SMS SPAM Detection

A Project for fulfillment of AI with Machine Learning course

Hemang Khatri

Table of Contents

List of Figures:

1. INTRODUCTION

In today's world there is a big need of analyzing the data as it has lot of information that goes un-noticed. In order to detect such pattern, the Machine Learning techniques are used.

In this project, we are going to work with a dataset which has the data of SMS which are collected to classify them into SPAM or HAM. So, if a message is sent by a real user then it should be tagged as Ham or if it is by a machine for advertisement purpose then it should be tagged as SPAM.

1.1 PURPOSE

In daily life, we receive lot of messages from different sources and those messages might be arriving from a machine just for the advertisement purpose. The mobile user gets irritated because of such messages and may even ignore a real message from a known person. In order to avoid such cases, we are going to classify these SMS messages into two categories as SPAM or HAM.

1.2 SCOPE

The project is made on Anaconda Jupyter Notebook. This application is an easy to use application as the user need to run the complete program and the user will get the accuracy of each model.

Since, there is no un-labelled data, we are going to divide the data in Train and Test dataset. Test dataset is used for validating our model performance.
Also, in case in future we have some new dataset which is not labelled then we can classify them too.

1.3 OVERVIEW

This document will give you an easy walk over through the application and act as a guide with easy steps to use and maintain the application. Detailed overview of each feature and design is covered below in the System Overview. This application does not involve any database, but this is a future aspect of this application in case there is a need to store the labelled data, the flow of application is explained with data flow in use case diagram under System Architecture section. In the end, a visual look and feel of this application with the flow of application is shown. This application act as a perfect medium to classify the SMS messages with text data using NLP techniques.

2. SYSTEM OVERVIEW

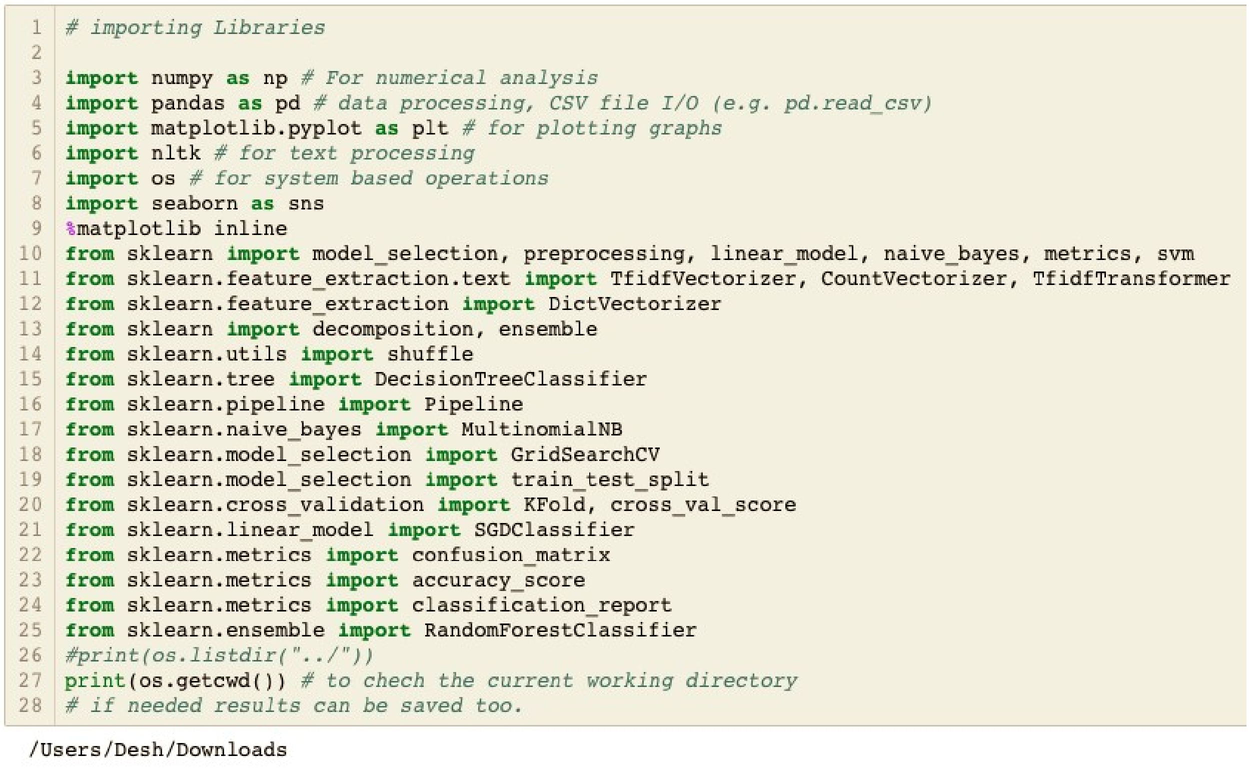
The project involves the text data which cannot be classified by basic modeling techniques and hence we are using NLP techniques. Natural Language Processing helps us to bridge the gap of text data with numerical data which is needed to run by the machine.

It also helps us to neutralize the dirty text data into a simpler form. We are removing Stop words punctuations, commonly appearing terms using TF-iDF (Term Frequency inverse Document Frequency).

The major steps that are involved in order to classify the data are as follows. Each step is described with Code Snippet.

Importing Libraries:

First thing we are going to do is to import the important libraries. Since we are working on Text data, we have imported NLTK for text analysis. Also, for data visualization, we have used matplotlib library.



