Machine Learning

CS 4641

Final Project Report

6 December 2019

1 Description

Datapoints
Although the original number of datapoints was 3624, only 1505 datapoints were used because many entries were incomplete (missing necessary features), and class was not common enough to be used in some of the algorithms, so it was omitted.

Features

1. Height (in)
2. Weight (lbs)
3. 40 Yard Dash (s)
4. Vertical Jump (in)
5. Bench Press (Reps)
6. Broad Jump (in)
7. 3 Cone (s)
8. Shuttle (s)
9. Offensive/Defense (0 – Offense/1 – Defense)

Labels:
Players are labeled by their position (e.g. Quarterback, Running Back, Safety). These labels were narrowed from the original data by grouping together similar positions. For example, 'Free Safeties' (FS) and 'Strong Safeties' (SS) were both labeled as just 'Safety' (S). Below is a table of the players' specific positions, the generalized positions used, and the numerical labeling used for calculation.

Listed Position	Listed Position Abbreviation	Generalized Position	Numerical Label
Offensive Line	OL	Offensive Line	0
Center	C	Offensive Line	0
Offensive Tackle	OT	Offensive Line	0
Offensive Guard	OG	Offensive Line	0
Long Snapper	LS	Offensive Line	0
Running Back	RB	Running Back	1
Full Back	FB	Running Back	1
Wide Receiver	WR	Wide Receiver	2
Tight End	TE	Tight End	3
Quarterback	QB	Quarterback	4
Defensive Line	DL	Defensive Line	5
Defensive End	DE	Defensive Line	5
Defensive Tackle	DT	Defensive Line	5
Edge Tackle	EDGE	Defensive Line	5
Linebacker	LB	Linebacker	6
Inside Linebacker	ILB	Linebacker	6
Outside Linebacker	OLB	Linebacker	6
Cornerback	CB	Cornerback	7
Safety	S	Safety	8
Free Safety	FS	Safety	8
Strong Safety	SS	Safety	8

2 Analysis

Introduction to the Dataset

The data I chose to use for this project consists of NFL combine statistics from the last ten years. The NFL combine is an event that many football players participate in prior to the NFL draft. The event consists of a set of standardized physical tests that give NFL scouts insight into the athleticism of players. I am interested in studying this dataset because I am curious if a future NFL player's position can be predicted by their combine stats. Although these classifiers alone won't have significant real-world application, it could serve as a valuable proof of concept for future studies that could be used to determine what sport/position an athlete *should* take on based on their physical abilities.

The primary metric I will use for comparing the chosen algorithms is the accuracy score of each model. I believe this is the best metric for comparison because the goal is to correctly predict as many players' positions as possible. There is no advantage or disadvantage of overpredicting or underpredicting one position, so accuracy score is a fine metric. We also don't run the risk of a model deciding to always or never predict a position because the positions are fairly evenly distributed, so doing such would yield a poor accuracy score anyway. The models will also be compared using their cross-validation scores and training time. The cross-validation scores will be analyzed within 95% confidence intervals.

Description of the Algorithms

The three algorithms I will be using to classify the data are random forests with bagging, support vector machines, and neural networks. Random forests with bagging groups the training data into sets or 'bags' and creates a classifier for each subset of the data. The bags are then aggregated by allowing each classifier to 'vote' on a label for each value being predicted. Support vector machines try to separate the data using hyperplanes by maximizing the distance between the separator and the nearest points. This can be done with a variety of Kernel functions which transform the data into ways that it can be separated better. Lastly, neural networks create a web of 'neurons' that fire to a successive level of neurons when they reach a threshold value. These essentially mimic neurons in the human brain in order to identify different classes. Each of these has several hyperparameters that can be tuned to improve the performance of the algorithms.

For the random forest classifier, implemented using sklearn.ensemble's Random Forest Classifier, the key hyperparameters consist of the number of trees in the forest, max depth of each tree, minimum samples necessary to split a node, minimum samples necessary for a node to be a leaf node, and the maximum number of features to look at when splitting the data. Optimizing these parameters will allow the models to better classify the data without overfitting (which can be achieved by using folds).

