spam_rf

April 24, 2023

0.1 Fit a Random Forest

In this exercise, you'll train a Random Forest classifier to predict whether or not a text message is "spam". In order to train the classifier, you'll use a dataset of SMS messages labeled as "spam" and "ham" (not spam). The predictions will be based on the counts of each word in the text message. Before using a Random Forest, see how well a simple Decision Tree model performs.

In [1]:
```
# Import our libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Read in our dataset
df = pd.read_table('SMSSpamCollection.dms',
                   sep='\t',
                   header=None,
                   names=['label', 'sms_message'])

# Fix our response value
df['label'] = df.label.map({'ham':0, 'spam':1})

# Split our dataset into training and testing data
X_train, X_test, y_train, y_test = train_test_split(df['sms_message'],
                                                    df['label'],
                                                    random_state=1)

# Instantiate the CountVectorizer method
count_vector = CountVectorizer()

# Fit the training data and then return the matrix
training_data = count_vector.fit_transform(X_train)

# Transform testing data and return the matrix. Note we are not fitting the testing data
testing_data = count_vector.transform(X_test)
```

```
# Instantiate our model
decision_tree = DecisionTreeClassifier()

# Fit our model to the training data
decision_tree.fit(training_data, y_train)

# Predict on the test data
predictions = decision_tree.predict(testing_data)

# Score our model
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))
```

```
Accuracy score:  0.9633883704235463
Precision score:  0.8418367346938775
Recall score:  0.8918918918918919
F1 score:  0.8661417322834645
```

The simple Decision Tree appears to have worked reasonably well, but there is room for improvement. Notice that in order to train and test the model, we took the following steps:

1. **Import** the model
2. **Instantiate** the model
3. **Fit** the model on training data
4. **Test** the model on testing data
5. **Score** the model by comparing the predictions to the true values

We'll do the same steps for the Random Forest model—but this time, you fill in the appropriate code!

Step 1: First `import` the RandomForestClassifier module.

```
In [2]: # Import the Random Forest Classifier
        from sklearn.ensemble import RandomForestClassifier
```

Step 2: Then, `instantiate` the classifier.

```
In [3]: # Instantiate a RandomForestClassifier with
        # 200 weak learners (n_estimators) and everything else as default values
        rf = RandomForestClassifier(n_estimators=200)
```

Step 3: Now, `fit` (train) the model with `training_data` and `y_train`. This may take a little time.

```
In [4]: # Fit the RandomForestClassifier model
        rf.fit(training_data, y_train)
```

Out[4]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Step 4: Use `predict` to test the model on previously unseen data.

In [5]:
```
# Call model.predict() to test the model on the test data
rf_predictions = rf.predict(testing_data)
```

Step 5: Score the predictions.

In [6]:
```
# Calculate the accuracy, precision, recall, and F1 scores
print('Random Forest scores:')
print('Accuracy score: ', format(accuracy_score(y_test, rf_predictions)))
print('Precision score: ', format(precision_score(y_test, rf_predictions)))
print('Recall score: ', format(recall_score(y_test, rf_predictions)))
print('F1 score: ', format(f1_score(y_test, rf_predictions)))
```

```
Random Forest scores:
Accuracy score:  0.9834888729361091
Precision score:  1.0
Recall score:  0.8756756756756757
F1 score:  0.9337175792507205
```

Let's re-print the Decision Tree scores again so we can look at them side-by-side.

In [ ]:
```
print('Decision Tree scores:')
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))
```

Interesting! It looks like the Random Forest outperformed the simple Decision Tree in all metrics except recall.

If you need a little help with this exercise, check out the solution notebook here.

In [ ]: