Spam Mini-Project

CECS 621: Web Mining for E-Commerce and Information Retrieval

Date: 05 April, 2018

Rajesh Sikder

Introduction:

Spam is a constant, unwanted facet of life in digital age. We are constantly inundated with more information than we can process by the solicited sources of information in our lives, but when spam is added to the situation, it can be unbearable at times. For the purposes of this report, "spam" refers to unwanted and unsolicited information. Spam is often the result of advertisers buying a user's contact information, and sending them mass generated emails or the like. Often spam items are annoying, but essentially innocuous. However, some spam items contain viruses. Therefore, effective spam filters are an essential feature of many digital platforms.

The goal of this project was to implement a spam classification process so that spam items could be reliably identified. The data was classified as either spam, or ham. For the purposes of this report, "ham" is defined as a text document that was useful to the user, and originating from a known or friendly source. In order to parse the data into spam or ham, the data was first pre-processed to allow it to be efficiently handled in the Jupyter iPython notebook. Next, two algorithms were utilized. The first algorithm utilized was the Naive Bayes process within the Scikit-Learn library in Python, called MultinomialNB. The second algorithm utilized was Support Vector Machine process in the Scikit-Learn library. Both algorithms were able to process the data, and accurately identify the spam documents.

The Naive Bayes classifier algorithm is a very useful approach to categorizing text. The basic Naive Bayes formula can be expressed as such:

$$
\begin{aligned}
p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) = \\
&= p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\
&= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,.
\end{aligned}
$$

In the above formula, ∝ denote proportionality. Given this formula, the conditional distribution over the class variable C can be explained as:

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

For this particular dataset, Multinomial Naive Bayes was utilized since it is specialized to deal with text documents. Therefore, it is useful to consider the formula for the Multinomial Naive Bayes algorithm as well:

$$\begin{aligned} \log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k) \prod_{i=1}^{n} {p_{ki}}^{x_i} \right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$.

While the standard Naive Bayes algorithm is suited to deal with frequencies and word counts within a collection of documents, Multinomial Naive Bayes allows for explicit modeling on word counts, and thus is ideally suited to work with datasets such as the one discussed here.

The Support Vector Machine algorithm was also implemented to analyze this dataset. Unlike Naive Bayes, which deals with probabilities when classifying words and documents, the Support Vector Machine approach, analyzes the words in relation to other words in the document. The Support Vector Machine algorithm rearranged the words along a new hyper plane in order to analyze them in a way that is not apparent from the unprocessed dataset. When dealing with a soft margin, the formula can be explained as:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-y_i(\vec{w}\cdot\vec{x}_i-b)\right)\right]+\lambda\|\vec{w}\|^2,$$

where ג (lambda) determines whether to increase the size of the margin on the hyper-plane.

Spam1 Dataset:

The Spam1 dataset was comprised of 5,572 documents. Each document was assigned either the class of "spam" or "ham," as noted above. Out of the total number of documents, 4,825, or 86.5%, were categorized as ham. The dataset was preprocessed by utilizing the Natural Language Toolkit Stopwords function to remove stop words and punctuation. The function that was implemented is show here:

```
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

Additionally, vectorization was utilized in order to transform the data so that the algorithms were able to process it. Below is the code for that function:

```
#Spam Vectorizer
spam_transformer = CountVectorizer(analyzer = text_process).fit(x)
len(spam_transformer.vocabulary_)
```

After the dataset was vectorized, there were 11,301 vectors created. Finally, the classes were transformed into binary for additional ease with the classification process.

Following the descriptive processes and the pre-processing steps, the Multinomial Naive Bayes and Support Vector Machine processes were applied to the Spam1 dataset. Both code snippets are shown below:

```
#Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)
preds = nb.predict(x_test)
probas = nb.predict_proba(x_test)
```

