CS 4641: Machine Learning
Final Project

Friday, December 6, 2019

Introduction

The data set chosen is statistics from all MLB games from 2015 to 2018. Statistics have always been a big part of baseball, and it is a unique sport in that there are more isolated one on one matchups between players than any other sport. The most pivotal matchup is between the pitcher and the batter. The pitcher controls the runs allowed and is usually the most important 'defensive' player. A pitcher has many tools in his repertoire which can help limit the amount of runs a batter will score on his team. The pitcher can throw either a strike or a ball which can serve to either 'ace' the batter or fool him in swinging for a ball outside of the strike zone. In order to pitch the best game, a pitcher will usually try to appear as random as possible or make moves unanticipated by the batter. In this way it almost becomes a game or rock, paper, scissors, where the players almost try to out think each other and go in recursive loops of thought, i.e. "if he's thinking this, then I must do that, unless of course he's thinking I'm thinking that". Decisions whether to throw a ball or strike are also dependent on how many balls and strikes are up on the board and the general layout of the game: players on base, points on the board, innings, outs, etc. From these realizations pitching only carries the façade of randomness, and in reality, it should follow certain patterns with specific circumstances considered. The probabilities of a certain event happening shrink or grow corresponding to new beliefs or events occurring: a Bayesian view.

This is why for the project I thought it'd be apt to try my hand at detecting patterns. It is especially important to me as a lifelong Braves fan as I just watched the team choke in the world series against the Cardinals in typical Atlanta sports fashion. The Braves gave up 10 runs within the first inning which is a record-breaking stat for a series game, and maybe with further analysis of pitching patterns a feat like this will be avoided next season. The features chosen for this project are pretty straight forward. From the data given, I only looked at variables that could be given before a pitch occurred: score, ball count, strike count, outs, pitch number, and a binary variable indicating which bases have players on. If given more time and more resources, I would look to factor in specific player stats in matchups and previous pitching data specific to the current matchup. Since the aim of this project is to test the application of specific algorithms to a dataset vs general feature engineering, I decided against the laborious work in trying to scrape other features together. However, depending on the results of the model it may be something I would explore after finals.

The main objective of the model is to classify if a ball or a strike will occur from the next pitch given the set of features mentioned above. The three models implemented are Random Forests, Support Vector Machines, and Neural Networks. The performance of the model will simply be measured by the classification accuracy it had with the testing data. Further analysis such as precision and recall will be considered as well as how the model performs in general with inferential statistics. Factors such as overfitting, reproducibility, and scalability will also be considered in comparing the efficacies of each model.

Algorithms

There are three algorithms used for the models: Random Forests, SVMs, and Neural Networks. Below are general descriptions of how each algorithm works and how it applies specifically to the data at hand. Implementation of all three algorithms are from the scikit-learn module in python. Complexity will only be explained to the parameter that is most relevant.

Random Forests

The Random Forests algorithm is made up of an ensemble of decisions trees. The individual trees are generated with random cuts and decision boundaries that vote on a classification output. The algorithm tallies up the votes and puts forward the most popular prediction. This is a very powerful algorithm and it utilizes the old adage of "the wisdom of crowds". The low correlation between the models makes the algorithm able to outperform individual trees as the "trees protect each other from their individual errors." [1]

Hyperparameters: [2]

- Number of Estimators- the number of trees in the forrest model; biggest contributor to complexity of the hypothesis class as all features below are a subset of this parameter
- Max Features- number of features to consider when looking for the best split
- Max Depth- maximum depth of each tree
- Min Samples Split- minimum number of samples required to split a leaf node
- Min Samples Leaf- minimum number of required to be at a leaf node
- Bootstrap- weather bootstrap samples are used when building trees (essentially bagging: randomly sampling data for the trees vs using the whole data)

AdaBoost Classifier

AdaBoost is a variation of the Random Forests algorithm and can also be known as boosting. While Random Forests generates a set number of trees with random splits at certain decision boundaries, boosting trains the trees in a sequential way where each tree learns from the mistakes of the prior. In this way, you can have a pseudo-random model which still gets feedback and direction from faults from previous trees.[3]

Hyperparameters: [4]

- Base Estimator- the base estimator from which the ensemble starts; again, deals with complexity of hypothesis class the most directly
- Number of Estimators- maximum number of estimators at which boosting is terminated
- Learning Rate- shrinks the contribution of each classifier
- Algorithm- option to use the SAMME or SAMME.R algorithm
- Random State- seed used by the random number generator

[1] https://towardsdatascience.com/understanding-random-forest-58381e0602d2
[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[3] https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d4
[4] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

SVMs

Support Vector Machines are intuitively an easy algorithm to understand. The algorithm looks to separate the data with a hyperplane which is placed in such a way that maximized the margin. Getting optimal margins is simply maximizing the perpendicular distance from all closest points to a line X. This algorithm works great with linearly separable data, but when the data isn't so defined a kernel transformation is implemented. A kernel transformation looks to transform existing features into new one which can be separable by a hyperplane. From this transformation, a seemingly unsolvable mess of data can become clear cut.[5]

Hyperparameters: [6]

- Kernel - specifies the kernel type to be used in the algorithm
- Gamma - kernel coefficient for each type
- C - regularization parameter
- Degree - degree of the polynomial kernel function; most complex attribute as raising something to a power can be very computationally greedy

Neural Networks

A neural network is a multi-layered network of neurons used to classify and predict outputs given an input. The network takes its name and general idea after the human brain; biological neurons are all connected to each other and have an activation energy where they either fire completely or don't at all. Between the input and output layers, a neural net has hidden layers which all host a certain activation function. If certain neurons within the hidden layers are activated, they then trigger specific neurons in the next which will eventually cascade to the output layer. The network can be trained by a variety of different means e.g. gradient descent, backpropagation, and from it very complex classification models can be formed.[7]

Hyperparameters:[8]

- Hidden Layer Sizes- representing the number of neurons in the ith hidden layer; like Random Forests, most complex parameter as the ones below are subsets of it
- Activation- activation function for the hidden layer
- Solver- solver for the weight optimization
- Alpha- L2 penalty
- Learning Rate- learning rate for weight updates

Hyperparameter Tuning-

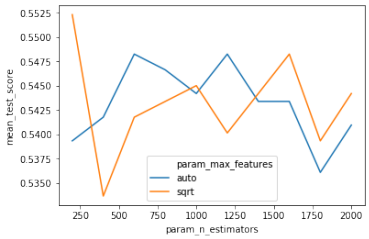
Scikit-learn has a function called GridSearchCV. It iterates through different permutations of hyper-parameter options while testing on different 'folds' of data. A fold of data is the original training data, but partitioned in unique subsets which offer uniqueness when tuning parameters. The folds test then averages out the score and parameters to not only give the best results, but also to try to prevent overfitting the training data. After the function finishes running, an output of the different combinations of hyper-parameters as well as its corresponding score is outputted. The best way to express the efficiency of the hyper-parameter

[5] https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496
[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[7] https://towardsdatascience.com/understanding-neural-networks-19020b758230
[8] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

was to plot the one which directly affected complexity the most. Below are plots of a specific parameter vs classification performance, a set of most optimal parameters, the time took to run the algorithm with the optimal parameters, and the classification score of each.

An interlude about different parameter searches: GridSearchCV is a great method to find the optimal hyper-parameters, but it becomes incredibly greedy after a while when you add multiple hyper-parameters and have to exaustively search through every permutation of them. To fix this problem, a method called RandomSearchCV was implemented. The search randomly tests different combinations of hyper-parameters and it can be as greedy or as efficient as needed as the number of iterations can be adjusted. If given enough iterations, the RandomSearchCV will generally find the optimal parameters more often then not by utilizing the almost voodo-magic property of randomness.[9] So below, for graphing just a one parameter against the accuracy, GridSearchCV was used. For finding the best combination of parameters to use for each model, RandomSearchCV was used.

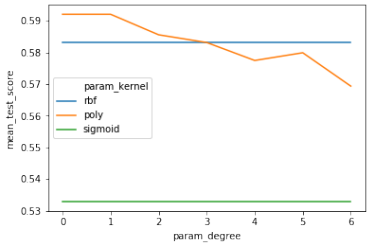
Random Forests



Classification score is: 0.6598580578154751
Total Model Run Time: 0.60 seconds

```
{'n_estimators': 200, 'min_samples_split': 5,
'min_samples_leaf': 2, 'max_features': 'sqrt',
'max_depth': 10, 'bootstrap': True}
```

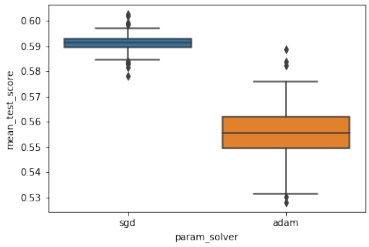
SVMs



Classification score is: 0.704691016098321
Total Model Run Time: 1.27 seconds

```
{'kernel': 'rbf', 'gamma': 10,
'degree': 2, 'C': 10}
```

Neural Networks



Classification score is: 0.5672494374242687
Total Model Run Time: 7.09 seconds

```
{'solver': 'sgd', 'learning_rate': 'constant',
'hidden_layer_sizes': (50, 40, 10, 30, 40, 10, 100, 100, 100, 100),
'alpha': 0.05, 'activation': 'tanh'}
```

Proof of Non-Triviality

The best method for proving non-triviality is to prove the data is not linearly separable. There were two ways devised to check for linear separability:

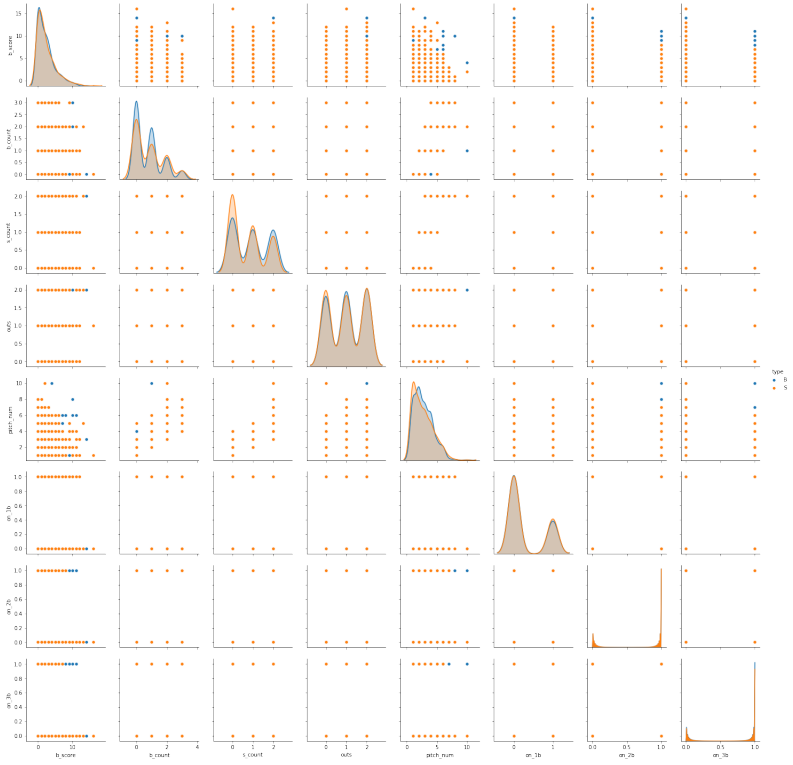
[9] https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85

1. A simple linear SVM model was run. If the data was linearly separable, the model would report a high classification score as it would be able to be separated by a straight line or hyperplane. As seen from the results, the classification was incredibly low, therefore indicating non-trivial data.

```
linear_test = LinearSVC()
linear_test.fit(X_train, y_train)
print(linear_test.score(X_test, y_test))
```

```
0.5476575121163166
```

2. A pair plot was produced of all of the different variables and a function was called to distinguish the different points with a class label. The orange and blue points signify strike or ball and proof of non-triviality is seen as there are not many ways to linearly separate the data in any of the plots.



