MLB Stolen Base Predictions

Colin Macy, Teagen Williams, Sam Handel,
Waleed Almousa, Thomas Huspeni

Introduction

Questions:

- Can we predict the outcome of a stolen base attempt?
- What factors influence the success of a stolen base attempt the most?

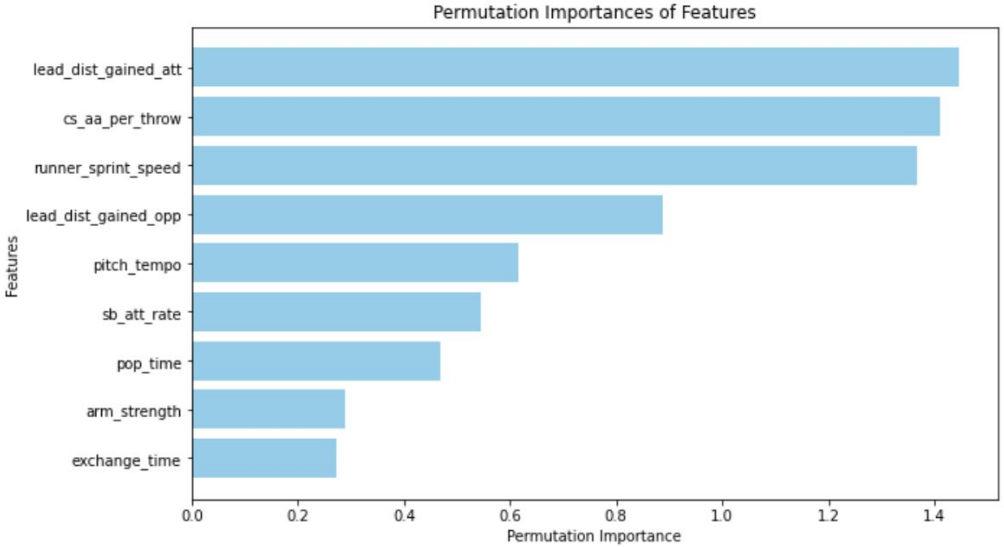
Dataset

- Custom dataset from a combination of MLB statcast resources.
- Includes data from the catcher throwing, pitch tempo, sprint speed, and pitcher running game leaderboards.
- Retrieved stolen base attempts with API call to baseballsavant.mlb.com.

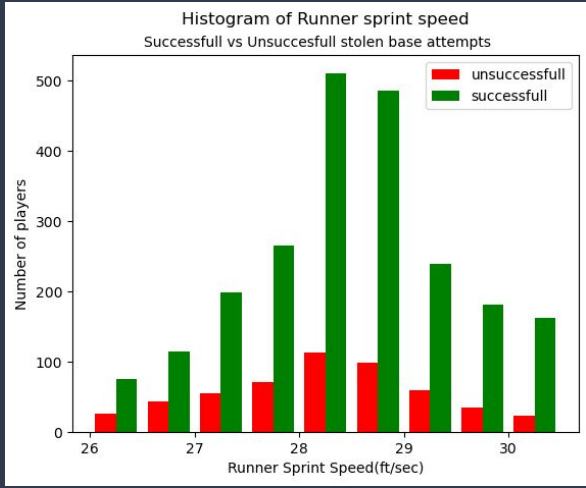
successful_sb	runner_sprint_speed	pitch_tempo	sb_att_rate	lead_dist_gained_opp	pop_time	exchange_time	arm_strength	cs_aa_per_throw
1	28.0	17.433	0.023847	1.847397	1.988417	0.644583	78.32061	0.175065
0	27.8	17.433	0.023847	1.847397	1.988417	0.644583	78.32061	0.175065
1	29.2	17.433	0.023847	1.847397	1.988417	0.644583	78.32061	0.175065
1	30.4	15.792	0.014737	1.715638	1.988417	0.644583	78.32061	0.175065
1	27.1	17.923	0.012097	3.617790	1.988417	0.644583	78.32061	0.175065

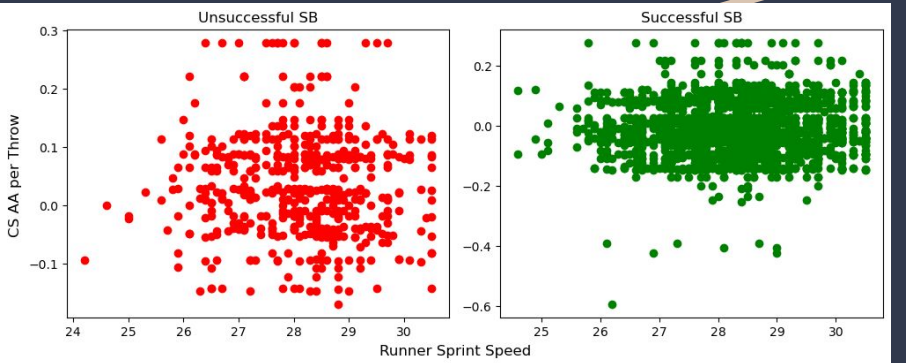
Feature Choice

- Started with nine potential variables that could be used to predict.
- Used permutation importance to narrow it down to four: runner_sprint_speed, lead_dist_gained_opp, lead_dist_gained_att, and cs_aa_per_throw.



