Spam_&_Ensembles_Solution

January 17, 2021

0.1 Our Mission

You recently used Naive Bayes to classify spam in this dataset. In this notebook, we will expand on the previous analysis by using a few of the new techniques you saw throughout this lesson.

> In order to get caught back up to speed with what was done in the previous notebook, run the cell below

In [3]:
```
# Import our libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Read in our dataset
df = pd.read_table('smsspamcollection/SMSSpamCollection',
                   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
naive_bayes = MultinomialNB()

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

# Predict on the test data
predictions = naive_bayes.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:  0.9885139985642498
Precision score:  0.9720670391061452
Recall score:  0.9405405405405406
F1 score:  0.9560439560439562
```

0.1.1 Turns Out...

It turns out that our naive bayes model actually does a pretty good job. However, let's take a look at a few additional models to see if we can't improve anyway.

Specifically in this notebook, we will take a look at the following techniques:

- BaggingClassifier
- RandomForestClassifier
- AdaBoostClassifier

Another really useful guide for ensemble methods can be found in the documentation here.

These ensemble methods use a combination of techniques you have seen throughout this lesson:

- **Bootstrap the data** passed through a learner (bagging).
- **Subset the features** used for a learner (combined with bagging signifies the two random components of random forests).
- **Ensemble learners** together in a way that allows those that perform best in certain areas to create the largest impact (boosting).

In this notebook, let's get some practice with these methods, which will also help you get comfortable with the process used for performing supervised machine learning in python in general.

Since you cleaned and vectorized the text in the previous notebook, this notebook can be focused on the fun part - the machine learning part.

0.1.2 This Process Looks Familiar...

In general, there is a five step process that can be used each type you want to use a supervised learning method (which you actually used above):

1. **Import** the model.
2. **Instantiate** the model with the hyperparameters of interest.
3. **Fit** the model to the training data.
4. **Predict** on the test data.
5. **Score** the model by comparing the predictions to the actual values.

Follow the steps through this notebook to perform these steps using each of the ensemble methods: **BaggingClassifier**, **RandomForestClassifier**, and **AdaBoostClassifier**.

Step 1: First use the documentation to `import` all three of the models.

```
In [4]: # Import the Bagging, RandomForest, and AdaBoost Classifier
        from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifi
```

Step 2: Now that you have imported each of the classifiers, `instantiate` each with the hyperparameters specified in each comment. In the upcoming lessons, you will see how we can automate the process to finding the best hyperparameters. For now, let's get comfortable with the process and our new algorithms.

```
In [5]: # Instantiate a BaggingClassifier with:
        # 200 weak learners (n_estimators) and everything else as default values
        bag_mod = BaggingClassifier(n_estimators=200)


        # Instantiate a RandomForestClassifier with:
        # 200 weak learners (n_estimators) and everything else as default values
        rf_mod = RandomForestClassifier(n_estimators=200)

        # Instantiate an a AdaBoostClassifier with:
        # With 300 weak learners (n_estimators) and a learning_rate of 0.2
        ada_mod = AdaBoostClassifier(n_estimators=300, learning_rate=0.2)
```

Step 3: Now that you have instantiated each of your models, `fit` them using the **training_data** and **y_train**. This may take a bit of time, you are fitting 700 weak learners after all!

```
In [6]: # Fit your BaggingClassifier to the training data
        bag_mod.fit(training_data, y_train)

        # Fit your RandomForestClassifier to the training data
        rf_mod.fit(training_data, y_train)

        # Fit your AdaBoostClassifier to the training data
        ada_mod.fit(training_data, y_train)
```

Out [6]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=0.2, n_estimators=300, random_state=None)

Step 4: Now that you have fit each of your models, you will use each to `predict` on the **testing_data**.

In [7]:
```
# Predict using BaggingClassifier on the test data
bag_preds = bag_mod.predict(testing_data)

# Predict using RandomForestClassifier on the test data
rf_preds = rf_mod.predict(testing_data)

# Predict using AdaBoostClassifier on the test data
ada_preds = ada_mod.predict(testing_data)
```

Step 5: Now that you have made your predictions, compare your predictions to the actual values using the function below for each of your models - this will give you the `score` for how well each of your models is performing. It might also be useful to show the naive bayes model again here.

In [14]:
```
def print_metrics(y_true, preds, model_name=None):
    '''
    INPUT:
    y_true - the y values that are actually true in the dataset (numpy array or pandas
    preds - the predictions for those values from some model (numpy array or pandas ser
    model_name - (str - optional) a name associated with the model if you would like to

    OUTPUT:
    None - prints the accuracy, precision, recall, and F1 score
    '''
    if model_name == None:
        print('Accuracy score: ', format(accuracy_score(y_true, preds)))
        print('Precision score: ', format(precision_score(y_true, preds)))
        print('Recall score: ', format(recall_score(y_true, preds)))
        print('F1 score: ', format(f1_score(y_true, preds)))
        print('\n\n')

    else:
        print('Accuracy score for ' + model_name + ' :' , format(accuracy_score(y_true,
        print('Precision score ' + model_name + ' :', format(precision_score(y_true, pr
        print('Recall score ' + model_name + ' :', format(recall_score(y_true, preds)))
        print('F1 score ' + model_name + ' :', format(f1_score(y_true, preds)))
        print('\n\n')
```

In [17]:
```
# Print Bagging scores
print_metrics(y_test, bag_preds, 'bagging')

# Print Random Forest scores
print_metrics(y_test, rf_preds, 'random forest')
```

```python
# Print AdaBoost scores
print_metrics(y_test, ada_preds, 'adaboost')

# Naive Bayes Classifier scores
print_metrics(y_test, predictions, 'naive bayes')
```

```
Accuracy score for bagging : 0.9748743718592965
Precision score bagging : 0.9166666666666666
Recall score bagging : 0.8918918918918919
F1 score bagging : 0.9041095890410958



Accuracy score for random forest : 0.9834888729361091
Precision score random forest : 1.0
Recall score random forest : 0.8756756756756757
F1 score random forest : 0.9337175792507205



Accuracy score for adaboost : 0.9770279971284996
Precision score adaboost : 0.9693251533742331
Recall score adaboost : 0.8540540540540541
F1 score adaboost : 0.9080459770114943



Accuracy score for naive bayes : 0.9885139985642498
Precision score naive bayes : 0.9720670391061452
Recall score naive bayes : 0.9405405405405406
F1 score naive bayes : 0.9560439560439562
```

0.1.3 Recap

Now you have seen the whole process for a few ensemble models!

1. **Import** the model.
2. **Instantiate** the model with the hyperparameters of interest.
3. **Fit** the model to the training data.
4. **Predict** on the test data.
5. **Score** the model by comparing the predictions to the actual values.

This is a very common process for performing machine learning.

0.1.4 But, Wait...

You might be asking -

- What do these metrics mean?
- How do I optimize to get the best model?
- There are so many hyperparameters to each of these models, how do I figure out what the best values are for each?

This is exactly what the last two lessons of this course on supervised learning are all about.

```
In [ ]:
```