low. Then, once an appropriate setup is decided, more algorithms such as Random Forest can be explored and considered.

First, some additional features are added that specify which teams are playing. It is likely that considering which team is batting/pitching will have a noticeable effect on the model. The 2 team features are both encoded using the Label Encoder, and then using the One Hot system. The results are shown below for Logistic Regression with tuned C:

	Accuracy	Precision	Recall
Training	0.725961833466	0.0	0.0
Test	0.727473538886	0.0	0.0

The numbers are still not great. One factor that has yet to be considered is the fact that the data is slightly imbalanced, as most of the innings were scoreless. Therefore, there were many more “False” flags than “True” flags. Some methods to rectify this include stratification, oversampling, and undersampling.

Imbalanced Learning with Logistic Regression

Stratification

We continue to use Logistic Regression as an initial model. The first new factor to consider is stratification during the train-test split. The split’s random state is fixed, but this time stratify is set to be used in the computations.

	Accuracy	Precision	Recall
Training	0. 726329999386	0.0	0.0
Test	0.726369075012	0.0	0.0

It appears stratification did not improve the model much either. Thus, the next step is to consider using undersampling and oversampling from the imblearn package and comparing the results.

Oversampling and Undersampling

First, oversampling is used alongside stratification on the Logistic Regression model. The results are shown below along with the confusion matrix for the test data:

	Accuracy	Precision	Recall
Training	0.533274197977	0.475880713019	0.537576943265
Test	0.528986884623	0.465788139888	0.533217290397

Runs scored?	False	True
False	4673	4216
True	3218	3676

The results clearly show improvement in precision and recall, but accuracy has decreased. There doesn't appear to be any pattern; the model is basically randomly assigning T/F to the data points.

Next, we consider using undersampling instead:

	Accuracy	Precision	Recall
Training	0.534895453781	0.418096199125	0.545494441194
Test	0.528838069615	0.41492938803	0.537456445993

Runs scored?	False	True
False	1911	1740
True	1062	1234

Once again, the model shows similar results. However, precision seems noticeably lower than in the case of oversampling. Thus, we decide to prioritize testing using oversampling first.

Classifier Testing

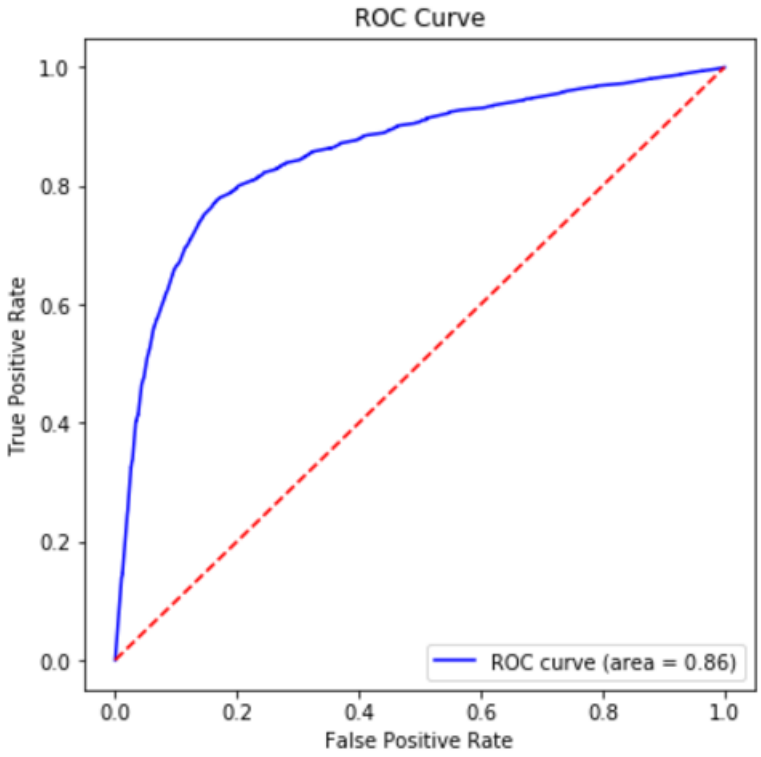
It is apparent that Logistic Regression does not seem to represent the data well. The only thing that may eventually increase the performance is by adding more and more features. Instead, we now move on to considering other modeling algorithms.