For my support vector machine, I used sklearn's support vector classifier, or SVC. The parameters that I tuned for this model were C, the regularization parameter, the kernel type, and gamma which can limit or expand the influence of a training point. I chose to optimize only three parameters because training support vector classifiers took a significant amount of time, so it made sense to focus on optimizing the most important hyperparameters.

Lastly, when training my neural network models, using sklearn's MLPClassifier, I focused on optimizing the dimensions of the hidden layers (size and number of layers), learning rate, alpha, and activation functions for each node. Learning rate is relevant to how the hypothesis class is updated each iteration while alpha is the L2 penalty.

Tuning Hyperparameters

To tune my hyperparameters, I used a random search followed by a grid search. In the random search, a wide range of values were used for each parameter in order to focus in on reasonable values for each. I chose this strategy because it is not necessary to test every value, like in the case of a grid search, because we will focus in on the chosen values later anyway. For the random search, we just want to gauge what range we should be looking in. Following the random search, we test parameter values around the values returned by the random fit using a grid search in order to get specific values that are close to optimal. Below are the metrics on the hyperparameter tuning and the final hyperparameters chosen. All unlisted hyperparameters were left as the sklearn defaults.

Random Forest Classifier

RFC Final Hyperparameters	
Number of Trees	550
Max Features	Auto
Min Sample Splits	7
Min Samples Leaf	2
Max Depth	20

Randomized Search CV Runtime: 3.6 minutes
Grid Search CV Runtime: 4.3 minutes

RFC Accuracy Analysis			
	Original Accuracy	Post-Tuning Accuracy	Percent Change
Training Data	99.17%	96.01%	-3.13%
Test Data	81.06%	87.71%	8.20%

Support Vector Machine

SVC Final Hyperparameters	
C	80
Gamma	0.007
Kernel	Radial Basis Function

Randomized Search CV Runtime: 7.7 seconds
Grid Search CV Runtime: 10.1 seconds

SVC Accuracy Analysis			
	Original Accuracy	Post-Tuning Accuracy	Percent Change
Training Data	99.59%	97.43%	-2.17%
Test Data	49.17%	79.07%	60.81%

Neural Network

NNC Final Hyperparameters	
Activation	Tan(h)
Alpha	0.2
Layers	125
Neurons per Layer	25
Learning Rate	Inverse Scaling

Randomized Search CV Runtime: 3.0 minutes
Grid Search CV Runtime: 5.1 minutes

NNC Accuracy Analysis			
	Original Accuracy	Post-Tuning Accuracy	Percent Change
Training Data	44.44%	54.73%	23.15%
Test Data	40.20%	52.49%	23.41%

The hyperparameter metrics above support the claim that the data is not trivial because complex learning models, enhanced with tuned hyperparameters, do an increasingly successful job of classifying the data. If the data was trivial, the data would be best classified by something simplistic like a linear support vector machine and would show little to no improvement with increased tuning.

Comparing Algorithm Performance

Random Forests with Bagging

Train Data Accuracy: 96.10%

Test Data Accuracy: 87.71%

Support Vector Machines (Non-linear Kernel)

Train Data Accuracy: 97.43%

Test Data Accuracy: 79.07%

Neural Network

Train Data Accuracy: 54.73%

Test Data Accuracy: 52.49%

To compare algorithm performance, I began by comparing the algorithms success at classifying both training and test data by calculating what percentage of points were classified correctly by each model. As can be seen above, random forests with bagging had the best overall performance based on this metric. It was only slightly behind support vector machines in terms of training accuracy, and it was far ahead of both other algorithms in terms of test data accuracy. Test data is the more important factor

when comparing these algorithms because we want to be able to generalize the models to all data and not just data that was in our training set, which would be a sign of overfitting. By this metric, with a very good training data accuracy to support it, random forests with bagging was the runaway best model.

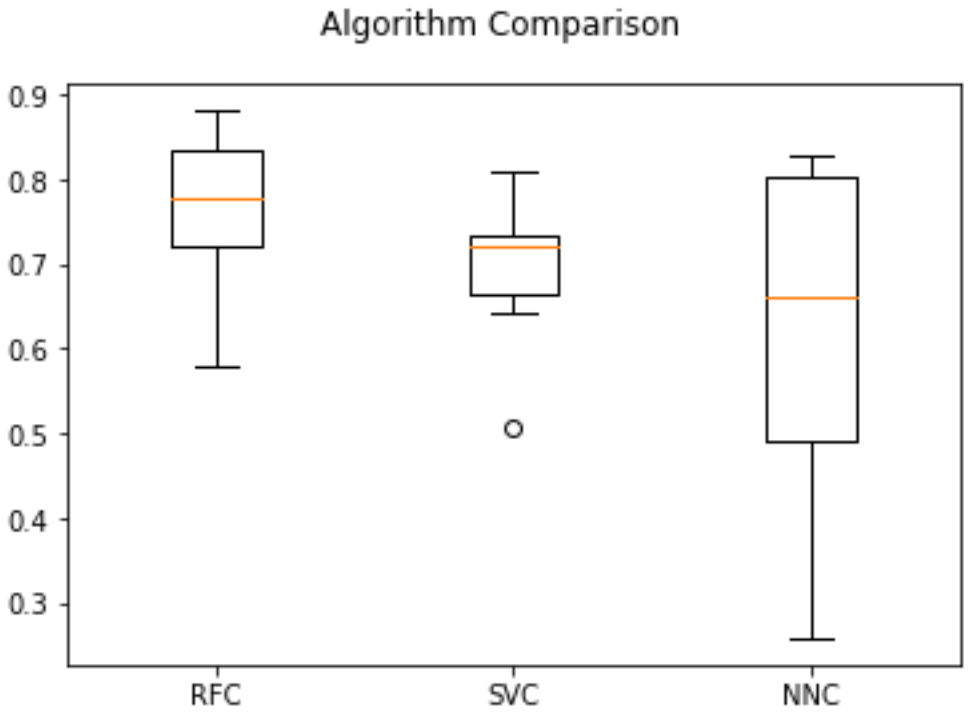

Figure 1. Box and whisker plot of algorithms and their corresponding cross validation scores over multiple folds.

	Cross Validation Score	Cross Validation Score Standard Deviation	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
Random Forest Classification	0.7647	0.0860	0.76	0.769
Support Vector Classification	0.6956	0.0768	0.692	0.699
Neural Network Classification	0.6272	0.1930	0.617	0.637

This observation is further supported by my next metric of comparison, cross-validation score. Cross validation varies the training and test set data and compares the average and standard deviation scores. As can be seen in Figure 1, the models were more similar in score in this regard than in terms of accuracy score, but random forests with bagging is still a convincing first place.

A final metric that should be considered is execution time. All models with specified hyperparameters took less than 30 seconds to train, and hyperparameter tuning only took a matter of minutes. Support vector machines were able to be hyperparameter-tuned in a matter of seconds as opposed to in minutes in the other cases (due to having fewer hyperparameters of interest), but this is insignificant due to the nature of the data. NFL combine stats only come out once a year, and there simply isn't enough data to make time a deciding factor. It is much more optimal to get the maximum learning out of the data possible, because it is limited, instead of worrying about a small difference in performance time.

Conclusion

From the data, it appears that random forests with bagging is the best algorithm for classification in almost every metric. It had the best performance in both test-data accuracy and cross-validation score, which I identified as the most important factors for comparison and did not take long enough to have concerns about execution time based on the nature of the data. This is the algorithm I would use for similar real-world applications.

Acknowledgements

For help with this assignment, I used the following sources:

Scikit-Learn Documentation found at scikit-learn.org
Towards Data Science for Train/Test Split and Cross Validation information found at
https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
Stack Overflow for a variety of non-Machine Learning related Python questions
Pro-Football-Reference.com for all NFL Combine stats used