Algorithm Perfomance

Once the hyper-parameters have been tuned and the model has been trained, the last step in the process is to tryout the model on test data. Using the data, each model's classification score, confusion matrix, and confidence intervals are shown.

Random Forests

Classification Score: 0.5452657088454215

Confidence Interval: (0.5309971138140019, 0.559534303876841)

Precision Score: 0.5737282678686413

Recall Score: 0.6508400292184076

Confusion Matrix

actual value	p	n	total
p′	True Positive: 461	False Negative: 662	P′
n′	False Postive: 478	True Negative: 861	N′
total	P	N	

SVMs

Classification Score: 0.5200623160810108

Confidence Interval: (0.5117250534324044, 0.5283995787296173)

Precision Score: 0.5675848814862268

Recall Score: 0.6471877282688093

Confusion Matrix

actual value	p	n	total
p′	True Positive: 699	False Negative: 424	P′
n′	False Postive: 209	True Negative: 1160	N′
total	P	N	

Neural Networks

Classification Score: 0.5575558248225723

Confidence Interval: (0.5485920291502575, 0.566519620494887)

Precision Score: 0.588681446907818

Recall Score: 0.7370343316289262

Confusion Matrix

actual value	p	n	total
p′	True Positive: 418	False Negative: 705	P′
n′	False Postive: 360	True Negative: 1009	N′
total	P	N	

As seen from the results, the Neural Network model was the highest scoring algorithm, but its confidence interval does not prove that its score is statistically different random forest at a degree of significance alpha=0.05. There was overlap with the CI of the Neural Net score with Random Forest so there could not be a statistical conclusion of which test had the absolute best score. Both Random Forests and Neural Nets were statistically different than SVMs as their confidence intervals did not overlap. The confusion matrices were added to see the false positive and negatives tested, and though they can usually be useful in say a medical test or a warranty analysis, for predicting pitching neither precision or recall score really matters. Even if it did matter, the Neural Net model still had the highest score of both as well.

Conclusion

Model Diagnostics
During an internship this summer, I was told that the hardest part of building out a model was not really fitting the data, but making sure you have the right data. I feel like that was the case with this project. The scores were pretty low and unfortunately for the project I could not get a useful model. To increase the score there are three paths to usually take: get more data, get more features about the data, do feature engineering. There was a lot of data in the repository found, but for speed and efficiency only a subset of the data was used. In most traditional ML algorithms, the performance of model will plateau once it reaches a certain point. For Deep Learning however, the performance will continually increase along with the data.[10] If given more time and more computational resources, the NN model could be vastly improved by adding more layers, more data, and more advanced tuning methods such as back-propagation. For getting more data features, if more aspects about the baseball game and players were given to the dataset, the accuracy and predictive power of the model would inevitably increase. Pitching at the end of the day is a human action, and it is almost impossible for a human to perform completely at random. On top of that, the pitcher has an absolute goal when trying to decide whether to throw a pitch or a strike: always to go against the favor of the batter. In terms of feature engineering, feature clustering could be implemented to help better improve the efficacy of the model.[11]