Random Forests

After considering Logistic Regression, we now move on to consider Random Forest Classifiers while also drawing upon knowledge gained while exploring the logistic regression models. We keep the extra features, encoding, stratification, and oversampling from before in hopes of attaining a better model. GridSearchCV is called to tune the parameters of n_estimators, max_features, and

min_samples_leaf. The results are shown in the following table (OOB score is used instead of training accuracy/precision/recall). Also shown is a plot of the ROC curve with the AUC score included.

OOB Score	Test Accuracy	Test Precision	Test Recall
0.768400599801	0.778812646518	0.839077546883	0.7488408911



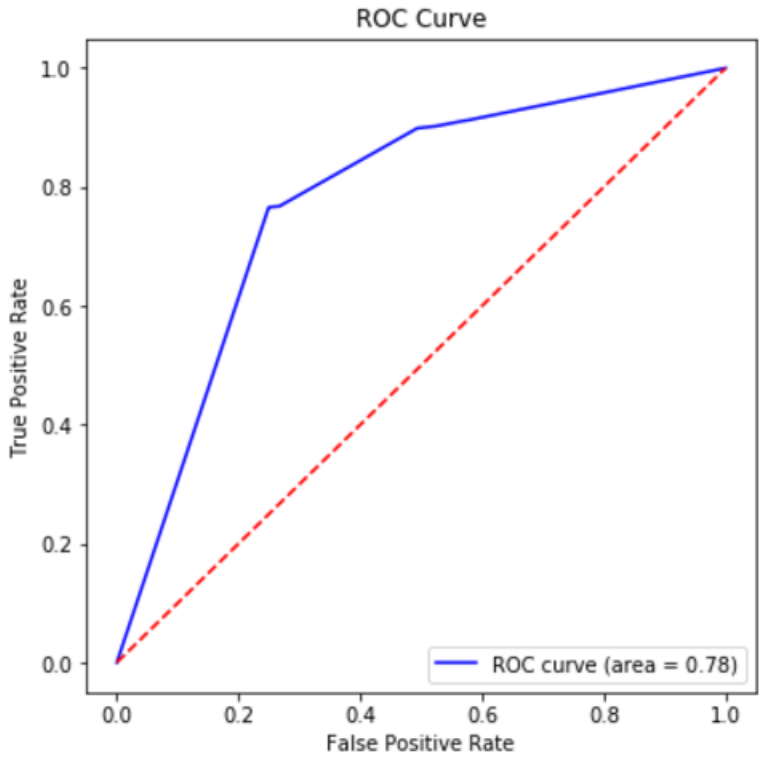
Now that we have obtained a decent model for the prediction of whether there will be scoring during an inning, we can now consider some other models as well.

K-neighbors Classifier

The next model we consider is the K-Neighbors classifier. The parameters are tuned using GridSearchCV. We note that this algorithm takes much longer to process than both logistic regression and random forests. The value for K selected by grid search turns out to be 2. The results are shown below in the table:

	Accuracy	Precision	Recall
Training	0.920990939619	0.875897609192	0.962718789173
Test	0.750364316036	0.767359351242	0.742156862745

There appears to be evidence of some overfitting, since the training statistics ended up being much higher than the test statistics. A plot of the ROC curve shows that the AUC is 0.78 and rises much slower than in the Random Forest case.



Support Vector Machines

We next consider support vector machines (SVMs). The first algorithm that we use from scikit-learn is SVC. While using SVC, parameter tuning had extremely wrong runtime, so max_iter had to be set to a value such that the algorithm terminated in a reasonable amount of time. However, the algorithm did not converge. Increasing max_iter seemed to improve results with the optimal parameters found from before: while running GridSearchCV with 1,000 as max iterations, the optimal accuracy score was roughly 0.52. Once the optimal parameters were taken, the max iterations was increased to 10,000, resulting in the more detailed score table shown below:

	Accuracy	**Precision**	**Recall**
Training	0.769055312678	0.765143195066	0.771169483588
Test	0.637458024457	0.681829700963	0.626280260708

Thus, we conclude that although there may be potential in using SVC, the runtime of the algorithm severely handicaps finding optimal parameters and getting a clear picture of the algorithm's performance due to hardware limitations.

Since the default kernel used by the SVC method is "rbf," we can also consider linear SVMs. For this purpose we consider LinearSVC. The algorithm performs much faster than SVC, but as shown in the results table, the performance is subpar. This may be because the default max iterations parameter is not set to be unbounded.