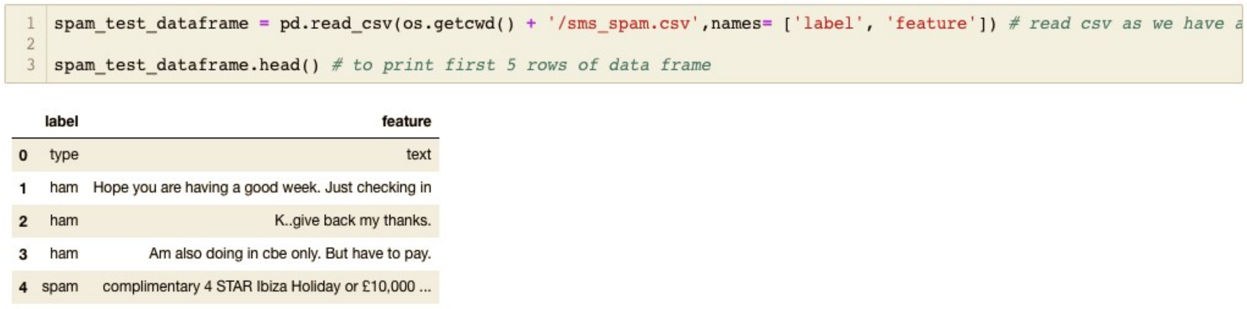
```python
# importing Libraries

import numpy as np # For numerical analysis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for plotting graphs
import nltk # for text processing
import os # for system based operations
import seaborn as sns
%matplotlib inline
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransformer
from sklearn.feature_extraction import DictVectorizer
from sklearn import decomposition, ensemble
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import KFold, cross_val_score
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
#print(os.listdir("../"))
print(os.getcwd()) # to chech the current working directory
# if needed results can be saved too.
```

```
/Users/Desh/Downloads
```

Figure 1: Importing Libraries

Read Dataset:

Next thing we need to do is to input the dataset we need to work on. The data is located in our current working directory which is Downloads in this case. The command dataframe.head() help us in checking the first 5 lines of our dataframe.



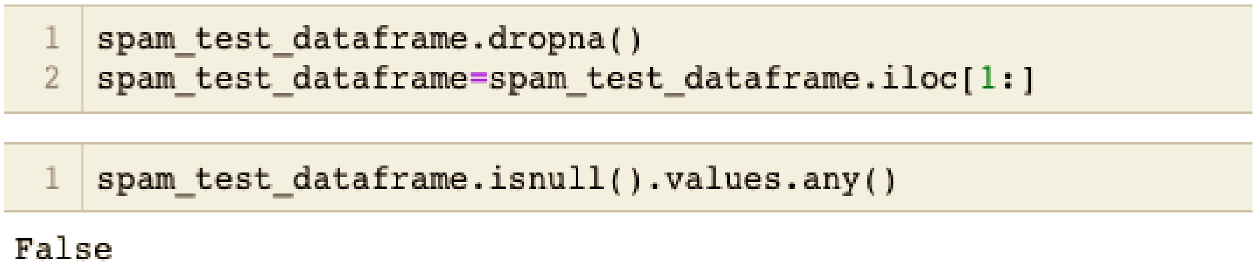
```
spam_test_dataframe = pd.read_csv(os.getcwd() + '/sms_spam.csv',names= ['label', 'feature']) # read csv as we have a

spam_test_dataframe.head() # to print first 5 rows of data frame
```

	label	feature
0	type	text
1	ham	Hope you are having a good week. Just checking in
2	ham	K..give back my thanks.
3	ham	Am also doing in cbe only. But have to pay.
4	spam	complimentary 4 STAR Ibiza Holiday or £10,000 ...

Figure 2: Reading the CSV File in a Data Frame

Remove NA values:

The biggest problem in any dataset is the NA values which needs to be handled very carefully. Since they are not going to add any meaning to our classification, we need to remove all the NA fields.



```
spam_test_dataframe.dropna()
spam_test_dataframe=spam_test_dataframe.iloc[1:]
```

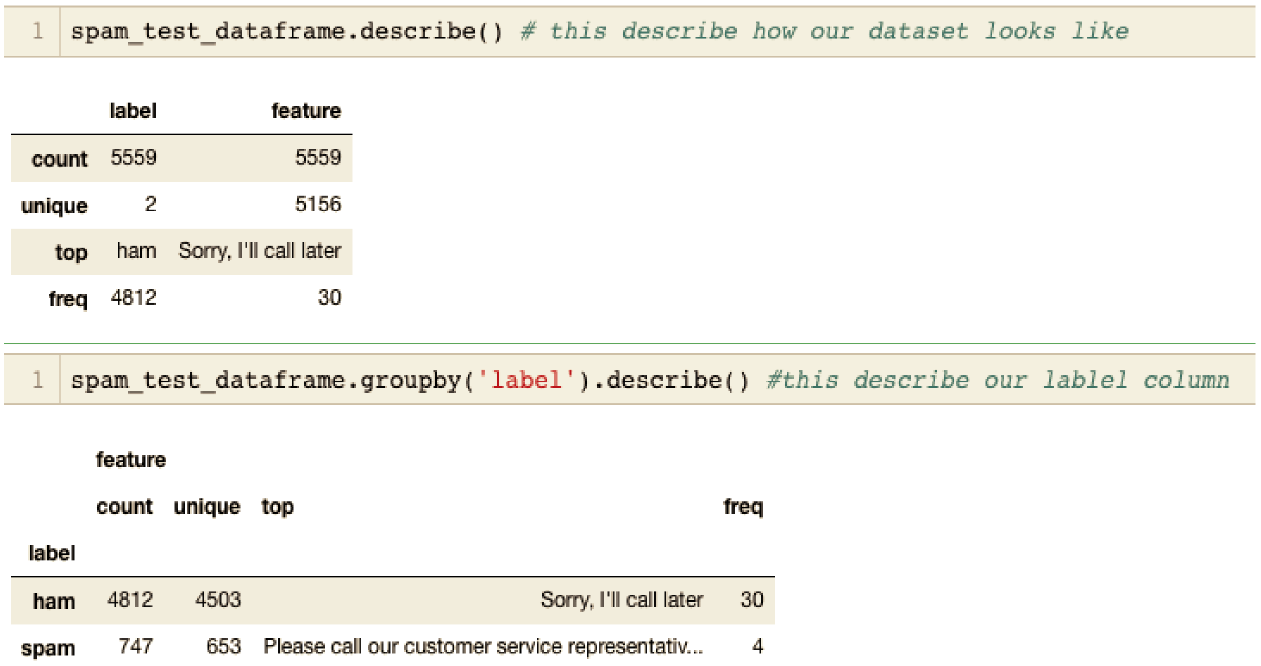
```
spam_test_dataframe.isnull().values.any()
```

```
False
```