```
#Support Vector Machine
list_C = np.arange(500, 2000, 100) #100000
score_train = np.zeros(len(list_C))
score_test = np.zeros(len(list_C))
recall_test = np.zeros(len(list_C))
precision_test= np.zeros(len(list_C))
count = 0
for C in list_C:
    svc = svm.SVC(C=C)
    svc.fit(x_train, y_train)
    score_train[count] = svc.score(x_train, y_train)
    score_test[count]= svc.score(x_test, y_test)
    recall_test[count] = metrics.recall_score(y_test, svc.predict(x_test))
    precision_test[count] = metrics.precision_score(y_test, svc.predict(x_test))
    count = count + 1
```

Spam2 Dataset:

Significantly smaller than the Spam1 dataset, the Spam2 dataset was comprised of 40 documents. Each document was assigned either the class of "spam" or "ham," as noted above. Out of the total number of documents, 20, or 50%, were categorized as ham. The function utilized to read and import the dataset was edited to load the Spam2 document, and the same functions were implemented to analyze Spam2 as were implemented to analyze Spam1. It is important to note that since this dataset was much smaller with regard to the number of documents, the results of the analyses were not as encouraging.

Results:

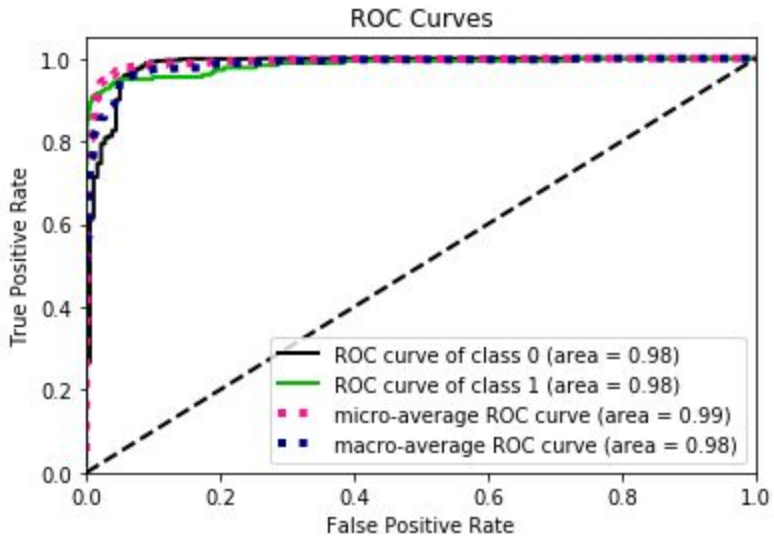
The Spam1 dataset was split into train and test sections in order to allow the Multinomial Naive Bayes and Support Vector Machine algorithms to have supervised learning with the dataset. After the models were trained, the test section was processed, and the results were displayed. The Multinomial

Naive Bayes confusion matrix code and output are below:

```
#Naive Bayes Confusion Matrix
nb_confusion = metrics.confusion_matrix(y_test, preds)
pd.DataFrame(data = nb_confusion, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
```

	Predicted 0	Predicted 1
Actual 0	1171	43
Actual 1	12	167

In the matrix above, 0 is a message classified as ham, and 1 is a message classified as spam. The vast majority of legitimate ham documents were identified as such. Likewise with the spam documents. The ROC curve below provides a visual representation of the sensitivity and specificity of the Multinomial Naive Bayes algorithm.



However, twelve documents were predicted to be ham and were actually spam documents. This may only indicate a slight inconvenience for the user. The more alarming number is that forty-three documents were ham, and were predicted to be spam messages. As anyone who has ever missed an important email can attest, it is never desirable to have a legitimate message be caught by the spam

filter. In order to improve this algorithm in the future, specificity with regard to "ham" messages may be preferable. For most users, it is preferable to have a few spam messages get through than it is for legitimate ham messages to end up in the spam folder.

The Support Vector Machine process provided an additional prospective with regard to the Spam1 dataset. The accuracy, precision and recall output below provides insight into how the Support Vector Machine algorithm handled the Spam1 dataset:

```
best_index = models['Test Precision'].idxmax()
models.iloc[best_index, :]
```

```
C                 500.000000
Train Accuracy      0.991864
Test Accuracy       0.976310
Test Recall         0.815642
Test Precision      1.000000
Name: 0, dtype: float64
```

The output above indicates that the Support Vector Machine algorithm has a high rate of precision and accuracy, but that the recall could be improved upon. The confusion matrix below suggests that despite the recall being less than 85%, no ham messages were mistakenly predicted as spam:

