Sentiment_Analysis

August 28, 2017

1 Amazon Fine Food Reviews Prediction Model

Data Fields Explanation

The Amazon Fine Food Reviews dataset consists of 568,454 food reviews. This dataset consists of a single CSV file, Reviews.csv. The columns in the table are:

We will built a model on this data using NAIVE BAYES. Let's do everything step by step.

1.1 STEP-1: Reading the data and removing unwanted columns

In [1]:
```
#Let's import pandas to read the csv file.
import pandas as pd
dataset = pd.read_csv("Reviews.csv")
dataset.head()
```

Out[1]:
```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
```

In [2]:
```
#Here we will sort the dataframe according to 'Time' feature
dataset.sort_values(['Time'], ascending=True, inplace=True)
dataset.head()
```

Out[2]:
```
                 Id   ProductId          UserId         ProfileName  \
150523  150524  0006641040   ACITT7DI6IDDL     shari zychinski
150500  150501  0006641040   AJ46FKXOVC7NR  Nicholas A Mesiano
451855  451856  B00004CXX9   AIUWLEQ1ADEG5    Elizabeth Medina
230284  230285  B00004RYGX  A344SMIA5JECGM     Vincent P. Ross
451877  451878  B00004CXX9  A344SMIA5JECGM     Vincent P. Ross

        HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
150523                     0                       0      5  939340800
150500                     2                       2      5  940809600
451855                     0                       0      5  944092800
230284                     1                       2      5  944438400
451877                     1                       2      5  944438400

                                                  Summary  \
150523                               EVERY book is educational
150500  This whole series is great way to spend time w...
451855                                    Entertainingl Funny!
230284                                  A modern day fairy tale
451877                                  A modern day fairy tale

                                                     Text
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...
451855  Beetlejuice is a well written movie ... ever...
230284  A twist of rumplestiskin captured on film, sta...
451877  A twist of rumplestiskin captured on film, sta...
```

In [3]:
```
#Dropping the unwanted columns from our data frame.
dataset.drop("Id", inplace=True, axis=1)
dataset.drop("ProductId", inplace=True, axis=1)
dataset.drop("ProfileName", inplace=True, axis=1)
dataset.drop("HelpfulnessNumerator", inplace=True, axis=1)
dataset.drop("HelpfulnessDenominator", inplace=True, axis=1)
dataset.drop("Time", inplace=True, axis=1)
dataset.head()
```

Out[3]:
```
                UserId  Score  \
150523   ACITT7DI6IDDL      5
150500   AJ46FKXOVC7NR      5
451855   AIUWLEQ1ADEG5      5
230284  A344SMIA5JECGM      5
451877  A344SMIA5JECGM      5

                                                  Summary  \
150523                               EVERY book is educational
150500  This whole series is great way to spend time w...
451855                                    Entertainingl Funny!
```

```
230284                              A modern day fairy tale
451877                              A modern day fairy tale

                                                       Text
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...
451855  Beetlejuice is a well written movie ... ever...
230284  A twist of rumplestiskin captured on film, sta...
451877  A twist of rumplestiskin captured on film, sta...
```

In [4]:
```
#Make all 'Score' less than 3 equal to -ve class and
# 'Score' greater than 3 equal to +ve class.
dataset.loc[dataset['Score']<3, 'Score'] = [0]
dataset.loc[dataset['Score']>3, 'Score'] = [1]
```

In [5]:
```
dataset.head()
```

Out[5]:
```
                UserId  Score  \
150523   ACITT7DI6IDDL      1
150500   AJ46FKXOVC7NR      1
451855   AIUWLEQ1ADEG5      1
230284  A344SMIA5JECGM      1
451877  A344SMIA5JECGM      1

                                                    Summary  \
150523                                EVERY book is educational
150500  This whole series is great way to spend time w...
451855                                     Entertainingl Funny!
230284                                  A modern day fairy tale
451877                                  A modern day fairy tale

                                                       Text
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...
451855  Beetlejuice is a well written movie ... ever...
230284  A twist of rumplestiskin captured on film, sta...
451877  A twist of rumplestiskin captured on film, sta...
```

In [6]:
```
#import numpy as np
#from sklearn.model_selection import train_test_split
#train, test = train_test_split(dataset, test_size = 0.3)

total_size=len(dataset)

train_size=int(0.70*total_size)

#training dataset
train=dataset.head(train_size)
```

```
#test dataset
test=dataset.tail(total_size - train_size)
```

In [7]:
```
train.Score.value_counts()
```

Out[7]:
```
1    313696
0     55168
3     29053
Name: Score, dtype: int64
```

In [8]:
```
test.Score.value_counts()
```

Out[8]:
```
1    130081
0     26869
3     13587
Name: Score, dtype: int64
```

In [9]:
```
# Removing all rows where 'Score' is equal to 3
train = train[train.Score != 3]
test = test[test.Score != 3]
```

In [10]:
```
print(train.shape)
print(test.shape)
```

```
(368864, 4)
(156950, 4)
```

In [11]:
```
train['Score'].value_counts()
```

Out[11]:
```
1    313696
0     55168
Name: Score, dtype: int64
```

In [12]:
```
test.Score.value_counts()
```

Out[12]:
```
1    130081
0     26869
Name: Score, dtype: int64
```

1.2 STEP-2: Text Preprocessing

Text preprocessing will further contain a sequence of steps: 1. Converting to lower-case. 2. Removing HTML Tags. 3. Removing Special Characters. 4. Removing Stop Words. 5. Stemming (Snowball Stemming)

In [13]:
```
#Taking the 'Text' & 'Summary' column in seperate list for further
#text preprocessing.
lst_text = train['Text'].tolist()
lst_summary = train['Summary'].tolist()
```

In [14]:
```
test_text = test['Text'].tolist()
```

1.2.1 Converting to lower-case

```
In [15]: #Converting the whole list to lower-case.
         lst_text = [str(item).lower() for item in lst_text]
         lst_summary = [str(item).lower() for item in lst_summary]

In [16]: test_text = [str(item).lower() for item in test_text]
```

1.2.2 Removing HTML Tags

```
In [17]: #Lets now remove all HTML tags from the list of strings.
         import re
         def striphtml(data):
             p = re.compile(r'<.*?>')
             return p.sub('', data)

         for i in range(len(lst_text)):
             lst_text[i] = striphtml(lst_text[i])
             lst_summary[i] = striphtml(lst_summary[i])

In [18]: for i in range(len(test_text)):
             test_text[i] = striphtml(test_text[i])

In [19]: lst_text[0:5]

Out[19]: ["this witty little book makes my son laugh at loud. i recite it in the car as we're 
          "i can remember seeing the show when it aired on television years ago, when i was a 
          'beetlejuice is a well written movie ... everything about it is excellent! from the 
          "a twist of rumplestiskin captured on film, starring michael keaton and geena davis 
          "a twist of rumplestiskin captured on film, starring michael keaton and geena davis 
```

