GridSearchCV

(https://github.com/learn-co-curriculum/dsc-gridsearchcv) (https://github.com/learn-co-curriculum/dsc-gridsearchcv/issues/new/choose)

Introduction

In this lesson, we'll explore the concept of parameter tuning to maximize our model performance using a combinatorial grid search!

Objectives

You will be able to:

- Design a parameter grid for use with scikit-learn's `GridSearchCV`
- Use `GridSearchCV` to increase model performance through parameter tuning

Parameter tuning

By now, you've seen that the process of building and training a supervised learning model is an iterative one. Your first model rarely performs the best! There are multiple ways we can potentially improve model performance. Thus far, most of the techniques we've used have been focused on our data. We can get better data, or more data, or both. We can engineer certain features, or clean up the data by removing rows/variables that hurt model performance, like multicollinearity.

The other major way to potentially improve model performance is to find good parameters to set when creating the model. For example, if we allow a decision tree to have too many leaves, the model will almost certainly overfit the data. Too few, and the model will underfit. However, each modeling problem is unique -- the same parameters could cause either of those situations, depending on the data, the task at hand, and the complexity of the model needed to best fit the data.

In this lesson, we'll learn how we can use a ***combinatorial grid search*** to find the best combination of parameters for a given model.

Grid search

When we set parameters in a model, the parameters are not independent of one another -- the value set for one parameter can have significant effects on other parameters, thereby affecting overall model performance. Consider the following grid.

Parameter	1	2	3	4
criterion	"gini"	"entropy"		

Parameter	1	2	3	4
max_depth	1	2	5	10
min_samples_split	1	5	10	20

All the parameters above work together to create the framework of the decision tree that will be trained. For a given problem, it may be the case that increasing the value of the parameter for `min_samples_split` generally improves model performance up to a certain point, by reducing overfitting. However, if the value for `max_depth` is too low or too high, this may doom the model to overfitting or underfitting, by having a tree with too many arbitrary levels and splits that overfit on noise, or limiting the model to nothing more than a "stump" by only allowing it to grow to one or two levels.

So how do we know which combination of parameters is best? The only way we can really know for sure is to try ***every single combination!*** For this reason, grid search is sometimes referred to as an ***exhaustive search***.

Use `GridSearchCV`

The `sklearn` library provides an easy way to tune model parameters through an exhaustive search by using its **`GridSearchCV`** (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) class, which can be found inside the `model_selection` module. `GridsearchCV` combines ***K-Fold Cross-Validation*** with a grid search of parameters. In order to do this, we must first create a ***parameter grid*** that tells `sklearn` which parameters to tune, and which values to try for each of those parameters.

The following code snippet demonstrates how to use `GridSearchCV` to perform a parameter grid search using a sample parameter grid, `param_grid`. Our parameter grid should be a dictionary, where the keys are the parameter names, and the values are the different parameter values we want to use in our grid search for each given key. After creating the dictionary, all you need to do is pass it to `GridSearchCV()` along with the classifier. You can also use K-fold cross-validation during this process, by specifying the `cv` parameter. In this case, we choose to use 3-fold cross-validation for each model created inside our grid search.

```
clf = DecisionTreeClassifier()

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [1, 2, 5, 10],
    'min_samples_split': [1, 5, 10, 20]
}

gs_tree = GridSearchCV(clf, param_grid, cv=3)
```

```
gs_tree.fit(train_data, train_labels)

gs_tree.best_params_
```

This code will run all combinations of the parameters above. The first model to be trained would be `DecisionTreeClassifier(criterion='gini', max_depth=1, min_samples_split=1)` using a 3-fold cross-validation, and recording the average score. Then, it will change one parameter, and repeat the process (e.g., `DecisionTreeClassifier(criterion='gini', max_depth=1, min_samples_split=5)` , and so on), keeping track of the overall performance of each model. Once it has tried every combination, the `GridSearchCV` object we created will automatically default the model that had the best score. We can even access the best combination of parameters by checking the `best_params_` attribute!

Drawbacks of `GridSearchCV`

GridSearchCV is a great tool for finding the best combination of parameters. However, it is only as good as the parameters we put in our parameter grid -- so we need to be very thoughtful during this step!

The main drawback of an exhaustive search such as `GridsearchCV` is that there is no way of telling what's best until we've exhausted all possibilities! This means training many versions of the same machine learning model, which can be very time consuming and computationally expensive. Consider the example code above -- we have three different parameters, with 2, 4, and 4 variations to try, respectively. We also set the model to use cross-validation with a value of 3, meaning that each model will be built 3 times, and their performances averaged together. If we do some simple math, we can see that this simple grid search we see above actually results in `2 * 4 * 4 * 3 =` ***96 different models trained!*** For projects that involve complex models and/or very large datasets, the time needed to run a grid search can often be prohibitive. For this reason, be very thoughtful about the parameters you set -- sometimes the extra runtime isn't worth it -- especially when there's no guarantee that the model performance will improve!

Summary

In this lesson, you learned about grid search, how to perform grid search, and the drawbacks associated with the method!

How do you feel about this lesson?

Have specific feedback?

Tell us here! (https://github.com/learn-co-curriculum/dsc-gridsearchcv/issues/new/choose)