```
#Support Vector Machine Confusion Matrix
svm_confusion = metrics.confusion_matrix(y_test, svc.predict(x_test))
pd.DataFrame(data = svm_confusion, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
```

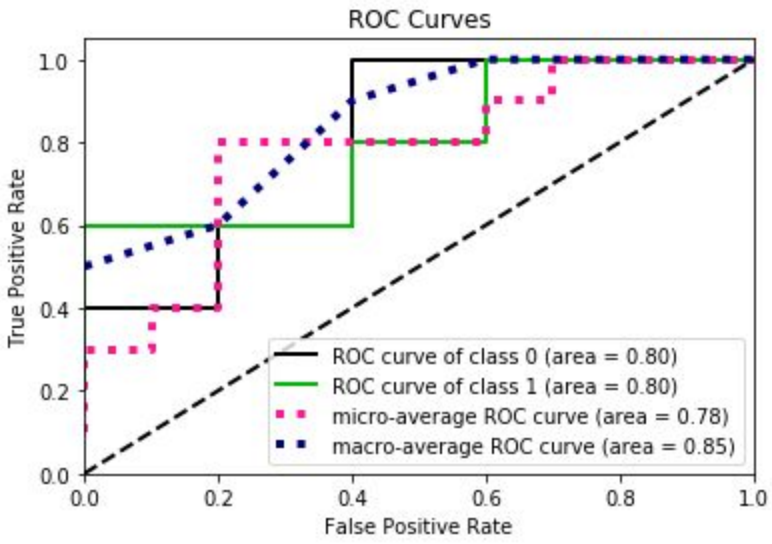
	Predicted 0	Predicted 1
Actual 0	1214	0
Actual 1	25	154

The Support Vector Machine process has a higher specificity than does the Multinomial Naive Bayes algorithm. Therefore, it is more suitable to being applied to a spam filter. Although some spam

documents may show up in a user's inbox, it is unlikely that a significant number of their “ham” messages will be lost to the spam folder.

Given the size of the dataset for Spam2, the results are less impressive for both algorithms. However, they are still worth mentioning. The confusion matrix and ROC curve for the Multinomial Naive Bayes email are below:

	Predicted 0	Predicted 1
Actual 0	5	0
Actual 1	2	3



Although the dataset is small, the Multinomial Naive Bayes algorithm demonstrates a suitable level of specificity. No ham messages were categorized as spam. The Support Vector Machine algorithm similarly processed the Spam2 dataset with high specificity.

```
C                 500.0
Train Accuracy      1.0
Test Accuracy       0.7
Test Recall         0.4
Test Precision      1.0
Name: 0, dtype: float64
```

	Predicted 0	Predicted 1
Actual 0	5	0
Actual 1	3	2

Conclusion:

Spam is a constant in our digital lives. It is unlikely to ever go away, and will likely increase as more and more advertisers have access to large dataset and sophisticated data mining algorithms. Therefore, effective spam filters are a necessity. The algorithms discussed here provide some insight into how to implement a spam classification system, and also demonstrate some of the pitfalls of creating an algorithm that does not have enough specificity or sensitivity. The full code is attached at the end of this report as an appendix.

In [255]:
```
import string
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
import pandas as pd
from pandas import DataFrame
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import feature_extraction, model_selection, naive_bayes, metrics, svm
import scikitplot as skplt
import matplotlib.pyplot as plt
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Brent\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [237]:
```
spam = pd.read_csv('spam1.csv')
spam.describe()
```

Out[237]:

	Class	Text
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

In [238]:
```
spam.head(n=10)
```

Out[238]:

	Class	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? U R entitle...