	Accuracy	**Precision**	**Recall**
Training	0.53749815202	0.53007518797	0.538052566136
Test	0.526832668061	0.511657374557	0.527705175118

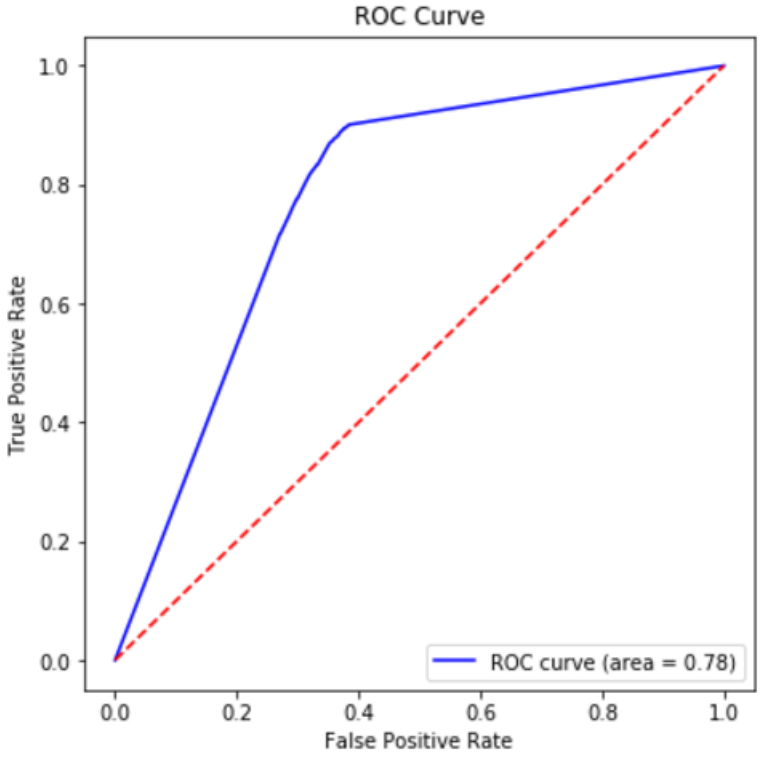
No ROC-AUC curve is shown for the SVC methods since the SVC method did not technically finish processing due to hardware limitations, and LinearSVC was already showing poor results.

Decision Tree Classifier

The final algorithm we will consider for now is the Decision Tree Classifier. After parameter tuning, it turns out the default implementation parameters provide the best score. The results are shown below:

	Accuracy	Precision	Recall
Training	0.942490865699	0.949607163977	0.936279205364
Test	0.750554393968	0.834262544349	0.714642353197

Clearly there is some overfitting, as the training statistics are very high.



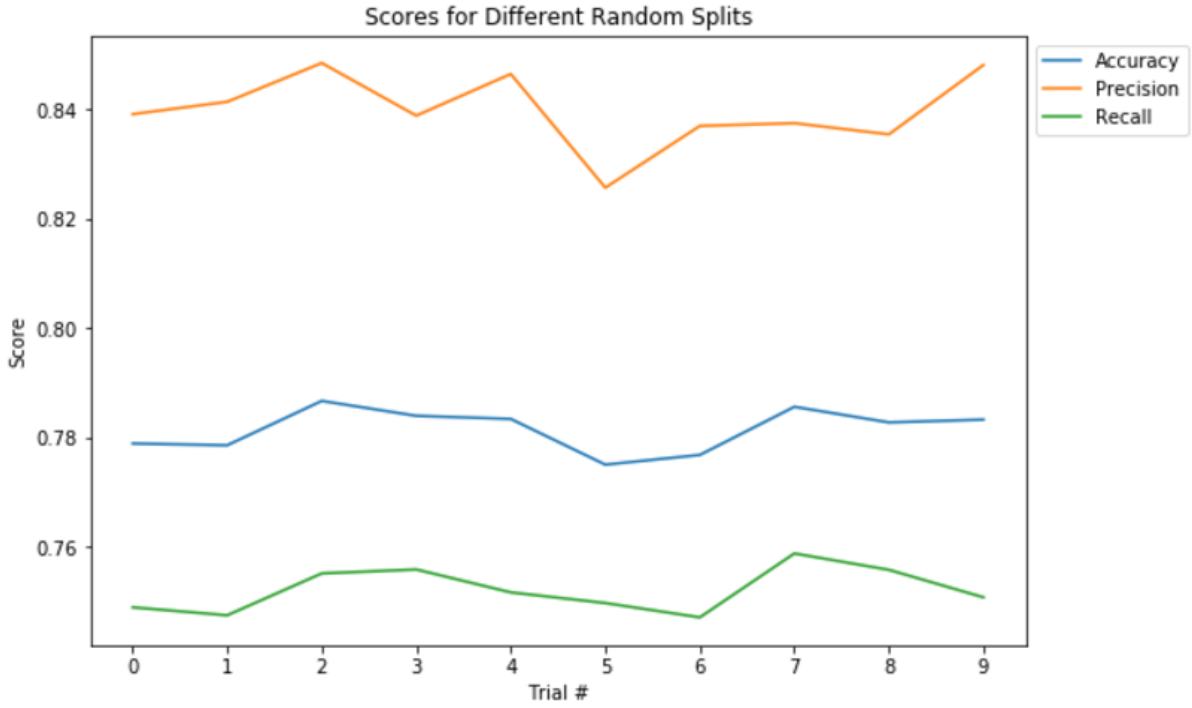
The ROC-AUC curve shows that the fit is decent, but not as good as the random forest model.