Figure 3: Remove NA values from Data Frame

Describe the dataset and the label column:

The data should be understood correctly and hence, we are going to describe the whole dataset and also the label column.



```
spam_test_dataframe.describe() # this describe how our dataset looks like
```

	label	feature
count	5559	5559
unique	2	5156
top	ham	Sorry, I'll call later
freq	4812	30

```
spam_test_dataframe.groupby('label').describe() #this describe our lablel column
```

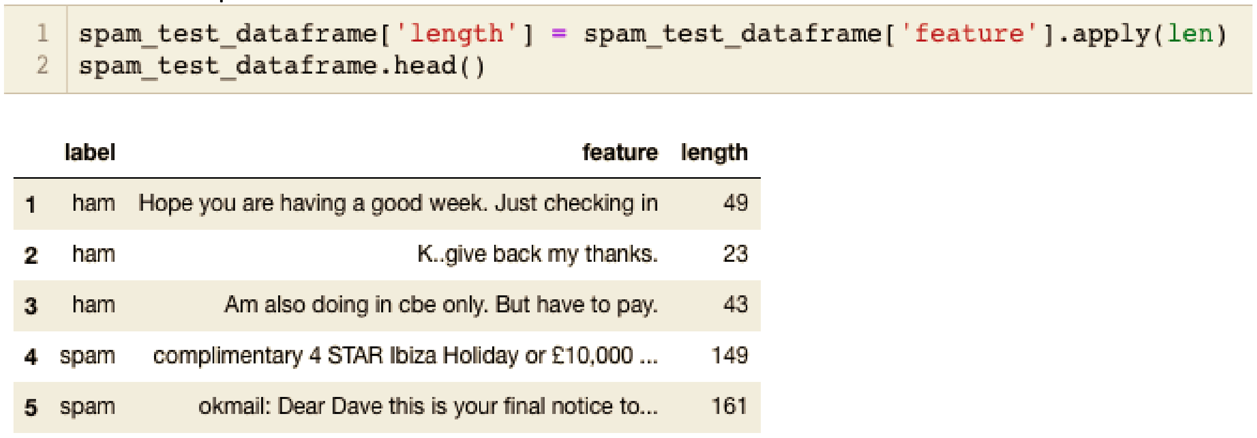
	feature			
	count	**unique**	**top**	**freq**
label				
ham	4812	4503	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

Figure 4: Describing the Dataset and the Label Set

This image clearly shows that our data has only two different kind of labels and also the count of unique values.

For Visualization – Adding new column:

In order to visualize our dataset, we need to add one column which shows the total number of words in that particular feature field.



```
spam_test_dataframe['length'] = spam_test_dataframe['feature'].apply(len)
spam_test_dataframe.head()
```

	label	feature	length
1	ham	Hope you are having a good week. Just checking in	49
2	ham	K..give back my thanks.	23
3	ham	Am also doing in cbe only. But have to pay.	43
4	spam	complimentary 4 STAR Ibiza Holiday or £10,000 ...	149
5	spam	okmail: Dear Dave this is your final notice to...	161

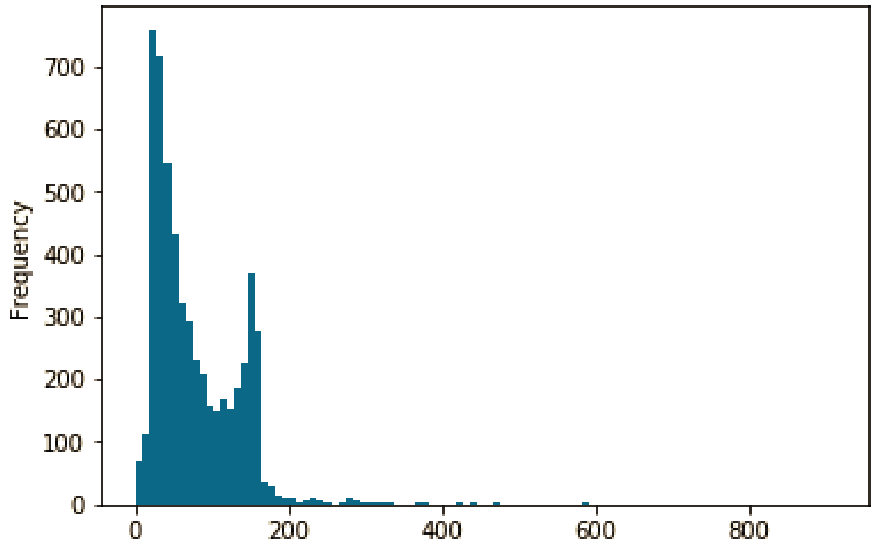
Figure 5: Adding new Column for Visualization

Basic Visualization of dataset:

Let's do a simple visualization of data before we start with machine learning.

```
spam_test_dataframe['length'].plot(bins=100, kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f01c0b8>
```



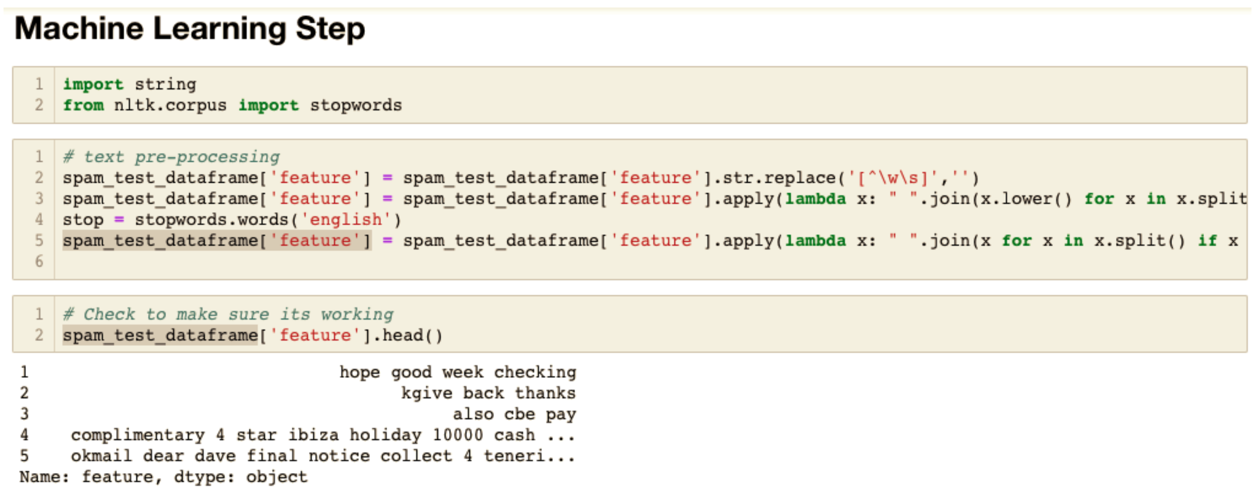
```
spam_test_dataframe.length.describe()
```

```
count    5559.000000
mean       79.781436
std        59.105497
min         2.000000
25%        35.000000
50%        61.000000
75%       121.000000
max       910.000000
Name: length, dtype: float64
```

Figure 6: Visualization

Machine Learning Steps – Basic Text pre-processing steps:

This step is our first important step and we are first going to do some text pre-processing in order to clean the dataset before we input it to our models.



Machine Learning Step

```
import string
from nltk.corpus import stopwords
```

```
# text pre-processing
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].str.replace('[^\w\s]','')
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].apply(lambda x: " ".join(x.lower() for x in x.split
stop = stopwords.words('english')
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].apply(lambda x: " ".join(x for x in x.split() if x

```

```
# Check to make sure its working
spam_test_dataframe['feature'].head()
```

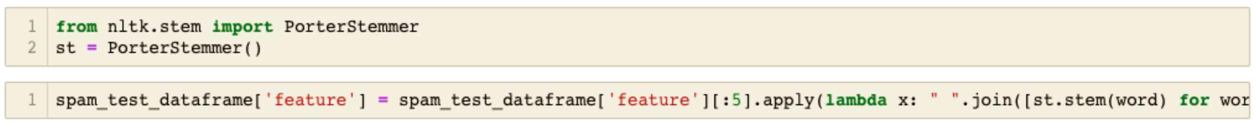
```
1                              hope good week checking
2                                     kgive back thanks
3                                          also cbe pay
4    complimentary 4 star ibiza holiday 10000 cash ...
5    okmail dear dave final notice collect 4 teneri...
Name: feature, dtype: object
```

Figure 7: Basic Text Pre-Processing

So, we have converted all the text into lower case and then we removed the punctuations from our dataset. Finally, we removed the stop words form our dataset.

Stemming each term using Porter Stemmer:

In order to make sure that our model predict the data better, we are going to stem our words into its stem form.



```
from nltk.stem import PorterStemmer
st = PorterStemmer()
```

```
spam_test_dataframe['feature'] = spam_test_dataframe['feature'][:5].apply(lambda x: " ".join([st.stem(word) for wor
```

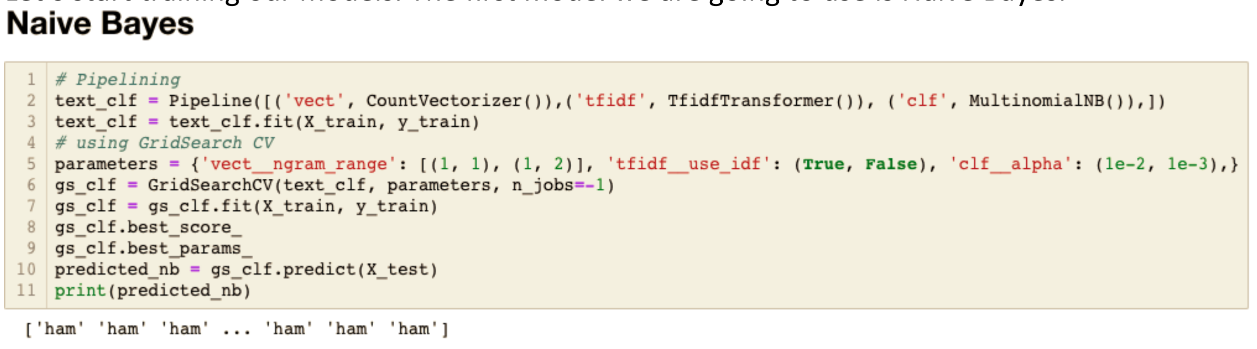
Figure 8: Stemming Feature set

Split the dataset into train and test:

Since, we do not have any train and test dataset split already, we are going to split the dataset into train and test. The test dataset is used for validating our models.

Training Models Naïve Bayes:

Let’s start training our models. The first model we are going to use is Naïve Bayes.



