Applying CV to Decision Trees

Apply K-Fold cross-validation to decision trees.

Chapter Goals:

- Apply K-Fold cross-validation to a decision tree

A. Decision tree depth

We've previously discussed cross-validation for tuning hyperparameters such as the α value for regularized regression. For decision trees, we can tune the tree's maximum depth hyperparameter (`max_depth`) by using K-Fold cross-validation.

K-Fold cross-validation gives an accurate measurement of how good the decision tree is for the dataset. We can use K-Fold cross-validation with different values of the `max_depth` hyperparameter and see which one gives the best cross-validation scores.

The code below demonstrates how to apply K-Fold CV to tune a decision tree's maximum depth. It uses the `cv_decision_tree` function that you will implement later in this chapter.

```
is_clf = True  # for classification
for depth in range(3, 8):
  # Predefined data and labels
  scores = cv_decision_tree(
    is_clf, data, labels, depth, 5)  # k = 5
  mean = scores.mean()  # Mean acc across folds
  std_2 = 2 * scores.std()  # 2 std devs
  print('95% C.I. for depth {}: {} +/- {:.2f}\n'.format(
    depth, mean, std_2))
```

Output

1.306s

```
95% C.I. for depth 3: 0.9201583773270352 +/- 0.03

95% C.I. for depth 4: 0.9234917106603685 +/- 0.04

95% C.I. for depth 5: 0.9299944429008058 +/- 0.04

95% C.I. for depth 6: 0.9199907381680095 +/- 0.03

95% C.I. for depth 7: 0.9067157543762155 +/- 0.06
```

In the above code, we use the `cv_decision_tree` function to apply 5-Fold cross-validation to a classification decision tree. We tune its maximum depth hyperparameter across depths of 3, 4, 5, 6, and 7. For each `max_depth` value, we print the 95% confidence interval (https://en.wikipedia.org/wiki/Confidence_interval) for the cross-validated scores across the 5 folds.

For the most part, the maximum depth of 4 produces the best 95% confidence interval of cross-validated scores. This would be the value of `max_depth` that we choose for the final decision tree.

If the confidence interval had consistently continued to improve for maximum depths of 5, 6 and 7, we would have continued applying the cross-validation process to evaluate larger maximum depth values.

Time to Code!

The coding exercise for this chapter is to complete the aforementioned `cv_decision_tree` function. The function's first argument defines whether the decision tree is for classification/regression, the next two arguments represent the data/labels, and the final two arguments represent the tree's maximum depth and number of folds, respectively.

First, we'll create the decision tree (using the `tree` module imported in the backend).

Initialize `d_tree` with `tree.DecisionTreeClassifier` if `is_clf` is `True`, otherwise use `tree.DecisionTreeRegressor`. In either case, initialize with keyword argument `max_depth` set to `max_depth`.

Then we'll use the `cross_val_score` function (imported in the backend) to obtain the CV scores.

Set `scores` equal to `cross_val_score` applied with `d_tree`, `data`, and `labels` for the first three arguments. Use `cv=cv` for the keyword argument, then return `scores`.

```
def cv_decision_tree(is_clf, data, labels,
                     max_depth, cv):
  # CODE HERE
  pass
```

Solution

```
def cv_decision_tree(is_clf, data, labels,
                     max_depth, cv):
  if is_clf:
    d_tree = tree.DecisionTreeClassifier(max_depth=max_depth)
  else:
    d_tree = tree.DecisionTreeRegressor(max_depth=max_depth)
  scores = cross_val_score(d_tree, data, labels, cv=cv)
  return scores

```

Back
Cross-Validation

Next
Evaluating Models

Mark as Completed