1.2.3 Removing Special Characters

```
In [20]: #Now we will remove all special characters from the strings.
         for i in range(len(lst_text)):
             lst_text[i] = re.sub(r'[^A-Za-z]+', ' ', lst_text[i])
             lst_summary[i] = re.sub(r'[^A-Za-z]+', ' ', lst_summary[i])

In [21]: for i in range(len(test_text)):
             test_text[i] = re.sub(r'[^A-Za-z]+', ' ', test_text[i])

In [22]: lst_text[0:5]

Out[22]: ['this witty little book makes my son laugh at loud i recite it in the car as we re d
          'i can remember seeing the show when it aired on television years ago when i was a c
          'beetlejuice is a well written movie everything about it is excellent from the acting
          'a twist of rumplestiskin captured on film starring michael keaton and geena davis in
          'a twist of rumplestiskin captured on film starring michael keaton and geena davis in
```

1.2.4 Removing Stop Words

```
In [23]: #Removing Stop Words
         from nltk.corpus import stopwords
         from nltk.tokenize import word_tokenize
         #word_tokenize accepts a string as an input, not a file.
         stop_words = set(stopwords.words('english'))
         for i in range(len(lst_text)):
             text_filtered = []
             summary_filtered = []
             text_word_tokens = []
             summary_word_tokens = []
             text_word_tokens = lst_text[i].split()
             summary_word_tokens = lst_summary[i].split()
             for r in text_word_tokens:
                 if not r in stop_words:
                     text_filtered.append(r)
             lst_text[i] = ' '.join(text_filtered)
             for r in summary_word_tokens:
                 if not r in stop_words:
                     summary_filtered.append(r)
             lst_summary[i] = ' '.join(summary_filtered)
```

```
In [24]: for i in range(len(test_text)):
             text_filtered = []
             text_word_tokens = []
             text_word_tokens = test_text[i].split()
             for r in text_word_tokens:
                 if not r in stop_words:
                     text_filtered.append(r)
             test_text[i] = ' '.join(text_filtered)
```

1.2.5 Stemming

The three major stemming algorithms in use today are Porter, Snowball(Porter2), and Lancaster (Paice-Husk), with the aggressiveness continuum basically following along those same lines.

Porter: Most commonly used stemmer without a doubt, also one of the most gentle stemmers. One of the few stemmers that actually has Java support which is a plus, though it is also the most computationally intensive of the algorithms(Granted not by a very significant margin). It is also the oldest stemming algorithm by a large margin.

Snowball: Nearly universally regarded as an improvement over porter, and for good reason. Porter himself in fact admits that it is better than his original algorithm. Slightly faster computation time than porter, with a fairly large community around it.

Lancaster: Very aggressive stemming algorithm, sometimes to a fault. With porter and snowball, the stemmed representations are usually fairly intuitive to a reader, not so with Lancaster, as many shorter words will become totally obfuscated. The fastest algorithm here, and will reduce your working set of words hugely, but if you want more distinction, not the tool you would want.

1.2.6 Snowball Stemming

```
In [25]: #Lets now stem each word.
         from nltk.stem.snowball import SnowballStemmer
         stemmer = SnowballStemmer("english")
         for i in range(len(lst_text)):
             text_filtered = []
             summary_filtered = []
             text_word_tokens = []
             summary_word_tokens = []
             text_word_tokens = lst_text[i].split()
             summary_word_tokens = lst_summary[i].split()
             for r in text_word_tokens:
                 text_filtered.append(str(stemmer.stem(r)))
             lst_text[i] = ' '.join(text_filtered)
             for r in summary_word_tokens:
                 summary_filtered.append(str(stemmer.stem(r)))
             lst_summary[i] = ' '.join(summary_filtered)
```

```
In [26]: for i in range(len(test_text)):
             text_filtered = []
             text_word_tokens = []
             text_word_tokens = test_text[i].split()
             for r in text_word_tokens:
                 if not r in stop_words:
                     text_filtered.append(str(stemmer.stem(r)))
             test_text[i] = ' '.join(text_filtered)
```

```
In [27]: lst_text[0:5]
```

```
Out[27]: ['witti littl book make son laugh loud recit car drive along alway sing refrain learn
          'rememb see show air televis year ago child sister later bought lp day thirti someth
          'beetlejuic well written movi everyth excel act special effect delight chose view mo
          'twist rumplestiskin captur film star michael keaton geena davi prime tim burton mas
          'twist rumplestiskin captur film star michael keaton geena davi prime tim burton mas
```

```
In [28]: test_text[0:5]
```

```
Out[28]: ['love make smoothi chocol protein powder peanut butter banana pb drove calori way ca
          'smooth impli weak robust impli bitter neither true negat true wonder complex simpl
          'alway hate instant oatmeal like old fashion thick cut roll oat like steel cut whene
          'tri creamer like caramel flavor coffe like creamer whether like matter person prefe
          'recommend pretti savvi dog owner third larg bag purchas incred price dog healthi ri
```

1.3 STEP-3: Vectorizing our dataset

From the scikit-learn documentation:

```
Text Analysis is a major application field for machine learning algorithms. However the raw da

We call vectorization the general process of turning a collection of text documents into numer
```

We will use CountVectorizer to "convert text into a matrix of token counts".

```
In [29]: from sklearn.feature_extraction.text import CountVectorizer

         # Initialize the "CountVectorizer" object, which is scikit-learn's
         # bag of words tool.
         vect = CountVectorizer()
         # fit_transform() does two functions: First, it fits the model
         # and learns the vocabulary; second, it transforms our training data
         # into feature vectors. The input to fit_transform should be a list of
         # strings.
         X_train_dtm = vect.fit_transform(lst_text)
         # Numpy arrays are easy to work with, so convert the result to an
         # array
         #train_data_features = train_data_features.toarray()
```

```
In [30]: # examine the document-term matrix
         X_train_dtm
```

```
Out[30]: <368864x71178 sparse matrix of type '<type 'numpy.int64'>'
                 with 11579336 stored elements in Compressed Sparse Row format>
```