Naive Bayes

```
# Pipelining
text_clf = Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()), ('clf', MultinomialNB()),])
text_clf = text_clf.fit(X_train, y_train)
# using GridSearch CV
parameters = {'vect__ngram_range': [(1, 1), (1, 2)], 'tfidf__use_idf': (True, False), 'clf__alpha': (1e-2, 1e-3),}
gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
gs_clf = gs_clf.fit(X_train, y_train)
gs_clf.best_score_
gs_clf.best_params_
predicted_nb = gs_clf.predict(X_test)
print(predicted_nb)
```

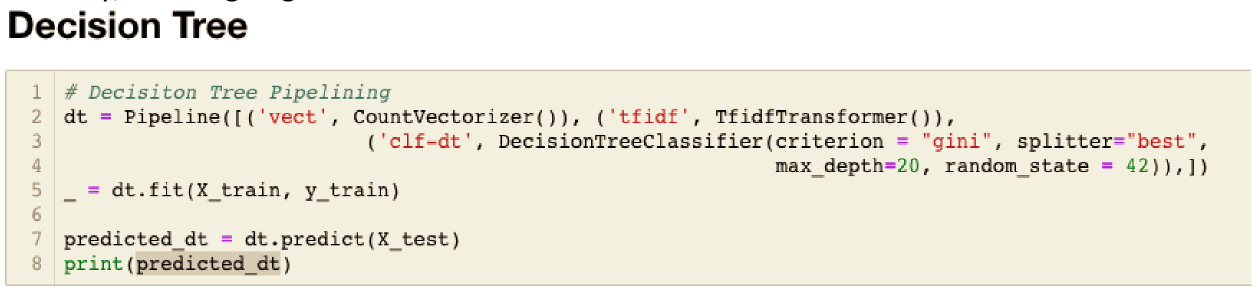
```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

Figure 9: Fit and Transform with Naive Bayes

In the last line, we have also shown the predicted values.

Decision Tree:

Similarly, we are going to train our data with Decision Tree.



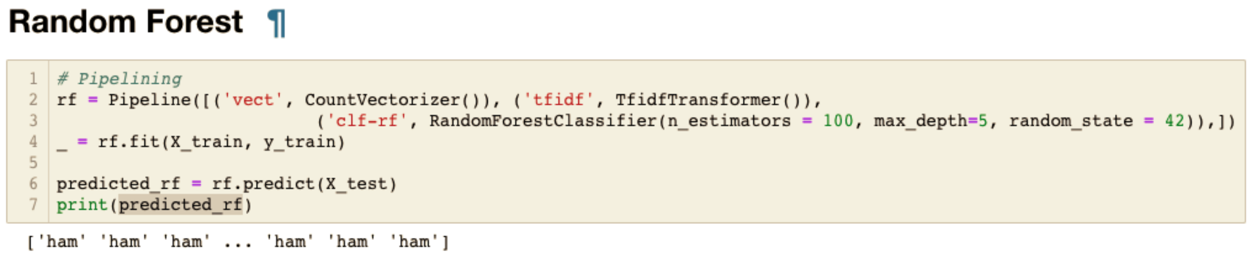
Decision Tree

```
# Decisiton Tree Pipelining
dt = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                         ('clf-dt', DecisionTreeClassifier(criterion = "gini", splitter="best",
                                                          max_depth=20, random_state = 42)),])
_ = dt.fit(X_train, y_train)

predicted_dt = dt.predict(X_test)
print(predicted_dt)
```

Figure 10: Fit and Transform with Decision Tree

Random Forest:

Also, we are going to use Random Forest as generally it gives higher accuracy then other models.



Random Forest ¶

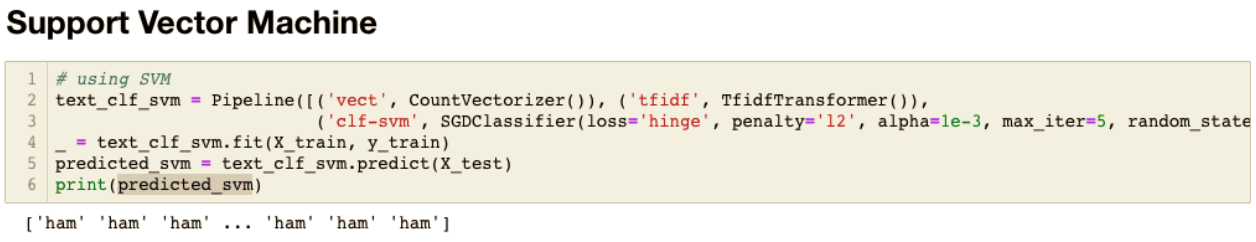
```
# Pipelining
rf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                     ('clf-rf', RandomForestClassifier(n_estimators = 100, max_depth=5, random_state = 42)),])
_ = rf.fit(X_train, y_train)

predicted_rf = rf.predict(X_test)
print(predicted_rf)
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

Figure 11: Fit and Transform with Random Forest

Support Vector Machine:

SVMs are always considered as good model for text classification. So, let's train the SVM too.