In [239]:
```
spam_class = spam[(spam['Class'] == 'ham') | (spam['Class'] == 'spam')]
x = spam_class['Text']
y = spam_class['Class']
```

In [240]:
```
#Function to remove stop words
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stopwords.words
('english')]
```

In [241]:
```
#Spam Vectorizer
spam_transformer = CountVectorizer(analyzer = text_process).fit(x)
len(spam_transformer.vocabulary_)
```

Out[241]: 11301

In [242]:
```
x = spam_transformer.transform(x)
print('Shape of Sparse Matrix: ', x.shape)
print('Amount of Non-zero occurences: ', x.nnz)

#percentage of non-zero values
density = (100.0 * x.nnz / (x.shape[0] * x.shape[1]))
print('Density: {}'.format((density)))
```

```
Shape of Sparse Matrix:  (5572, 11301)
Amount of Non-zero occurences:  50145
Density: 0.07963420576659322
```

In [243]:
```
#Split dataset to create test and train sections
spam["Class"]=spam["Class"].map({'spam':1,'ham':0})
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, spam['Class
'], test_size=0.25, random_state=101)
print([np.shape(x_train), np.shape(x_test)])
```

```
[(4179, 11301), (1393, 11301)]
```

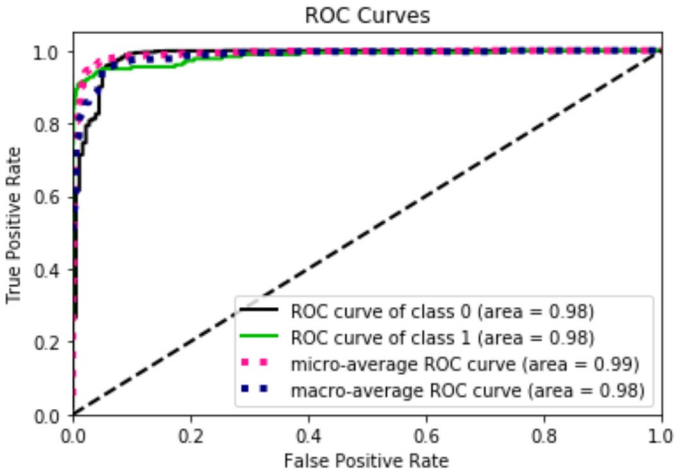
In [244]:
```
#Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)
preds = nb.predict(x_test)
probas = nb.predict_proba(x_test)
```

In [253]:
```
#Naive Bayes Confusion Matrix
nb_confusion = metrics.confusion_matrix(y_test, preds)
pd.DataFrame(data = nb_confusion, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
```

Out[253]:

	Predicted 0	Predicted 1
Actual 0	1171	43
Actual 1	12	167

In [246]:
```
#ROC Curve
skplt.metrics.plot_roc_curve(y_test, probas)
plt.show()
```



In [247]:
```
#Support Vector Machine
list_C = np.arange(500, 2000, 100) #100000
score_train = np.zeros(len(list_C))
score_test = np.zeros(len(list_C))
recall_test = np.zeros(len(list_C))
precision_test= np.zeros(len(list_C))
count = 0
for C in list_C:
    svc = svm.SVC(C=C)
    svc.fit(x_train, y_train)
    score_train[count] = svc.score(x_train, y_train)
    score_test[count]= svc.score(x_test, y_test)
    recall_test[count] = metrics.recall_score(y_test, svc.predict(x_test))
    precision_test[count] = metrics.precision_score(y_test, svc.predict(x_test))
    count = count + 1
```

In [248]:
```
matrix = np.matrix(np.c_[list_C, score_train, score_test, recall_test, precision_t
est])
models = pd.DataFrame(data = matrix, columns =
             ['C', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precisi
on'])
models.head(n=10)
```

Out[248]:

	C	Train Accuracy	Test Accuracy	Test Recall	Test Precision
0	500.0	0.991864	0.976310	0.815642	1.0
1	600.0	0.992343	0.979182	0.837989	1.0
2	700.0	0.993061	0.980617	0.849162	1.0
3	800.0	0.994496	0.980617	0.849162	1.0
4	900.0	0.996889	0.980617	0.849162	1.0
5	1000.0	0.996889	0.979899	0.843575	1.0
6	1100.0	0.997368	0.979899	0.843575	1.0
7	1200.0	0.997368	0.979899	0.843575	1.0
8	1300.0	0.997607	0.979899	0.843575	1.0
9	1400.0	0.997846	0.980617	0.849162	1.0