In order to make a prediction, the new observation must have the same features as the training observations, both in number and meaning.

```
In [31]: # transform testing data (using fitted vocabulary) into a document-term matrix
         X_test_dtm = vect.transform(test_text)
         X_test_dtm
```

```
Out[31]: <156950x71178 sparse matrix of type '<type 'numpy.int64'>'
                 with 5017739 stored elements in Compressed Sparse Row format>
```

Summary:

```
<li> <code>vect.fit(lst_text)</code> <b>learns the vocabulary</b> of the training data
<li> <code>vect.transform(lst_text)</code> <b>uses the fitted vocabulary</b> to build a <b>docu
<li> <code>vect.transform(test_text)</code> <b>uses the fitted vocabulary</b> to build a docume
```

1.4 STEP-4: Building and evaluating the model

1.4.1 Models

Here we will be implementing various models and comparing their accuracies. We would be implementing below mentioned machine learning algorithms:

Naive Bayes
Logistic Regression - with L1 and L2 regularizors
Linear SVM
RBF Kernel SVM

1.4.2 Evaluation Metrics

What is the Confusion Mattix? A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

Let's now define the most basic terms, which are whole numbers (not rates):

Various Evaluation metrics are: Accuracy: Overall, how often is the classifier correct? Accuracy works best if false positives and false negatives have similar cost. This fails in case of imbalanced dataset. For that we have other metrics, like Precision and Recall. Accuracy = TP+TN/TP+FP+FN+TN True Positive Rate: When it's actually yes, how often does it predict yes? Also known as "Sensitivity" or "Recall" False Positive Rate: When it's actually no, how often does it predict yes? F1 Score: This is a weighted average of the true positive rate (recall) and precision. ROC Curve: This is a commonly used graph that summarizes the performance of a classifier over all possible thresholds. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class. Log-loss: It is usefull in comparing two models. For a perfect Classifier log-loss=0. For a incorrect classifier log-loss=infinity.

Precision (P) is defined as the number of true positives (TP) over the number of true positives plus the number of false positives (FP). Precision = TP/TP+FP

Recall (R) is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FP). Recall = TP/TP+FN

These quantities are also related to the F1 score, which is defined as the harmonic mean of precision and recall. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. F1 Score = 2*(Recall* Precision) / (Recall + Precision)

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. Note the below mentioned points:

High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels.

A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels.

An ideal system with high precision and high recall will return many results, with all results labeled correctly.

1.4.3 Multinomial Naive Bayes

We will use Multinomial Naive Bayes:

```
The multinomial Naive Bayes classifier is suitable for classification with discrete features (
```

```
In [32]: # import and instantiate a Multinomial Naive Bayes model
         from sklearn.naive_bayes import MultinomialNB
         nb = MultinomialNB()
```

In [33]:
```python
# train the model using X_train_dtm (timing it with an IPython
#"magic command")
%time nb.fit(X_train_dtm, train.Score)
```

```
Wall time: 157 ms
```

Out[33]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [34]:
```python
# make class predictions for X_test_dtm
y_pred_class_nb = nb.predict(X_test_dtm)
```

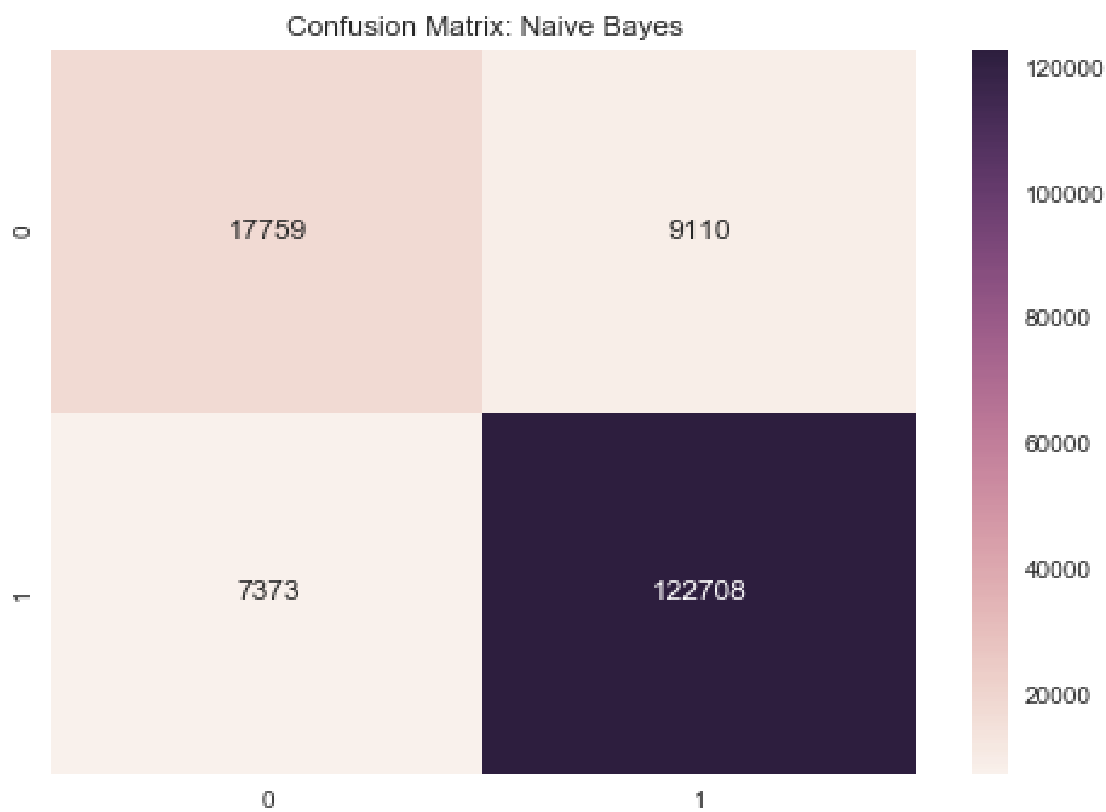
In [35]:
```python
# calculate accuracy of class predictions
from sklearn import metrics
metrics.accuracy_score(test.Score, y_pred_class_nb)
```

Out[35]:
```
0.89497929276839761
```

In [36]:
```python
# print the confusion matrix
con_metrics_nb = metrics.confusion_matrix(test.Score, y_pred_class_nb)
con_metrics_nb
```

Out[36]:
```
array([[ 17759,   9110],
       [  7373, 122708]])
```

In [37]:
```python
#ploting heatmap for confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(con_metrics_nb, annot=True, fmt='d')
plt.title("Confusion Matrix: Naive Bayes")
plt.show()
```