Support Vector Machine

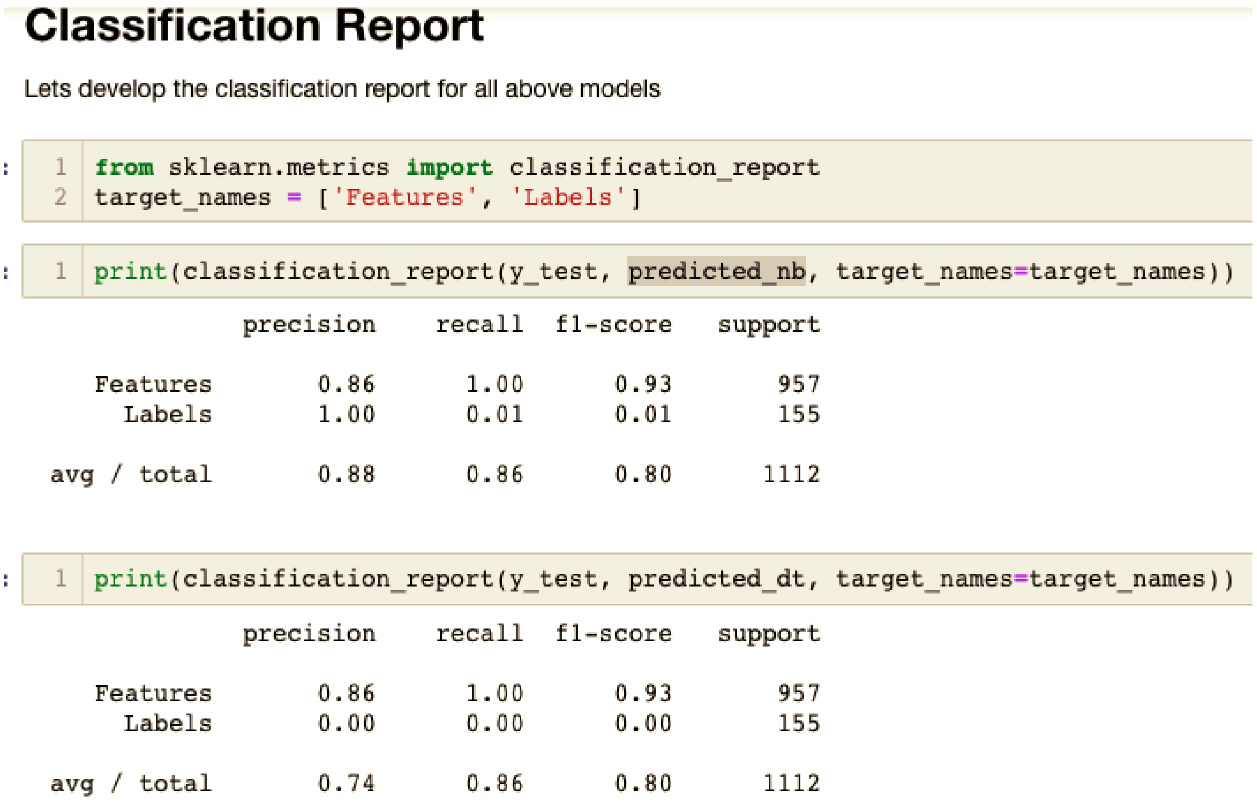
```
# using SVM
text_clf_svm = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                     ('clf-svm', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, max_iter=5, random_state
_ = text_clf_svm.fit(X_train, y_train)
predicted_svm = text_clf_svm.predict(X_test)
print(predicted_svm)
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

Figure 12: Fit and Transform with Support Vector Machine

Classification Report for all the models:

As we are done with all the models we wanted to use, let's check the classification report of these models.



Classification Report

Lets develop the classification report for all above models

```
from sklearn.metrics import classification_report
target_names = ['Features', 'Labels']
```

```
print(classification_report(y_test, predicted_nb, target_names=target_names))
```

```
             precision    recall  f1-score   support

   Features       0.86      1.00      0.93       957
     Labels       1.00      0.01      0.01       155

avg / total       0.88      0.86      0.80      1112
```

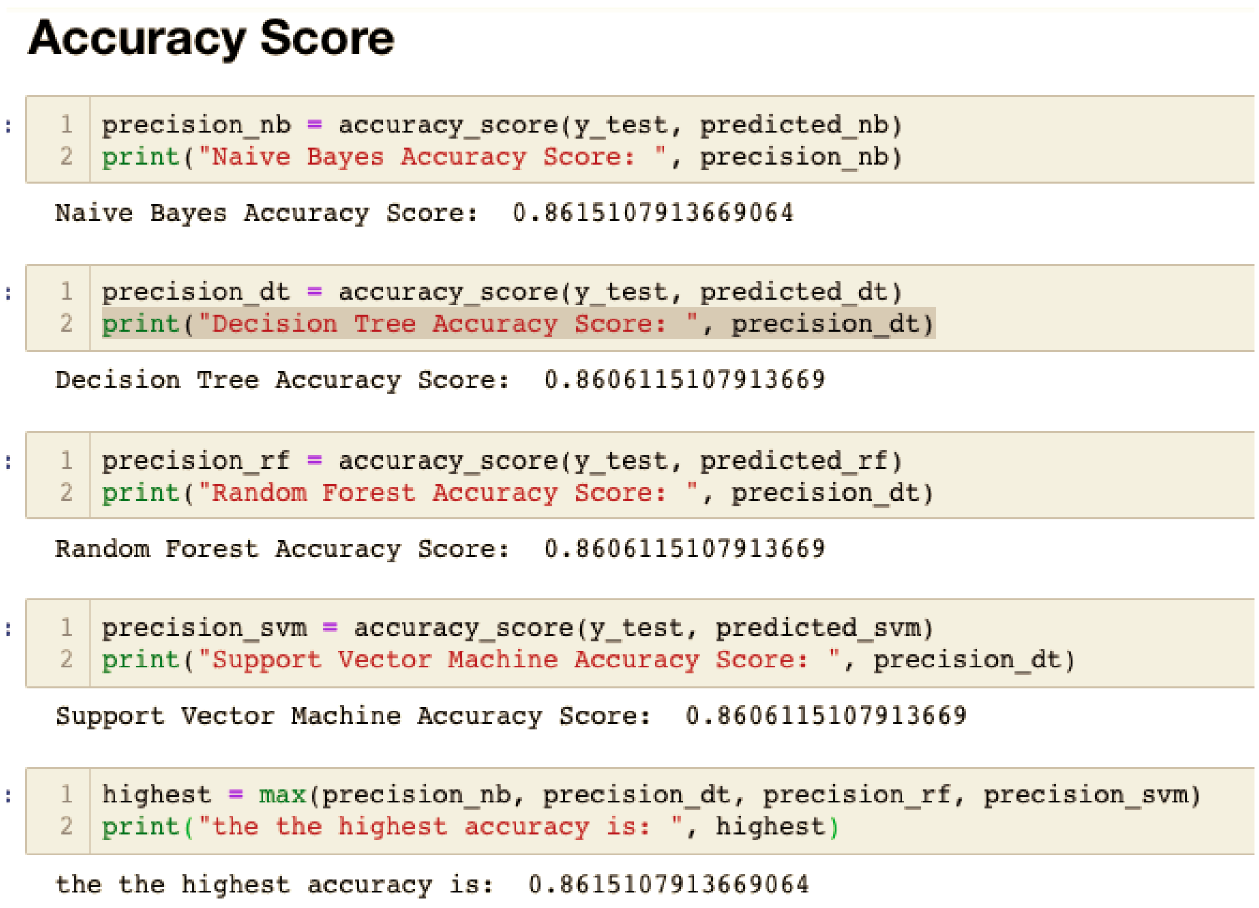
```
print(classification_report(y_test, predicted_dt, target_names=target_names))
```

