KFoldCrossValidation

December 20, 2016

1 K-Fold Cross Validation

Let's revisit the Iris data set:

```
In [1]: import numpy as np
        from sklearn import cross_validation
        from sklearn import datasets
        from sklearn import svm

        iris = datasets.load_iris()
```

A single train/test split is made easy with the train_test_split function in the cross_validation library:

```
In [2]: # Split the iris data into train/test data sets with 40% reserved for testing
        X_train, X_test, y_train, y_test = cross_validation.train_test_split(iris.data, iris.tar

        # Build an SVC model for predicting iris classifications using training data
        clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)

        # Now measure its performance with the test data
        clf.score(X_test, y_test)
```

```
Out[2]: 0.96666666666666667
```

K-Fold cross validation is just as easy; let's use a K of 5:

```
In [3]: # We give cross_val_score a model, the entire data set and its "real" values, and the nu
        scores = cross_validation.cross_val_score(clf, iris.data, iris.target, cv=5)

        # Print the accuracy for each fold:
        print(scores)

        # And the mean accuracy of all 5 folds:
        print(scores.mean())
```

```
[ 0.96666667  1.          0.96666667  0.96666667  1.        ]
0.98
```

Our model is even better than we thought! Can we do better? Let's try a different kernel (poly):

```
In [4]: clf = svm.SVC(kernel='poly', C=1).fit(X_train, y_train)
        scores = cross_validation.cross_val_score(clf, iris.data, iris.target, cv=5)
        print(scores)
        print(scores.mean())

[ 1.          1.          0.9         0.93333333  1.        ]
0.966666666667
```

No! The more complex polynomial kernel produced lower accuracy than a simple linear kernel. The polynomial kernel is overfitting. But we couldn't have told that with a single train/test split:

```
In [5]: # Build an SVC model for predicting iris classifications using training data
        clf = svm.SVC(kernel='poly', C=1).fit(X_train, y_train)

        # Now measure its performance with the test data
        clf.score(X_test, y_test)

Out[5]: 0.9666666666666667
```

That's the same score we got with a single train/test split on the linear kernel.

1.1 Activity

The "poly" kernel for SVC actually has another attribute for the number of degrees of the polynomial used, which defaults to 3. For example, svm.SVC(kernel='poly', degree=3, C=1)

We think the default third-degree polynomial is overfitting, based on the results above. But how about 2? Give that a try and compare it to the linear kernel.

```
In [ ]:
```