INTRODUCTION

In [2]:

```
"""
This is my first attempt at deploying machine learning techniques to try and solve a re
al world problem.
To do so, i worked on Amazon's recommendation dataset,
The purpose of this analysis is to make up a prediction model where we will be able to
 predict whether a recommendation is positive or negative.
In this analysis, we will not focus on the Score, but only the positive/negative sentim
ent of the recommendation.
the focus of project is just to benchmark off the shelf available ML algorithms?—?naïve
 Bayesian,Logistic Regression,KNN, Decision Trees and Boosted Decision Trees.
"""
```

Out[2]:

```
"\nThis is my first attempt at deploying machine learning techniques to tr
y and solve a real world problem.\nTo do so, i worked on Amazon's recommen
dation dataset,\nThe purpose of this analysis is to make up a prediction m
odel where we will be able to predict whether a recommendation is positive
or negative.\nIn this analysis, we will not focus on the Score, but only t
he positive/negative sentiment of the recommendation.\nthe focus of projec
t is just to benchmark off the shelf available ML algorithms?—?naïve Bayes
ian,Logistic Regression,KNN, Decision Trees and Boosted Decision Trees.\n"
```

OBJECTIVE

In [3]:

```
"""
The purpose of this analysis is to make up a prediction model
where we will be able to predict whether a recommendation is positive or negative.
"""
```

Out[3]:

```
'\nThe purpose of this analysis is to make up a prediction model\nwhere we
will be able to predict whether a recommendation is positive or negativ
e.\n'
```

In [4]:

```
"""
As we only want to get the global sentiment of the recommendations (positive or negativ
e), we will purposefully ignore all Scores equal to 3.
If the score id above 3, then the recommendation wil be set to "postive". Otherwise, it
 will be set to "negative".
"""
```

Out[4]:

```
'\nAs we only want to get the global sentiment of the recommendations (pos
itive or negative), we will purposefully ignore all Scores equal to 3.\nIf
the score id above 3, then the recommendation wil be set to "postive". Oth
erwise, it will be set to "negative".\n'
```

In [1]:

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import numpy as np
import scipy.sparse as sparse

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cross_validation import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from nltk.corpus import stopwords
from sklearn.cross_validation import StratifiedKFold
from sklearn.naive_bayes import BernoulliNB
from sklearn import linear_model
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import recall_score
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and funct
ions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
import os

import seaborn as sns
import matplotlib.pyplot as plt

from pandas import Series
import time
import datetime







#Load Reviews.csv into a pandas dataFrame.
reviews = pd.read_csv("Reviews.csv")
```

In [3]:

```
# (Q) how many data-points and featrues are there?
print (reviews.shape)
```

```
(568454, 10)
```

In [8]:

```
reviews.sample()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
345046	345047	B0076MLL12	AA1L8TO0LAHX9	Nece	0

In [9]:

```
len(reviews.ProductId.unique())
```

Out[9]:

```
74258
```

In [10]:

```
len(reviews.ProductId.unique())/len(reviews)
```

Out[10]:

```
0.13063150228514533
```

In [11]:

```
#(Q) What are the column names in our dataset?
print (reviews.columns)
```

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

Helpfulness Numerator

In [12]:

```
reviews.HelpfulnessNumerator.isnull().sum()
```

Out[12]:

```
0
```

In [13]:

```
np.sum(reviews.HelpfulnessNumerator==0)
```

Out[13]:

303826

In [14]:

```
np.sum(reviews.HelpfulnessNumerator==0)/len(reviews)
```

Out[14]:

0.53447772379119507

In [15]:

```
# At least 53% are not helpful
```

In [16]:

```
reviews.HelpfulnessNumerator.describe()
```

Out[16]:

```
count    568454.000000
mean          1.743817
std           7.636513
min           0.000000
25%           0.000000
50%           0.000000
75%           2.000000
max         866.000000
Name: HelpfulnessNumerator, dtype: float64
```

Helpfulness Denominator

In [17]:

```
reviews.HelpfulnessDenominator.isnull().sum()
```

Out[17]:

0

In [18]:

```
np.sum(reviews.HelpfulnessDenominator==0)
```

Out[18]:

270052

In [19]:

```
np.sum(reviews.HelpfulnessDenominator==0)/len(reviews)
```

Out[19]:

0.4750639453676111

In [20]:

```
# At least 47% are not helpful
```

In [21]:

```
reviews.HelpfulnessDenominator.describe()
```

Out[21]:

```
count    568454.00000
mean          2.22881
std           8.28974
min           0.00000
25%           0.00000
50%           1.00000
75%           2.00000
max         923.00000
Name: HelpfulnessDenominator, dtype: float64
```

In [22]:

```
reviews[reviews.HelpfulnessDenominator>100].HelpfulnessDenominator.describe()
```

Out[22]:

```
count    423.000000
mean     204.886525
std      131.705928
min      101.000000
25%      120.000000
50%      152.000000
75%      235.000000
max      923.000000
Name: HelpfulnessDenominator, dtype: float64
```

Helpfulness Numerator/Denominator

In [23]:

```
len(reviews[reviews.HelpfulnessDenominator<reviews.HelpfulnessNumerator])
```

Out[23]:

```
2
```

time

In [24]:

```
#(Q) How many data points for each class are present?
#(or) How many datapoints for each score are present?

reviews["Score"].value_counts()
# balanced-dataset vs imbalanced datasets
#reviews is a unbalanced dataset.
```