```
             precision    recall  f1-score   support

   Features       0.86      1.00      0.93       957
     Labels       0.00      0.00      0.00       155

avg / total       0.74      0.86      0.80      1112
```

Figure 13: Checking Classification Report

Finally Checking the Accuracy Score:

Finally let’s check their accuracy score in order to check which model performed best amongst all.



Accuracy Score

```
precision_nb = accuracy_score(y_test, predicted_nb)
print("Naive Bayes Accuracy Score: ", precision_nb)
```

```
Naive Bayes Accuracy Score:  0.8615107913669064
```

```
precision_dt = accuracy_score(y_test, predicted_dt)
print("Decision Tree Accuracy Score: ", precision_dt)
```

```
Decision Tree Accuracy Score:  0.8606115107913669
```

```
precision_rf = accuracy_score(y_test, predicted_rf)
print("Random Forest Accuracy Score: ", precision_dt)
```

```
Random Forest Accuracy Score:  0.8606115107913669
```

```
precision_svm = accuracy_score(y_test, predicted_svm)
print("Support Vector Machine Accuracy Score: ", precision_dt)
```

```
Support Vector Machine Accuracy Score:  0.8606115107913669
```

```
highest = max(precision_nb, precision_dt, precision_rf, precision_svm)
print("the the highest accuracy is: ", highest)
```

```
the the highest accuracy is:  0.8615107913669064
```

Figure 14: Measuring Accuracy

So, we can see that SVM and Naïve Bayes has given the best accuracy although there is no much difference with other models.

This is a simple implementation of Machine Learning to classify the text data. The code is attached below for more details.

3. Code:

```
#!/usr/bin/env python
# coding: utf-8

# # SPAM SMS DETECTION

# **Input data files are available in the current working directory**
# **By running this files all the cells, we will be able to find the results of spam detection
with Natural language processing and performance of different models**
# **Natural Language Processing is  basically consists of combining machine learning
techniques with text, and using math and statistics to get that text in a format that the
machine learning algorithms can understand!**
#
# **NLTK should be installed, along with downloading the corpus for stopwords. Download
using nltk.download()**
#
# **Now, we are going to see NLP techniques to classify SMS into Ham or Spam using
different Machine Learning Models.**

# In[46]:


# importing Libraries

import numpy as np # For numerical analysis
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for plotting graphs
import nltk # for text processing
import os # for system based operations
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
#print(os.listdir("../"))
print(os.getcwd()) # to chech the current working directory
# if needed results can be saved too.


# ## Lets first get the Data in a Dataframe
# The dataset downloaded file contains a collection of more than 5 thousand SMS phone
messages.
#
```

```
# In[2]:


spam_test_dataframe = pd.read_csv(os.getcwd() + '/sms_spam.csv',names= ['label',
'feature']) # read csv as we have a csv file to read

spam_test_dataframe.head() # to print first 5 rows of data frame


# As we can see that first row contains header, we need to remove them first.

# In[3]:


spam_test_dataframe.dropna()
spam_test_dataframe=spam_test_dataframe.iloc[1:]


# In[4]:


spam_test_dataframe.isnull().values.any()


# In[5]:


spam_test_dataframe.head()


# ## Data Analysis
#
# First we will analyse the data which we have received and then we will do machine
learning on it.

# In[6]:


spam_test_dataframe.describe() # this describe how our dataset looks like


# In[7]:
```

```
spam_test_dataframe.groupby('label').describe() #this describe our lablel column


# As suspected length of message could bre really useful to identify the spam or a ham sms.
#
# Let us add another column called length of feature which will have how much does the
message length is.

# In[8]:


spam_test_dataframe['length'] = spam_test_dataframe['feature'].apply(len)
spam_test_dataframe.head()


# ### Data Visualization
# Lets Analyse the data before we do some machine learning

# In[9]:


spam_test_dataframe['length'].plot(bins=100, kind='hist')


# In[11]:


spam_test_dataframe.length.describe()


# So the message with longest length is of 910 characters

# In[12]:


spam_test_dataframe[spam_test_dataframe['length'] == 910]['feature'].iloc[0]


# So this SMS message was sent by one person to other personally so it is ham. But this
does not help much in Ham Spam identification

# In[13]:
```

```
spam_test_dataframe.hist(column='length', by='label', bins=50,figsize=(12,4))


# Our data is text data so first it should be in a vector format which is then input to machine learning model.
# In this section we'll convert the raw messages (sequence of characters) into vectors (sequences of numbers).
#
# Step1: Do some preprocessing like removing punctuation, stop words etc.
# Step2: Do some advance text processing like converting to bag of words, N-gram etc.
# Step3: Machine Learning model fit adn transform
# Step4: Model accuracy check.
# Let's first start with with step 1 and then rest will follow.

# # Machine Learning Step

# In[14]:


import string
from nltk.corpus import stopwords


# In[15]:


# text pre-processing
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].str.replace('[^\w\s]','')
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].apply(lambda x: " ".join(x.lower() for x in x.split()))
stop = stopwords.words('english')
spam_test_dataframe['feature'] = spam_test_dataframe['feature'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))


# In[16]:


# Check to make sure its working
spam_test_dataframe['feature'].head()


# Lets do something even better beside above techniques, lets shorten the terms to their stem form.
# In[17]:


from nltk.stem import PorterStemmer
st = PorterStemmer()


# In[18]:


spam_test_dataframe['feature'] = spam_test_dataframe['feature'][:5].apply(lambda x: "
".join([st.stem(word) for word in x.split()]))


# Each vector will have as many dimensions as there are unique words in the SMS corpus.
We will first use SciKit Learn's **CountVectorizer**. This model will convert a collection of
text documents to a matrix of token counts.
#
# We can imagine this as a 2-Dimensional matrix. Where the 1-dimension is the entire
vocabulary (1 row per word) and the other dimension are the actual documents, in this case
a column per text message.
#
# For example:
#
# <table border = “1“>
# <tr>
# <th></th> <th>Message 1</th> <th>Message 2</th> <th>...</th> <th>Message N</th>
# </tr>
# <tr>
# <td><b>Word 1 Count</b></td><td>0</td><td>1</td><td>...</td><td>0</td>
# </tr>
# <tr>
# <td><b>Word 2 Count</b></td><td>0</td><td>0</td><td>...</td><td>0</td>
# </tr>
# <tr>
# <td><b>...</b></td> <td>1</td><td>2</td><td>...</td><td>0</td>
# </tr>
# <tr>
# <td><b>Word N Count</b></td> <td>0</td><td>1</td><td>...</td><td>1</td>
# </tr>
# </table>
#
#
```

```
# Since there are so many messages, we can expect a lot of zero counts for the presence of that word in that document. Because of this, SciKit Learn will output a [Sparse Matrix](https://en.wikipedia.org/wiki/Sparse_matrix).