In [38]:
```
#Checking Precision, Recall and F1 Score
print metrics.classification_report(test.Score, y_pred_class_nb)
```

```
             precision    recall  f1-score   support

          0       0.71      0.66      0.68     26869
          1       0.93      0.94      0.94    130081

avg / total       0.89      0.89      0.89    156950
```

In [39]:
```
# calculate predicted probabilities for X_test_dtm (poorly calibrated)
#y_pred_prob = nb.predict_proba(X_test_dtm)[:, 1]
#y_pred_prob
```

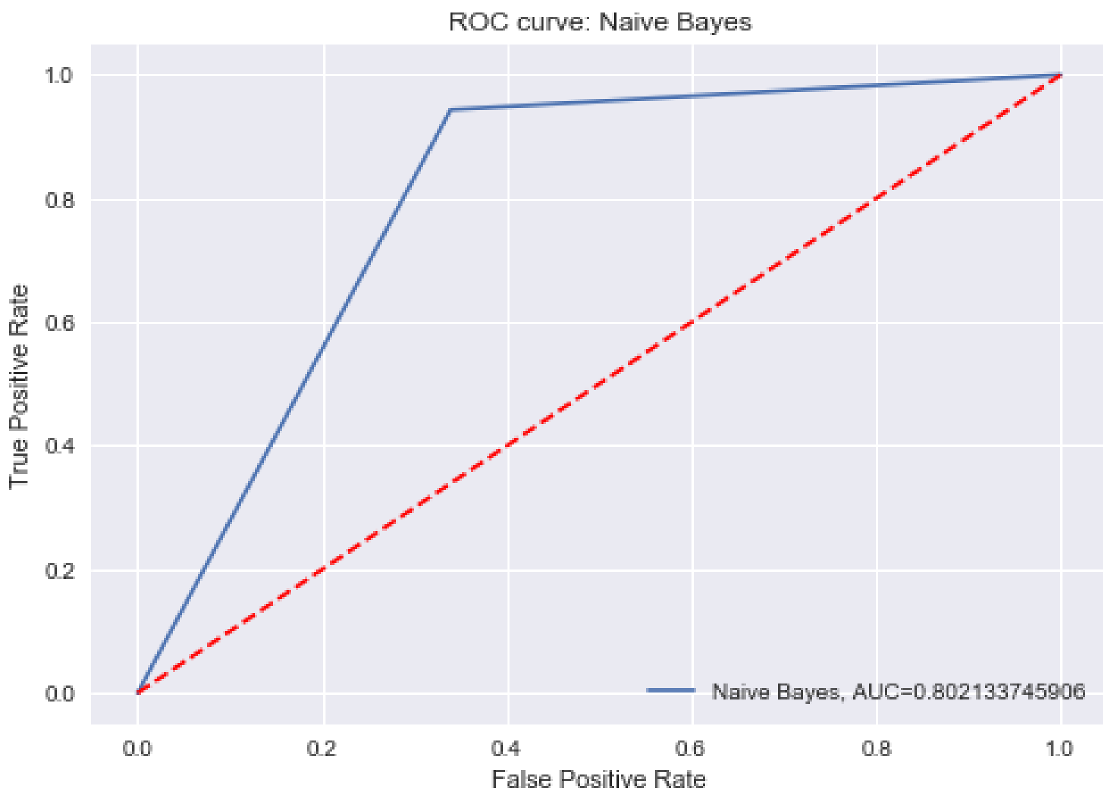
In [40]:
```
# calculate AUC - Method 1
false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(test.Score, y_
print metrics.auc(false_positive_rate, true_positive_rate)
```

```
0.802133745906
```

In [41]:
```
# calculate AUC _ Method - 2
auc_nb = metrics.roc_auc_score(test.Score, y_pred_class_nb)
print(auc_nb)
```

```
0.802133745906
```

In [42]:
```
#Plotting Area Under the Curve
plt.plot(false_positive_rate,true_positive_rate,label="Naive Bayes, AUC="+str(auc_nb))
plt.plot([0,1],[0,1],'r--')
plt.title('ROC curve: Naive Bayes')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

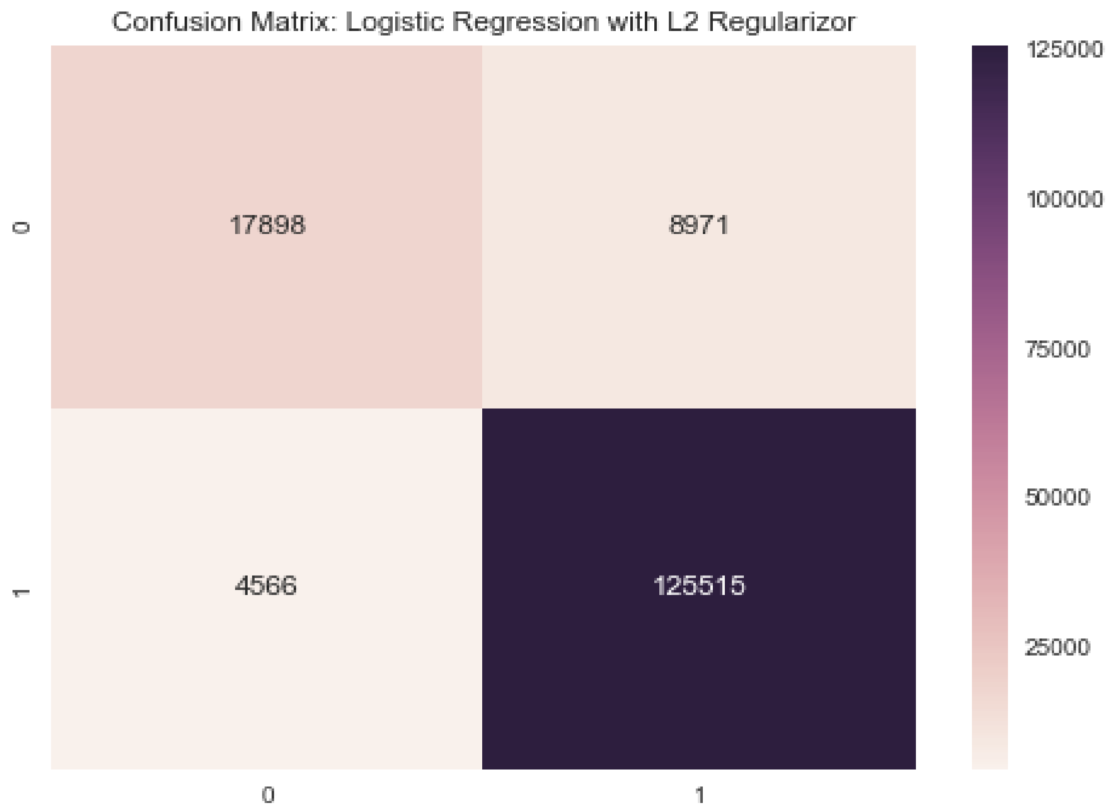


In [43]:
```
#Calculating Log-Loss
metrics.log_loss(test.Score, y_pred_class_nb)
```

Out[43]: 3.6273331357113858

1.4.4 Logistic Regression with L2 Regularizor