In [249]:
```
best_index = models['Test Precision'].idxmax()
models.iloc[best_index, :]
```

Out[249]:
```
C                 500.000000
Train Accuracy      0.991864
Test Accuracy       0.976310
Test Recall         0.815642
Test Precision      1.000000
Name: 0, dtype: float64
```

In [251]:
```
best_index = models[models['Test Precision']==1]['Test Accuracy'].idxmax()
svc = svm.SVC(C=list_C[best_index])
svc.fit(x_train, y_train)
models.iloc[best_index, :]
```

Out[251]:
```
C                 1500.000000
Train Accuracy       0.998325
Test Accuracy        0.982053
Test Recall          0.860335
Test Precision       1.000000
Name: 10, dtype: float64
```

In [254]:
```
#Support Vector Machine Confusion Matrix
svm_confusion = metrics.confusion_matrix(y_test, svc.predict(x_test))
pd.DataFrame(data = svm_confusion, columns = ['Predicted 0', 'Predicted 1'],
            index = ['Actual 0', 'Actual 1'])
```

Out[254]:

	Predicted 0	Predicted 1
Actual 0	1214	0
Actual 1	25	154

In [22]:
```
import string
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
import pandas as pd
from pandas import DataFrame
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import feature_extraction, model_selection, naive_bayes, metrics, svm
import scikitplot as skplt
import matplotlib.pyplot as plt
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Brent\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [23]:
```
spam = pd.read_csv('Spam2.csv')
spam.describe()
```

Out[23]:

	Class	Text
count	40	40
unique	2	40
top	spam	I will look over them at lunch and see what I ...
freq	20	1

In [24]:
```
spam.head(n=10)
```

Out[24]:

	Class	Text
0	ham	You can mention CardSmart access in the Additi...
1	spam	A quick, simple DIY setup means no costly inst...
2	ham	I am in my office today
3	spam	See all Toyota special offers including offers...
4	ham	We will be shutting down all the CS non produc...
5	ham	They may need to go in and complete it prior t...
6	spam	Professionally monitored security by triple re...
7	ham	The following test has been made available in ...
8	spam	Your trade in quote will be from the highly re...
9	spam	These innovative features were designed to hel...

In [25]:
```
spam_class = spam[(spam['Class'] == 'ham') | (spam['Class'] == 'spam')]
x = spam_class['Text']
y = spam_class['Class']
```

In [26]:
```
#Function to remove stop words
def text_process(text):
    nopunc = [char for char in text if char not in string.punctuation]
    nopunc = ''.join(nopunc)
    return [word for word in nopunc.split() if word.lower() not in stopwords.words(
'english')]
```

In [27]:
```
#Spam Vectorizer
spam_transformer = CountVectorizer(analyzer = text_process).fit(x)
len(spam_transformer.vocabulary_)
```

Out[27]: 249

In [28]:
```
x = spam_transformer.transform(x)
print('Shape of Sparse Matrix: ', x.shape)
print('Amount of Non-zero occurences: ', x.nnz)

#percentage of non-zero values
density = (100.0 * x.nnz / (x.shape[0] * x.shape[1]))
print('Density: {}'.format((density)))
```

```
Shape of Sparse Matrix:  (40, 249)
Amount of Non-zero occurences:  300
Density: 3.0120481927710845
```

In [29]:
```
#Split dataset to create test and train sections
spam["Class"]=spam["Class"].map({'spam':1,'ham':0})
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, spam['Class'
], test_size=0.25, random_state=101)
print([np.shape(x_train), np.shape(x_test)])
```

