CIS 508 Individual Assignment #2

Paul Bernert

September 22, 2020

To Do List

Task One: Perform hyper-parameter tuning of the two classifiers by changing at least three different hyper-parameters of each classifier.

Task Two: Use both random and grid search for hyper-parameter tuning. Also use cross-validation.

Task Three: Discuss what hyper-parameter ranges are best for each classifier for these two problems.

Task Four: Compare and evaluate the different classifiers using the test-set and find the best classifier for criterion (e.g. accuracy, precision, recall, F1-score).

Solution

The objective of this homework is to move away from feeding random parameters into the*DecisionTreeClassifier* function within Scikit-learn and try to refine our models. This week, we will focus on achieving more accurate models by doing hyper-parameter tuning on parameters within the *DecisionTreeClassifier* and *RandomForest* functions.

We can achieve that knowledge by applying sklearn functions to the provided datasets. We will be solving two problems. The first problem aims to be able to identify parameters that can be used to detect fraud, while the second problem wants us to accurately predict targeted marketing with Portuguese Bank data.

Problem One: Fraud Detection

Cleaning the Data

The provided datasets for this problem include a variety of different variables, most of which are considered categorical. In order to proceed, we must first concatenate the two datasets and perform *OneHotEncoding*. Fortunately, the Pandas library has recently made great progress in their *get_dummies* function, so we can simply proceed using that.

It's worth noting that the provided datasets were split in a way that have less than 20% of the observations in the **train** dataset and over 80% in the **test** dataset. Because of this, the two ways to proceed would be doing analysis with **bagging**

or bootstrapping, or simply creating my own split of the data into train and test. I decided to use the latter, since the observations are assumed to have interchangeability between the two files.

With *OneHotEncoding* out of the way, we can then split the data back into a train-test split using Sklearn's *train_test_split* function.

With the data reclassified into training and testing sets, we can now begin to create our first model.

DecisionTreeClassifier

We begin with a basic *DecisionTreeClassifier* function with default parameters.

$$DecisionTreeClassifier()$$

This default *DecisionTreeClassifier* model produces an Accuracy of: 95.89 percent.

Similar to last week's assignment, we can use this default outcome and compare how different parameters impact the accuracy of the model when cast against the train and test datasets. Let's begin with *GridSearchCV*.

GridSearchCV

GridSearchCV is a function within Sklearn where, when provided a dictionary of parameters and ranges of values for those parameters, can iterate through every single available option and determine which one produces the most accurate model. Since parameter scanning is an iterative process, it can be quite resource-intensive, usually taking significantly longer (and thus, more "expensive") than alternatives such as *RandomSearchCV*. However,the results can potentially be more accurate, and I predict that will be the case on this assignment.

To do *GridSearch*, we must start by providing the set of parameters and their ranges. To help with this process, it's usually a good practice to start with broad estimates and continue to refine the grid search until you've reached a conclusion you're satisfied with.

Parameters:

min_samples_split : [i*5 for i in range(1,20)]

criterion : ['gini','entropy']

max_leaf_nodes : [i for i in range(5,10)]

Outcome:

After testing 950 potential fits, the optimal solution was:

$$DecisionTreeClassifier(min\_sample\_split = 35, criterion = `gini', max\_leaf\_nodes = 9)$$

Producing an Accuracy of: 94.22 percent.

RandomSearchCV

RandomSearchCV is also a function with Sklearn where, when provided a dictionary of parameters and ranges of values for those parameters, it will randomly select a user-determined number of choices available within the ranges provided. This can often be significantly faster and less resource-intensive when compared to *GridSearchCV*, but can often result in a less accurate model.

Outcome:

When using the same parameters listed above, we received the following outcome:

$$DecisionTreeClassifier(min\_sample\_split = 10, criterion = `gini', max\_leaf\_nodes = 9)$$

It produced basically the same accuracy, but that is also due to the fact that after many (and I mean **MANY**) iterations through optimal parameters, I've created such a refined range of parameters that either model will generally produce the same results. However, in the *RandomForest* models below, I will intentionally keep the range higher to show that *GridSearchCV* will produce more accurate results on average.

RandomForest

RandomForest operates similarly to the standard *DecisionTreeClassifier* function in many ways, as it combines the output of multiple (randomly created) Decision Trees to produce its final output. However, because of it's combined, iterative process, it should produce slightly more accurate results. Let's test that theory using by doing a basic *RandomForest*:

$$RandomForestClassifier()$$

This model produces an accuracy of: 97.93 percent. This is already higher than the *DecisionTreeClassifier*, though we can test additional parameters, starting once again with *GridSearchCV*.