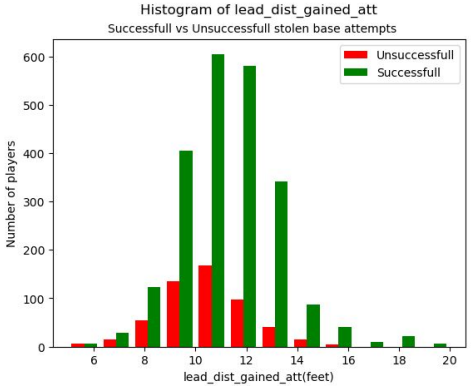
Exploratory Plots

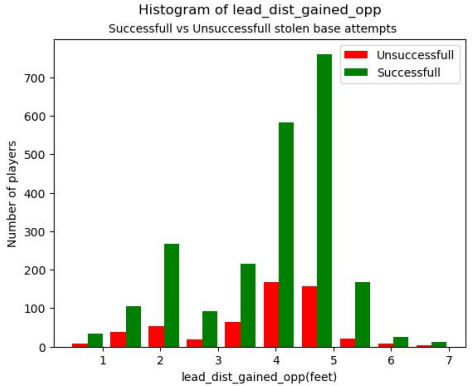




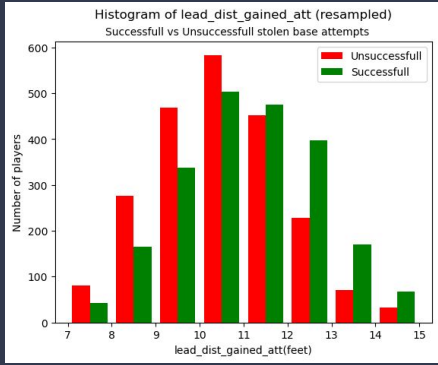
The Dataset is imbalanced

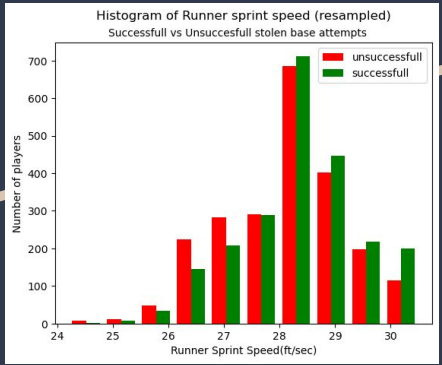
- 81% of observations were successes
- Resampling/data manipulation Was needed