```
[(30, 249), (10, 249)]
```

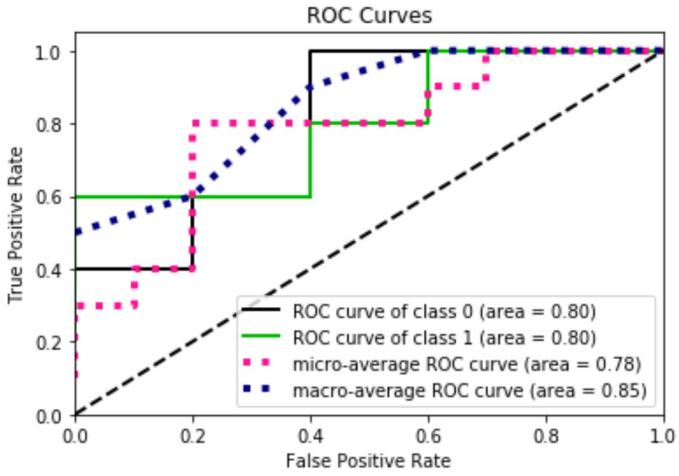
In [30]:
```
#Multinomial Naive Bayes
nb = MultinomialNB()
nb.fit(x_train, y_train)
preds = nb.predict(x_test)
probas = nb.predict_proba(x_test)
```

In [31]:
```
#Naive Bayes Confusion Matrix
nb_confusion = metrics.confusion_matrix(y_test, preds)
pd.DataFrame(data = nb_confusion, columns = ['Predicted 0', 'Predicted 1'],
             index = ['Actual 0', 'Actual 1'])
```

Out[31]:

	Predicted 0	Predicted 1
Actual 0	5	0
Actual 1	2	3

In [32]:
```
#ROC Curve
skplt.metrics.plot_roc_curve(y_test, probas)
plt.show()
```



In [33]:
```
#Support Vector Machine
list_C = np.arange(500, 2000, 100) #100000
score_train = np.zeros(len(list_C))
score_test = np.zeros(len(list_C))
recall_test = np.zeros(len(list_C))
precision_test= np.zeros(len(list_C))
count = 0
for C in list_C:
    svc = svm.SVC(C=C)
    svc.fit(x_train, y_train)
    score_train[count] = svc.score(x_train, y_train)
    score_test[count]= svc.score(x_test, y_test)
    recall_test[count] = metrics.recall_score(y_test, svc.predict(x_test))
    precision_test[count] = metrics.precision_score(y_test, svc.predict(x_test))
    count = count + 1
```

In [34]:
```
matrix = np.matrix(np.c_[list_C, score_train, score_test, recall_test, precision_te
st])
models = pd.DataFrame(data = matrix, columns =
             ['C', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precisio
n'])
models.head(n=10)
```

Out[34]:

	C	Train Accuracy	Test Accuracy	Test Recall	Test Precision
0	500.0	1.0	0.7	0.4	1.0
1	600.0	1.0	0.7	0.4	1.0
2	700.0	1.0	0.7	0.4	1.0
3	800.0	1.0	0.7	0.4	1.0
4	900.0	1.0	0.7	0.4	1.0
5	1000.0	1.0	0.7	0.4	1.0
6	1100.0	1.0	0.7	0.4	1.0
7	1200.0	1.0	0.7	0.4	1.0
8	1300.0	1.0	0.7	0.4	1.0
9	1400.0	1.0	0.7	0.4	1.0

In [35]:
```
best_index = models['Test Precision'].idxmax()
models.iloc[best_index, :]
```

Out[35]:
```
C                 500.0
Train Accuracy      1.0
Test Accuracy       0.7
Test Recall         0.4
Test Precision      1.0
Name: 0, dtype: float64
```

In [36]:
```
best_index = models[models['Test Precision']==1]['Test Accuracy'].idxmax()
svc = svm.SVC(C=list_C[best_index])
svc.fit(x_train, y_train)
models.iloc[best_index, :]
```

Out[36]:
```
C                 500.0
Train Accuracy      1.0
Test Accuracy       0.7
Test Recall         0.4
Test Precision      1.0
Name: 0, dtype: float64
```

In [37]:
```
#Support Vector Machine Confusion Matrix
svm_confusion = metrics.confusion_matrix(y_test, svc.predict(x_test))
pd.DataFrame(data = svm_confusion, columns = ['Predicted 0', 'Predicted 1'],
            index = ['Actual 0', 'Actual 1'])
```

Out[37]:

	Predicted 0	Predicted 1
Actual 0	5	0
Actual 1	3	2