```
In [44]: # import and instantiate a logistic regression model
         from sklearn.linear_model import LogisticRegression
         logreg_l2 = LogisticRegression()

In [45]: # train the model using X_train_dtm
         %time logreg_l2.fit(X_train_dtm, train.Score)

Wall time: 2min 8s


Out[45]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [46]: # make class predictions for X_test_dtm
         y_pred_class_l2 = logreg_l2.predict(X_test_dtm)

In [47]: # calculate accuracy
         metrics.accuracy_score(test.Score, y_pred_class_l2)

Out[47]: 0.91374960178400766

In [48]: # print the confusion matrix
         con_metrics_l2 = metrics.confusion_matrix(test.Score, y_pred_class_l2)
         con_metrics_l2

Out[48]: array([[ 17898,   8971],
                [  4566, 125515]])

In [49]: #ploting heatmap for confusion matrix
         sns.heatmap(con_metrics_l2, annot=True, fmt='d')
         plt.title("Confusion Matrix: Logistic Regression with L2 Regularizor")
         plt.show()
```




```
In [50]: #Checking Precision, Recall and F1 Score
         print metrics.classification_report(test.Score, y_pred_class_l2)
```

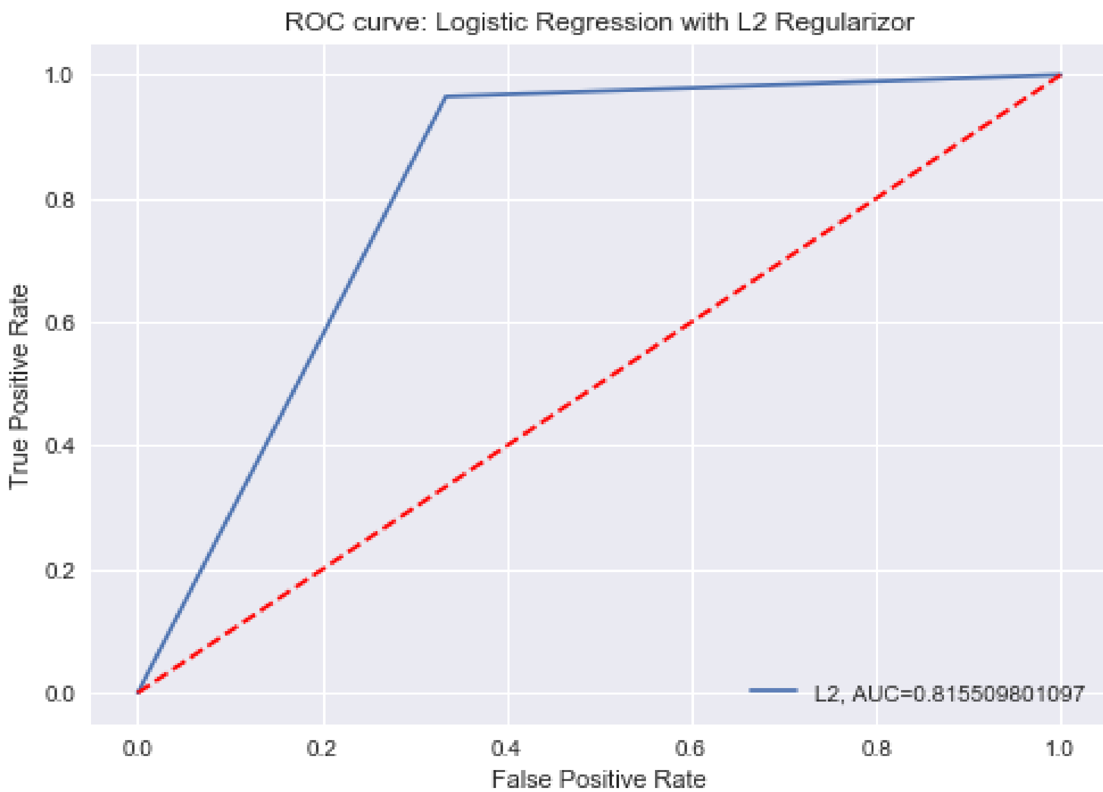
```
             precision    recall  f1-score   support

          0       0.80      0.67      0.73     26869
          1       0.93      0.96      0.95    130081

avg / total       0.91      0.91      0.91    156950
```

```
In [51]: # calculate AUC
         auc_l2 = metrics.roc_auc_score(test.Score, y_pred_class_l2)
         print(auc_l2)
```

```
0.815509801097
```

```
In [52]: #Plotting Area Under the Curve
         false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(test.Score, y_
         plt.plot(false_positive_rate,true_positive_rate,label="L2, AUC="+str(auc_l2))
```

```
plt.plot([0,1],[0,1],'r--')
plt.title('ROC curve: Logistic Regression with L2 Regularizor')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [53]:
```
#Calculating Log-Loss
metrics.log_loss(test.Score, y_pred_class_l2)
```

Out[53]: 2.9790289216084886

1.4.5 Logistic Regression with L1 Regularizor

In [54]:
```
# import and instantiate a logistic regression model
from sklearn.linear_model import LogisticRegression
logreg_l1 = LogisticRegression(penalty='l1')
```

In [55]:
```
# train the model using X_train_dtm
%time logreg_l1.fit(X_train_dtm, train.Score)
```

```
Wall time: 15.3 s
```

```
Out[55]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [56]: # make class predictions for X_test_dtm
         y_pred_class_l1 = logreg_l1.predict(X_test_dtm)
```

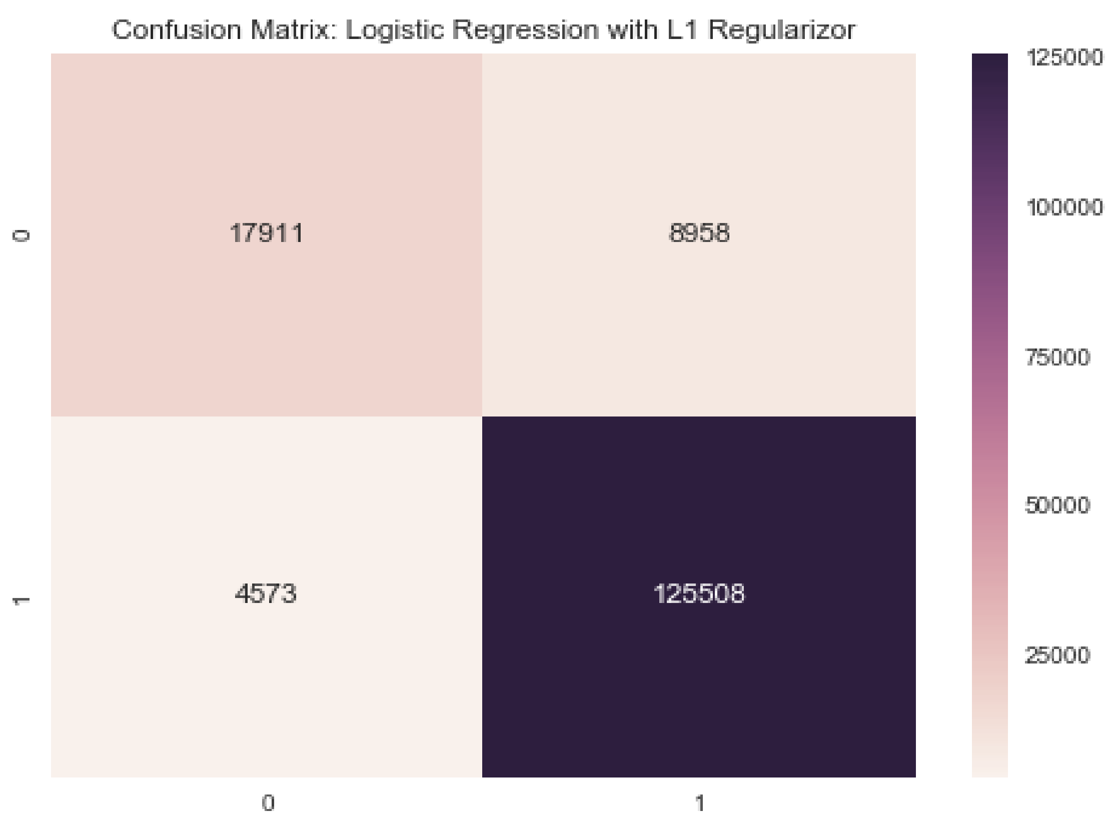
```
In [57]: # calculate accuracy
         metrics.accuracy_score(test.Score, y_pred_class_l1)
```

```
Out[57]: 0.9137878305192737
```

```
In [58]: # print the confusion matrix
         con_metrics_l1 = metrics.confusion_matrix(test.Score, y_pred_class_l1)
         con_metrics_l1
```

```
Out[58]: array([[ 17911,   8958],
                [  4573, 125508]])
```

```
In [59]: #ploting heatmap for confusion matrix
         sns.heatmap(con_metrics_l1, annot=True, fmt='d')
         plt.title("Confusion Matrix: Logistic Regression with L1 Regularizor")
         plt.show()
```



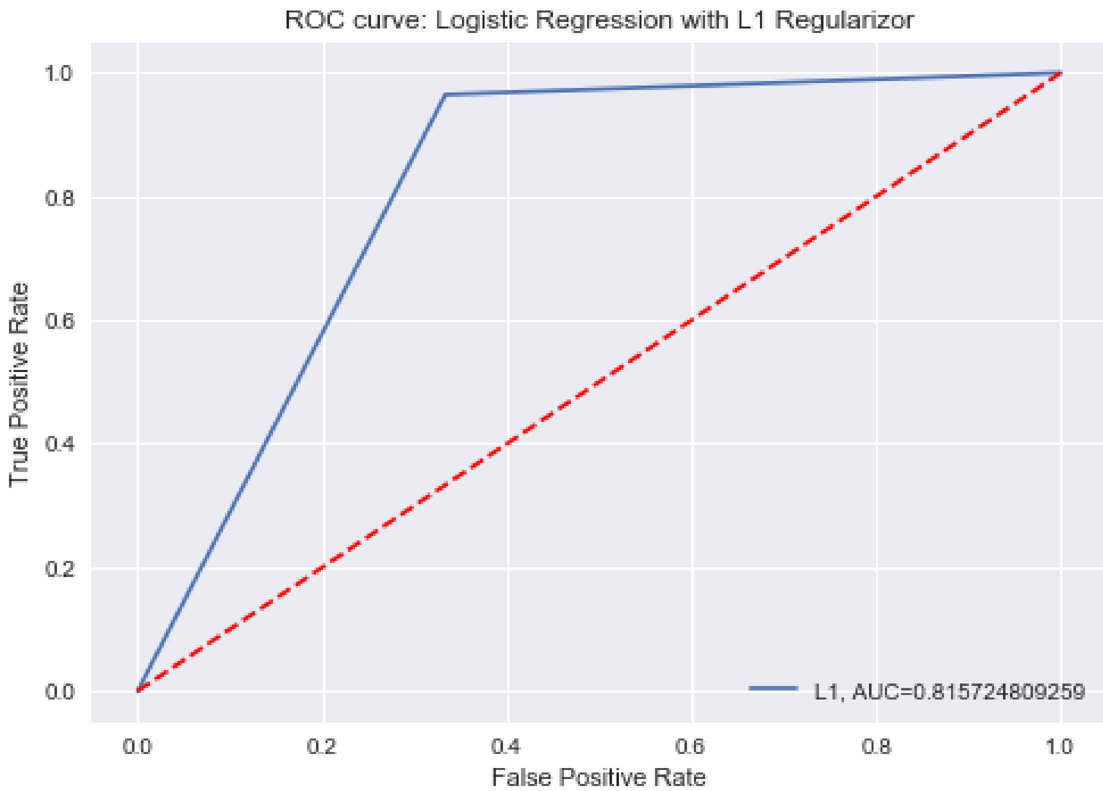
In [60]:
```
#Checking Precision, Recall and F1 Score
print metrics.classification_report(test.Score, y_pred_class_l1)
```

```
             precision    recall  f1-score   support

          0       0.80      0.67      0.73     26869
          1       0.93      0.96      0.95    130081

avg / total       0.91      0.91      0.91    156950
```

In [61]:
```
# calculate AUC
auc_l1 = metrics.roc_auc_score(test.Score, y_pred_class_l1)
print(auc_l1)
```

```
0.815724809259
```

In [72]:
```
#Plotting Area Under the Curve
false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(test.Score, y_
plt.plot(false_positive_rate,true_positive_rate,label="L1, AUC="+str(auc_l1))
plt.plot([0,1],[0,1],'r--')
plt.title('ROC curve: Logistic Regression with L1 Regularizor')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
In [73]: #Calculating Log-Loss
         metrics.log_loss(test.Score, y_pred_class_l1)
```

```
Out[73]: 2.9777084816394788
```

1.4.6 Support Vector Machines: Linear SVC

```
In [74]: # import and instantiate a Linear SVM model
         from sklearn import svm
         lrsvc = svm.LinearSVC()
         # train the model using X_train_dtm
         %time lrsvc.fit(X_train_dtm, train.Score)
```

```
Wall time: 1min 14s
```

```
Out[74]: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)
```

```
In [75]: # make class predictions for X_test_dtm
         y_pred_class_lrsvc = lrsvc.predict(X_test_dtm)
```

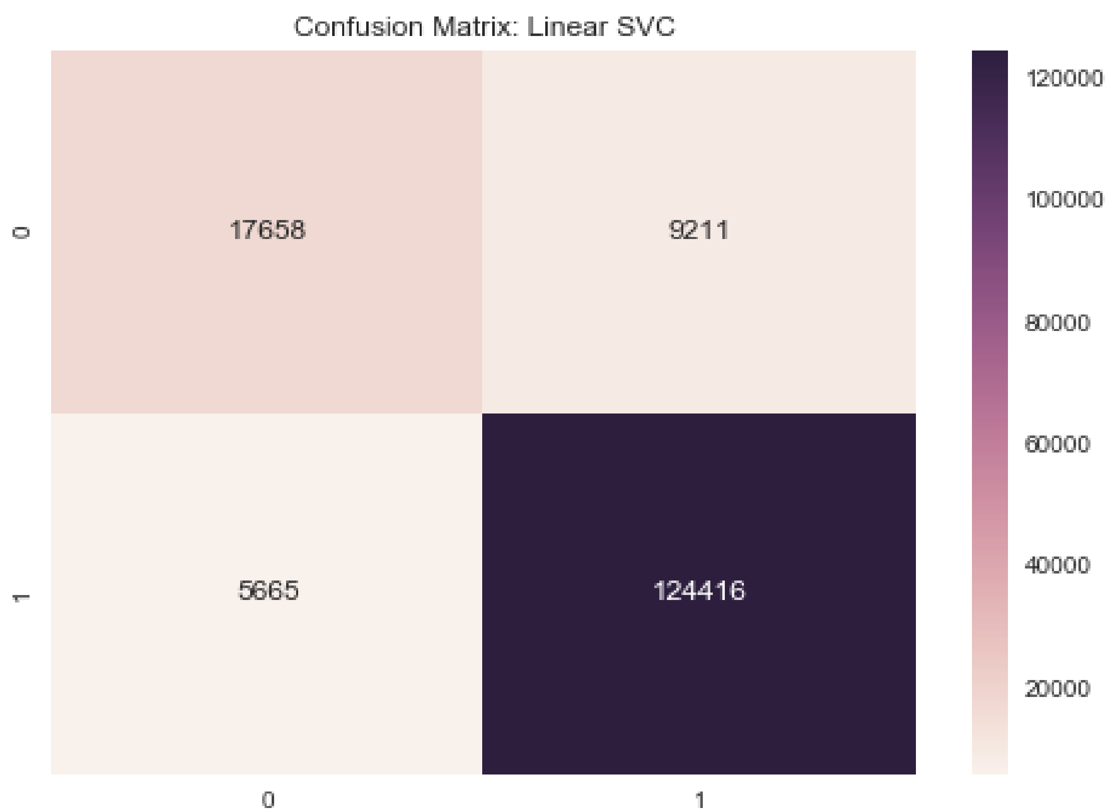
In [76]:
```
# calculate accuracy
metrics.accuracy_score(test.Score, y_pred_class_lrsvc)
```

Out[76]: 0.90521822236381011

In [77]:
```
# print the confusion matrix
con_metrics_lrsvc = metrics.confusion_matrix(test.Score, y_pred_class_lrsvc)
con_metrics_lrsvc
```

Out[77]:
```
array([[ 17658,   9211],
       [  5665, 124416]])
```

In [78]:
```
#ploting heatmap for confusion matrix
sns.heatmap(con_metrics_lrsvc, annot=True, fmt='d')
plt.title("Confusion Matrix: Linear SVC")
plt.show()
```



In [79]:
```
#Checking Precision, Recall and F1 Score
print metrics.classification_report(test.Score, y_pred_class_lrsvc)
```

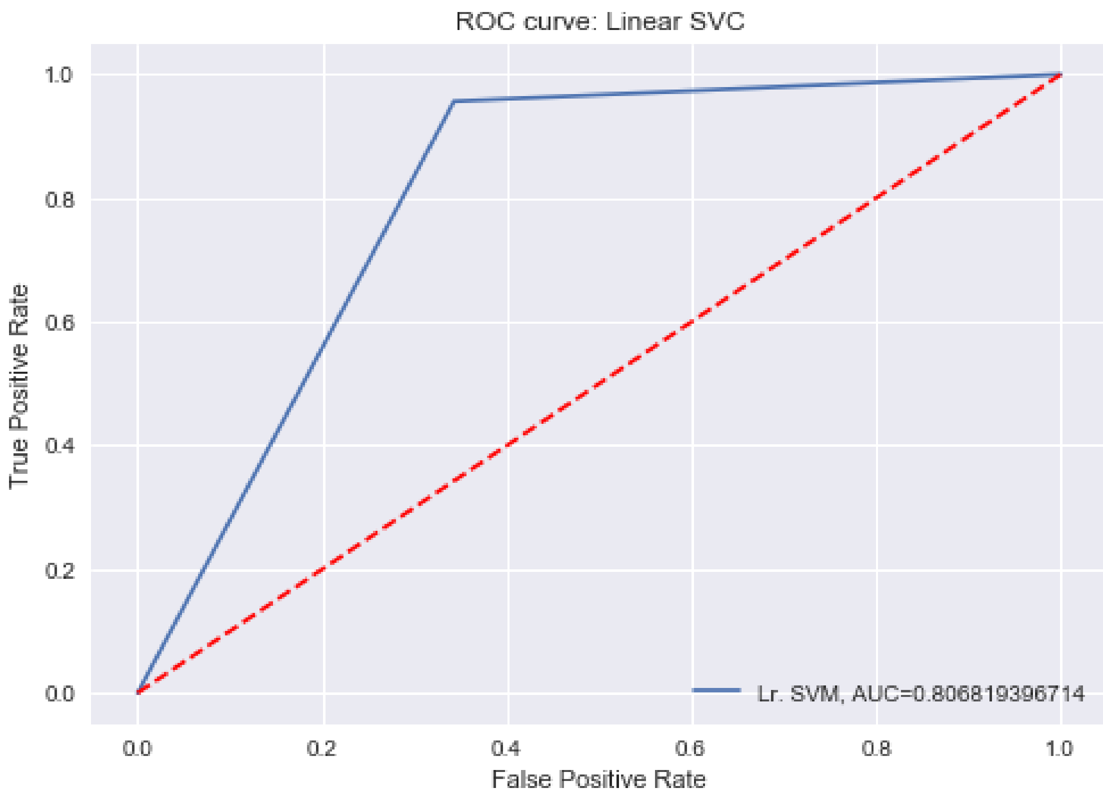
```
    precision    recall  f1-score   support
```

```
             0       0.76      0.66      0.70     26869
             1       0.93      0.96      0.94    130081

avg / total       0.90      0.91      0.90    156950
```

```
In [80]: # calculate AUC
         auc_lrsvc = metrics.roc_auc_score(test.Score, y_pred_class_lrsvc)
         print(auc_lrsvc)
```

```
0.806819396714
```

```
In [81]: #Plotting Area Under the Curve
         false_positive_rate, true_positive_rate, thresholds = metrics.roc_curve(test.Score, y_
         plt.plot(false_positive_rate,true_positive_rate,label="Lr. SVM, AUC="+str(auc_lrsvc))
         plt.plot([0,1],[0,1],'r--')
         plt.title('ROC curve: Linear SVC')
         plt.legend(loc='lower right')
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```



In [82]:
```python
#Calculating Log-Loss
metrics.log_loss(test.Score, y_pred_class_lrsvc)
```

Out[82]:
```
3.2736935504467723
```

1.4.7 Support Vector Machines: RBF Kernel SVC

In [83]:
```python
from sklearn.kernel_approximation import RBFSampler
from sklearn.linear_model import SGDClassifier

rbf_feature = RBFSampler(gamma=1, random_state=1)
X_features = rbf_feature.fit_transform(X_train_dtm)
clf = SGDClassifier()
clf.fit(X_features, train.Score)
```

Out[83]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=None, shuffle=True,
       verbose=0, warm_start=False)
```

In [84]:
```python
test_feature = rbf_feature.transform(X_test_dtm)
```

In [85]:
```python
y_pred_class_rbfsvc = clf.predict(test_feature)
```

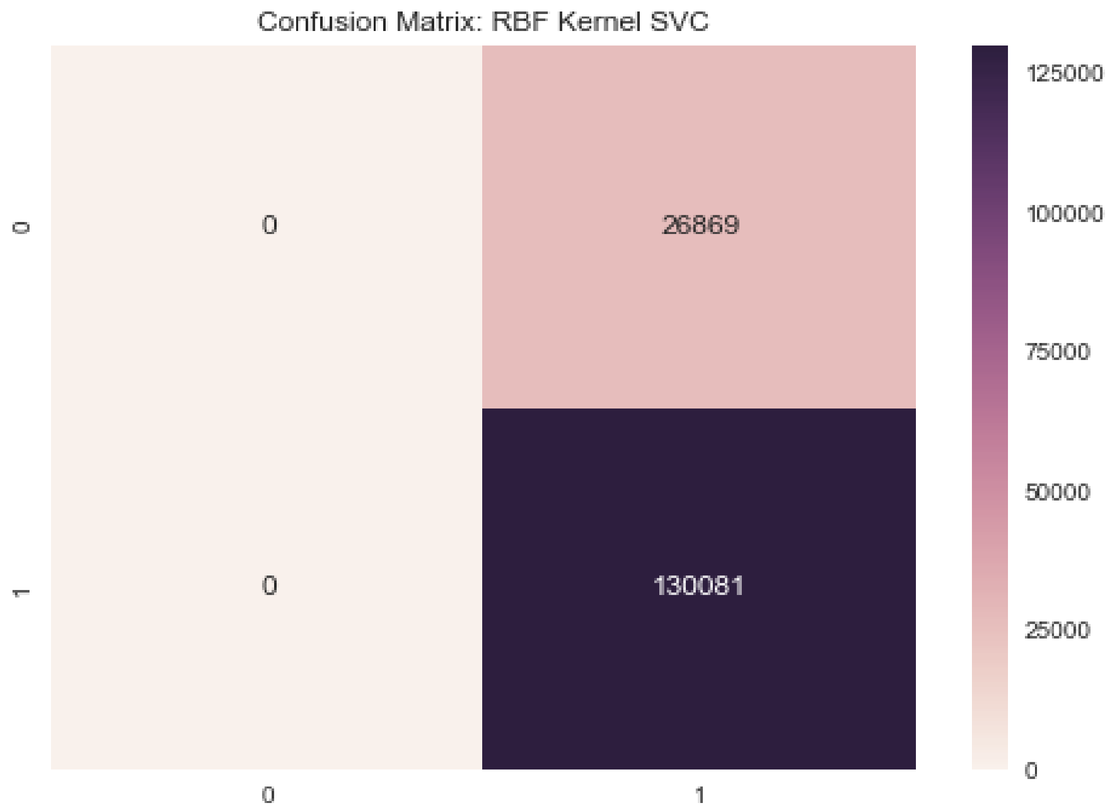
In [86]:
```python
# calculate accuracy
metrics.accuracy_score(test.Score, y_pred_class_rbfsvc)
```

Out[86]:
```
0.82880535202293726
```

In [87]:
```python
# print the confusion matrix
con_metrics_rbfsvc = metrics.confusion_matrix(test.Score, y_pred_class_rbfsvc)
con_metrics_rbfsvc
```

Out[87]:
```
array([[     0,  26869],
       [     0, 130081]])
```

In [88]:
```python
#ploting heatmap for confusion matrix
sns.heatmap(con_metrics_rbfsvc, annot=True, fmt='d')
plt.title("Confusion Matrix: RBF Kernel SVC")
plt.show()
```



In [90]:
```
#Checking Precision, Recall and F1 Score
print metrics.classification_report(test.Score, y_pred_class_rbfsvc)
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00     26869
          1       0.83      1.00      0.91    130081

avg / total       0.69      0.83      0.75    156950
```