# In[19]:


from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransformer
from sklearn.feature_extraction import DictVectorizer
from sklearn import decomposition, ensemble
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import KFold, cross_val_score
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier


# we will use pipelines to make the steps short.
# The pipeline will do the vectorization, Term frequcz transformation and model fitting together.

# After the counting, the term weighting and normalization can be done with [TF-IDF](http://en.wikipedia.org/wiki/Tf%E2%80%93idf), using scikit-learn's `TfidfTransformer`.
#
# ____
# ### So what is TF-IDF?
# TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.
#
```

# One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.
#
# Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.
#
# **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
#
# *TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).*
#
# **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
#
# *IDF(t) = log_e(Total number of documents / Number of documents with term t in it).*
#
# See below for a simple example.
#
# **Example:**
#
# Consider a document containing 100 words wherein the word cat appears 3 times.
#
# The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.
____
#
# Let's go ahead and see how we can do this in SciKit Learn:
# To transform the entire bag-of-words corpus into TF-IDF corpus at once:

Train Test Split

# In[22]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(spam_test_dataframe['feature'].values.astype('U'),
spam_test_dataframe['label'], test_size=0.2, random_state=1)

print(len(X_train), len(X_test), len(y_train) + len(y_test))


# The test size is 20% of the entire dataset (1112 messages out of total 5559), and the
training is the rest (4447 out of 5559). Note the default split would have been 30/70.
#
# ## Creating a Data Pipeline
#
# Let's run our model again and then predict off the test set. We will use SciKit Learn's
[pipeline](http://scikit-learn.org/stable/modules/pipeline.html) capabilities to store a
pipeline of workflow. This will allow us to set up all the transformations that we will do to
the data for future use. Let's see an example of how it works:

# ## Training a model
#
# With messages represented as vectors, we can finally train our spam/ham classifier. Now
we can actually use almost any sort of classification algorithms. For a [variety of reasons]
(http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn-note07-2up.pdf),
the Naive Bayes classifier algorithm is a good choice.
#
# Using scikit-learn here, choosing the Naive Bayes, Decision Tree, Random Forest, Support
Vector Machine classifiers to start with:
# In the end we will compare the accuracy of each model.
#

# # Naive Bayes

# In[32]:


# Pipelining
text_clf = Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()), ('clf',
MultinomialNB()),])
text_clf = text_clf.fit(X_train, y_train)
# using GridSearch CV
```

```
parameters = {'vect__ngram_range': [(1, 1), (1, 2)], 'tfidf__use_idf': (True, False),
'clf__alpha': (1e-2, 1e-3),}
gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
gs_clf = gs_clf.fit(X_train, y_train)
gs_clf.best_score_
gs_clf.best_params_
predicted_nb = gs_clf.predict(X_test)
print(predicted_nb)


# Lets build other models too as promised earlier.
#
# # Decision Tree

# In[25]:



# Decisiton Tree Pipelining
dt = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
               ('clf-dt', DecisionTreeClassifier(criterion = "gini", splitter="best",
                                    max_depth=20, random_state = 42)),])
_ = dt.fit(X_train, y_train)

predicted_dt = dt.predict(X_test)
print(predicted_dt)


# # Random Forest

# In[26]:



# Pipelining
rf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
               ('clf-rf', RandomForestClassifier(n_estimators = 100, max_depth=5,
random_state = 42)),])
_ = rf.fit(X_train, y_train)

predicted_rf = rf.predict(X_test)
print(predicted_rf)


# # Support Vector Machine
# In[27]:


# using SVM
text_clf_svm = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                    ('clf-svm', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, max_iter=5,
random_state=42)),])
_ = text_clf_svm.fit(X_train, y_train)
predicted_svm = text_clf_svm.predict(X_test)
print(predicted_svm)


# # Classification Report
# Lets develop the classification report for all above models

# In[29]:


from sklearn.metrics import classification_report
target_names = ['Features', 'Labels']


# In[33]:


print(classification_report(y_test, predicted_nb, target_names=target_names))


# In[34]:


print(classification_report(y_test, predicted_dt, target_names=target_names))


# In[35]:


print(classification_report(y_test, predicted_rf, target_names=target_names))


# In[36]:


print(classification_report(y_test, predicted_svm, target_names=target_names))
```

```
# # Accuracy Score

# In[41]:


precision_nb = accuracy_score(y_test, predicted_nb)
print("Naive Bayes Accuracy Score: ", precision_nb)


# In[42]:


precision_dt = accuracy_score(y_test, predicted_dt)
print("Decision Tree Accuracy Score: ", precision_dt)


# In[43]:


precision_rf = accuracy_score(y_test, predicted_rf)
print("Random Forest Accuracy Score: ", precision_dt)


# In[44]:


precision_svm = accuracy_score(y_test, predicted_svm)
print("Support Vector Machine Accuracy Score: ", precision_dt)


# In[45]:


highest = max(precision_nb, precision_dt, precision_rf, precision_svm)
print("the the highest accuracy is: ", highest)


# **So our model predicted very well on the dataset with an accuracy about 86%. If we fine
tune our model, our accuracy could increase. **
```