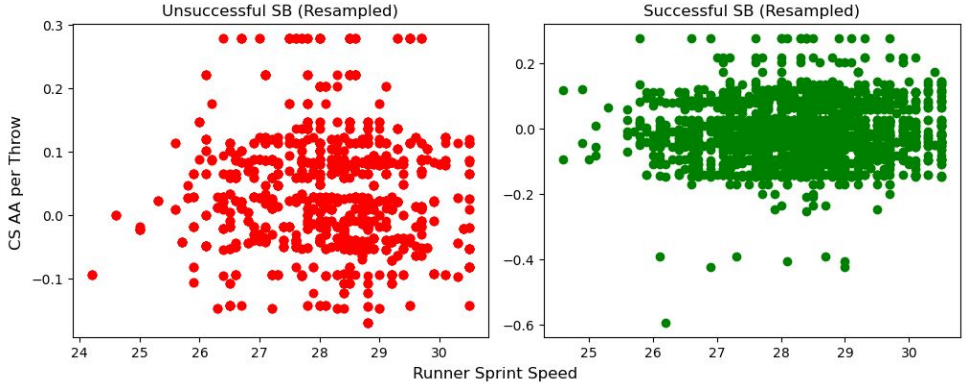


Over/Undersampling

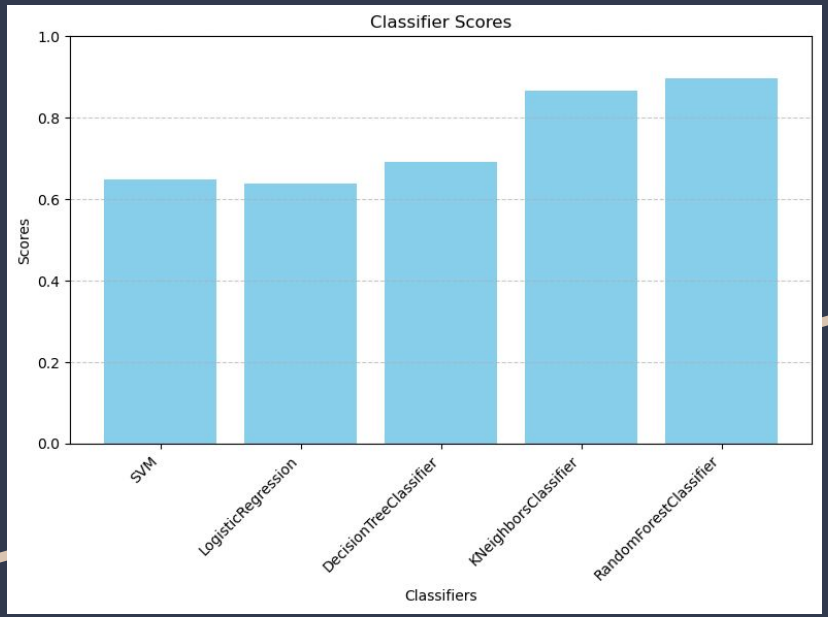




- Using this data set, any model can predict 1 for every observation and get an 81% accuracy score
  - This model would be useless!
- Oversampling solved this by creating more “fake” observations that had an outcome of 0
- Our models that used oversampling had a better accuracy score than undersampling
- Undersampling would require for 1000+ observations to be removed, leaving the model with significantly less data to be trained on



Model Selection



SVC('C': 1, 'Kernel': 'Linear') scored .648

LogisticRegression('C': 1, 'max_iter': 5000) scored .639

DecisionTreeClassifier('criterion': 'Entropy', 'max_depth': 7) scored .692

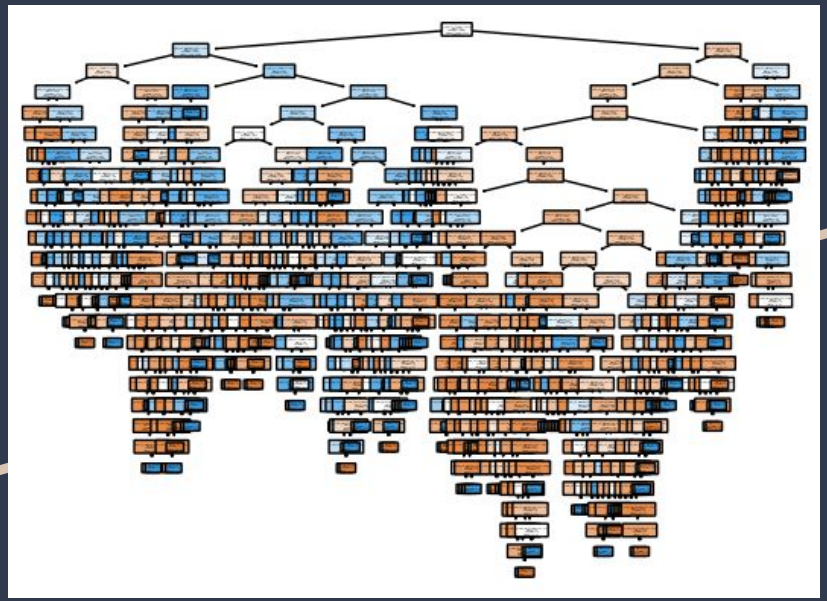
KNeighborsClassifier('metric': 'euclidean', 'n_neighbors': 1) scored .867

RandomForestClassifier('max_depth': 100, 'n_estimators': 100) scored .896

- We tried a range of different models with a range of different hyperparameters to get the best fits
- The first three performed worse than just guessing a successful stolen base (.80)
- KNeighborsClassifier and RandomForestClassifier performed better than guessing a successful stolen base (.80)

From this training and results on our validation data, we decided to use RandomForestClassifier() for the best overall performance in predicting MLB stolen bases.

Random Forest Classifier



Random forest works better than a decision tree as it reduces overfitting and is subject to less noise. The only way to confirm such is through obtaining the accuracy of our test data.

- Upon testing we received a score of .**88 or 88%**
- This is **7.8%** more accurate than the success rate of stolen bases throughout the year.

Based on these results we can confirm that our model works and we can efficiently predict stolen base success rate to a better extent than just guessing.

Conclusion

Obtain Dataset

- Compiled data set from many different MLB statistical resources

Feature Choice and Oversampling

- Narrowed down features with permutation importance
- Data was oversampled so we had to find a model that could predict better than always predicting success

Random Forest model predicts at **.896**

- MLB teams could use this model to decide when they should have runners steal bases
- This would enable teams that use this to have an advantage over other teams in terms of stolen base successes