Algorithm Comparisons
For terms of real use the Random Forest Model should be used. The Random Forest algorithm is very powerful and it is great for the fact that it can prevent overfitting very easily with random splits at the data and bagging (bagging was used for the model as bootstrap=True - boosting did not make an appreciable difference so it was left out). The Random Forests model is also very robust, as the only real parameter that needs to be tuned is number of trees. When comparing metrics, factors such as training time and number or hyper-parameters shouldn't matter in real world situations. I think both aspects need to be proceeded with caution as overfitting could associate with training time and number of parameters, but it's not a causal relationship if the model is implemented properly. Shifting algorithms, Neural Nets may not be as easy to tune as Random Forests, but they do have the edge of scalability: i.e. training on large datasets. Since the data set used was a subset of the original, the NN was not the most ideal as it could be prone to overfitting, especially with the number of hidden layers used.[12] For example, if a person were trying to train a model to recognize images of cats on the internet, the model could overfit to categorize wrong objects as a cat. If there was a picture of a cat with a bell around its neck, the model may start associated the bell with what a cat is. So to combat overfitting in a Neural Network, there has to be a good balance between number of hidden layers and data size. In class we were told it's more of an art than an absolute science, and given the enigmatic 'black box' nature of neural networks in general, it may take a while for an absolute scientific approach to be made. Not to be forgotten, the SVM could be an approach to consider, but there should probably be more intuitive understanding of what the kernel transformation is doing to the data at hand or even maybe investigation into making a custom kernel.

[10] https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee
[11] https://towardsdatascience.com/how-to-create-new-features-using-clustering-4ae772387290
[12] https://www.kdnuggets.com/2016/04/deep-learning-vs-svm-random-forest.html

References

[1] "Basic Ensemble Learning (Random Forest, AdaBoost, Gradient Boosting)- Step by Step Explained." Medium, https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725.

[2] "Support Vector Machine — Simply Explained." Medium, https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496.

[3] "Classification - What Are the Differences between SVC, NuSVC, and LinearSVC?" Data Science Stack Exchange, https://datascience.stackexchange.com/questions/51813/what-are-the-differences-between-svc-nusvc-and-linearsvc.

[4] Does Adding More Layers Always Result in More Accuracy in Convolutional Neural Networks? - Quora. https://www.quora.com/Does-adding-more-layers-always-result-in-more-accuracy-in-convolutional-neural-networks.

[5] "How to Create New Features Using Clustering!!" Medium, https://towardsdatascience.com/how-to-create-new-features-using-clustering-4ae772387290.

[6] "In Depth: Parameter Tuning for SVC." Medium, https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769.

[7] "Machine Learning - How to Know Whether the Data Is Linearly Separable?" Cross Validated, https://stats.stackexchange.com/questions/182329/how-to-know-whether-the-data-is-linearly-separable.

[8] "How Do You Know You Have Enough Training Data?" Medium, https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee.

[9] "Neural Network - How to Implement Python's MLPClassifier with GridsearchCV?" Data Science Stack Exchange, https://datascience.stackexchange.com/questions/19768/how-to-implement-pythons-mlpclassifier-with-gridsearchcv. Accessed

[10] "Simple Tutorial on SVM and Parameter Tuning in Python and R." HackerEarth Blog, https://www.hackerearth.com/blog/developers/simple-tutorial-svm-parameter-tuning-python-r/.

[11] What Is the Difference between the AdaBoost and Random Forests Algorithms? - Quora. https://www.quora.com/What-is-the-difference-between-the-AdaBoost-and-random-forests-algorithms.

[12] "When Does Deep Learning Work Better Than SVMs or Random Forests?" KDnuggets, https://www.kdnuggets.com/when-does-deep-learning-work-better-than-svms-or-random-forests.html/.

[13] Worcester, Peter. "A Comparison of Grid Search and Randomized Search Using Scikit Learn." Medium, 6 June 2019, https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85.

[14] Yiu, Tony. "Understanding Neural Networks." Medium, https://towardsdatascience.com/understanding-neural-networks-19020b758230.

[15] "Understanding Random Forest." Medium, https://towardsdatascience.com/understanding-random-forest-58381e0602d2.