Final Model

From the analysis in the previous section, we decide to select the Random Forest Classifier model as the "best" model, due to its good performance in the scoring metrics as well as algorithm runtime.

One additional variable to consider is the fact that the random state of the kernel could heavily influence the model. First, the mere nature of the "Random" Forests obviously depends on randomness. In addition, the train-test splitting done at the beginning of each modeling iteration could easily vary the performance depending on how the data is split into training and test sets. To address this concern, we

run the optimal Random Forest model several times with different random states, and plot the resulting scoring metrics below:



We see that even though the training/test split random seeds were varied, the resulting score metrics remained stable. Thus, we remain confident in the ability of the final random forest model to model the data and predict whether any runs will be scored in a particular half-inning.

Future Approach

Despite our findings and decision on a model, there can still be improvements made to the modeling. More models can be considered, and more parameter tuning can be completed. More features can be added to the model for consideration as well. These features can include data that was not available in the data set, such as stats of the player batting, time of game, etc.

If one desires to expand upon the modeling by considering the players batting/pitching at the beginning of an inning, one must consider the fact that the data would become very sparse. Perhaps more seasons of data would need to be added, and better hardware would need to be considered. There were already some issues with hardware and processing speed in this project; the SVM testing seemed to show some promise, but the extremely long computation times due to hardware limitations meant that the best possible results from the algorithm were not able to be obtained for now.

Conclusion and Recommendations

Based on the findings in this project, we conclude that is possible to predict with roughly 75% accuracy whether or not there will be any scoring in a half-inning based only on information available at the beginning of the half-inning. The model that performed the best under the scoring metrics was the Random Forest Classifier, which achieved >75% accuracy, precision, and recall while having the highest AUC for the ROC curve.

We can recommend potential clients to take note of the inning, leadoff position, teams playing, and score differential at the beginning of an inning, and input those points into the model to get a prediction of whether there will be scoring in the inning. The model can provide an estimate of the scoring, which could help advertisers and broadcasters from determining when to air ads for maximum viewership. Fans can decide whether they want to watch a particular half inning if they are only interested in watching innings involving scoring.

Any prospective analysts interested in further improving the model can attempt to include more features into the model, or consider a wider variety of modeling algorithms. A new definition of an "interesting" inning could be determined; for instance, instead of the dependent variable being a flag that specifies whether any scoring occurred, the dependent variable could be changed to a flag indicating a certain number of hits, whether there were homeruns, or a certain amount of batters came to the plate. Numeric dependent variables could be considered as well, such as number of runs scored in an inning. This leads the way to consideration of regression models in addition to classification models. However, better hardware is recommended, as the processing time for some the models was already quite long, and including more features would only increase that time even further.

Overall, we have found that the Random Forest Classifier model can provide a quick, rough estimate of whether there will be any scoring in an inning given only some features that are available when the inning starts. The model performed relatively well when considering various scoring metrics, but can definitely be improved with future work.