Our Mission

You recently used Naive Bayes to classify spam in this [dataset (https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection)](https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection). In this notebook, we will expand on the previous analysis by using a few of the new techniques you've learned throughout this lesson.

> Let's quickly re-create what we did in the previous Naive Bayes Spam Classifier notebook. We're providing the essential code from that previous workspace here, so please run this cell below.

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.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_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 = 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
```

Turns Out...

We can see from the scores above that our Naive Bayes model actually does a pretty good job of classifying spam and "ham." 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 (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.)
- [RandomForestClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ens](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ens)
- [AdaBoostClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensembl](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensembl)

Another really useful guide for ensemble methods can be found [in the documentation here (http://scikit-learn.org/stable/modules/ensemble.html)](http://scikit-learn.org/stable/modules/ensemble.html).

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.

This Process Looks Familiar...

In general, there is a five step process that can be used each time 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
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
```

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

bagging_model = BaggingClassifier(n_estimators=200)

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

forest_model = RandomForestClassifier(n_estimators=200)

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

ada_model = AdaBoostClassifier(n_estimators=300, learning_rate=.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 [10]:
```
# Fit your BaggingClassifier to the training data
bagging_fit = bagging_model.fit(training_data, y_train)

# Fit your RandomForestClassifier to the training data

foret_fit = forest_model.fit(training_data, y_train)

# Fit your AdaBoostClassifier to the training data

ada_fit = ada_model.fit(training_data, y_train)
```

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

In [11]:
```
# Predict using BaggingClassifier on the test data
bagging_preds = bagging_model.predict(testing_data)

# Predict using RandomForestClassifier on the test data

forest_preds = forest_model.predict(testing_data)

# Predict using AdaBoostClassifier on the test data

ada_preds = ada_model.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, so we can compare them all side by side.

In [13]:
```
def print_metrics(y_true, preds, model_name=None):
    '''
    INPUT:
    y_true - the y values that are actually true in the dataset (NumPy array
    preds - the predictions for those values from some model (NumPy array or
    model_name - (str - optional) a name associated with the model if you wo

    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_sc
        print('Precision score ' + model_name + ' :', format(precision_score
        print('Recall score ' + model_name + ' :', format(recall_score(y_tru
        print('F1 score ' + model_name + ' :', format(f1_score(y_true, preds
        print('\n\n')
```

In [16]:
```
# Print Bagging scores

print_metrics(y_test, bagging_preds, model_name = 'Bagging')

# Print Random Forest scores

print_metrics(y_test, forest_preds, model_name = 'Random Forest')

# Print AdaBoost scores

print_metrics(y_test, ada_preds, model_name = 'AdaBoost')

# Naive Bayes Classifier scores

print_metrics(y_test, predictions, model_name = 'Naive Bayes')
```

```
Accuracy score for Bagging : 0.9763101220387652
Precision score Bagging : 0.9269662921348315
Recall score Bagging : 0.8918918918918919
F1 score Bagging : 0.9090909090909092



Accuracy score for Random Forest : 0.9813352476669059
Precision score Random Forest : 1.0
Recall score Random Forest : 0.8594594594594595
F1 score Random Forest : 0.9244186046511628



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
```

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.

And that's it. This is a very common process for performing machine learning.

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.

Notice, you can obtain a solution to this notebook by clicking the orange icon in the top left!

In [ ]:

```
```