GridSearchCV

The parameters used for *GridSearchCV* included the following:

min_samples_leaf : [i for i in range(1,3)]

max_depth : [i for i in range(20,25)]

max_features : [i for i in range(15,25)]

n_estimators : [i for i in range(1,10)]

Outcome:

Those parameters produced the following as the ideal parameters:

$$RandomForestClassifier(max\_depth = 24, max\_features = 19, min\_samples\_leaf = 1, n\_estimators = 8)$$

This model produced an accuracy of: 96.77 percent. Let's continue with *RandomSearchCV*.

RandomSearchCV

When doing *RandomSearchCV*, the same parameters are used. However, I am using only 25 different random collections from those parameter options. After running 25 tests, the optimal solution provided was:

$$RandomForestClassifier(max\_depth = 24, max\_features = 16, min\_samples\_leaf = 1, n\_estimators = 9)$$

This model produced an accuracy of: 96.73 percent. Indeed, *GridSearchCV* produces a more accurate model as anticipated.

Conclusion

See the end of this report for a universal conclusion about the data, models and outcomes.

Problem Two: Bank Marketing

This problem has a similar structure to the previous. However, not all variables are categorical this time around. As a result, it is a little more nuanced when structuring the data for encoding and for the train and test split.

However, if we are able to apply encoding to only the variables that need it, we can proceed very similarly to the previous question.

DecisionTreeClassifier

We begin with a basic *DecisionTreeClassifier* function with default parameters.

$$DecisionTreeClassifier()$$

This model produces an accuracy of 89.35 percent when tested against the training dataset (potentially higher in the test dataset). However, there should be room for improvement as we alter parameters, starting once again with *GridSearchCV*.

GridSearchCV

Parameters:

min_samples_split : [i*5 for i in range(10,20)]

criterion : ['gini','entropy']

max_leaf_nodes : [i*5 for i in range(2,6)]

Outcome:

After testing 400 potential fits, the optimal solution was:

$$DecisionTreeClassifier(min\_sample\_split = 85, criterion = `gini', max\_leaf\_nodes = 25)$$

Producing an Accuracy of: 90.22 percent. Finally, we have a model that has been improved due to additional parameters! Let's continue with *RandomSearchCV*

RandomSearchCV

Outcome:

When using the same parameters listed above, we received the following outcome after testing against 25 random selections of parameters:

$$DecisionTreeClassifier(min\_sample\_split = 95, criterion = `gini', max\_leaf\_nodes = 25)$$

It produced basically the same accuracy, but that is also due to the fact that after many (and I mean **MANY**) iterations through optimal parameters, I've created such a refined range of parameters that either model will generally produce the same results. Once again, in the *RandomForest* models below, I will intentionally keep the range higher to show that *GridSearchCV* will produce more accurate results on average.

RandomForest

We once again start with a default *RandomForest* function:

$$RandomForestClassifier()$$

With an accuracy of 91.34, we have once again improved relative to a basic *DecisionTreeClassifier* function. Let's continue with *GridSearchCV*.

GridSearchCV

The parameters used for *GridSearchCV* included the following:

min_samples_leaf : [i for i in range(1,2)]

max_depth : [i*5 for i in range(3,5)]

max_features : [i*5 for i in range(3,5)]

n_estimators : [i for i in range(2,6)]

Outcome:

Those parameters produced the following as the ideal parameters:

$$RandomForestClassifier(max\_depth = 20, max\_features = 20, min\_samples\_leaf = 1, n\_estimators = 5)$$

This model produced an accuracy of: 89.65 percent. Let's continue with *RandomSearchCV*.

RandomSearchCV

When using the same parameters listed above, we received the following outcome after testing against 25 random selections of parameters:

$$RandomForestClassifier(max\_depth = 20, max\_features = 20, min\_samples\_leaf = 1, n\_estimators = 5)$$

This model produced an accuracy of 89.29 percent. Lower than the GridSearchCV, as expected!

Conclusions

Once again, we are working with an imbalanced datasest. As a result, our parameter scanning doesn't necessarily produce more accurate results. We should consider switching to a model scheme that focuses less on accuracy and perhaps moving towards a Cost model for penalizing false-positives and false-negatives.