Out[24]:

```
5    363122
4     80655
1     52268
3     42640
2     29769
Name: Score, dtype: int64
```

PECENTAGE OF SCORE

In [25]:

```
reviews_count_prcnt = reviews.Score.value_counts()

def compute_percentage(x):
    pct = float(x/reviews_count_prcnt.sum()) * 100
    return round(pct, 2)

reviews_count_prcnt = reviews_count_prcnt.apply(compute_percentage)

reviews_count_prcnt.plot(kind="bar", colormap='jet')

print(reviews_count_prcnt)
```

```
5    63.88
4    14.19
1     9.19
3     7.50
2     5.24
Name: Score, dtype: float64
```

Distribution of ratings

""" I first looked at the distribution of ratings among all of the reviews. We see that 5-star reviews constitute a large proportion (63%) of all reviews. The next most prevalent rating is 4-stars(14%), followed by 1-star (9%), 3-star (8%), and finally 2-star reviews (6%). """

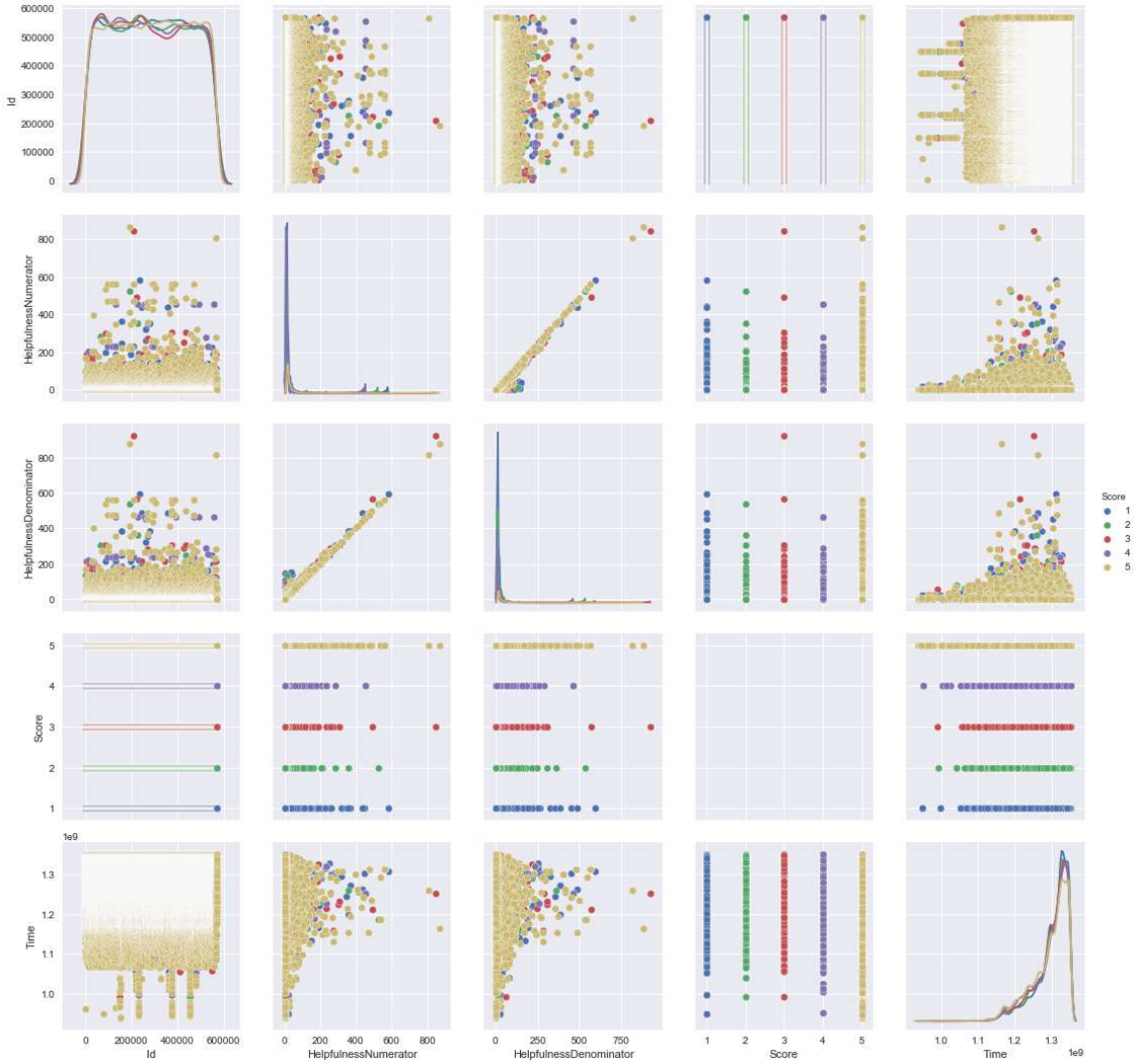
PAIR-PLOTS

In [26]:

```
# pairwise scatter plot: Pair-Plot
# Dis-advantages:
##Can be used when number of features are high.
##Cannot visualize higher dimensional patterns in 3-D and 4-D. Only possible to view 2D
 patterns.
plt.close();
sns.pairplot(reviews, hue="Score", size=3, diag_kind="kde");
plt.show()


# NOTE: the diagnol elements are PDFs for each feature. PDFs are expalined below.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.p
y:494: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X,a,b,gridsize)/(delta*nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdeto
ols.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

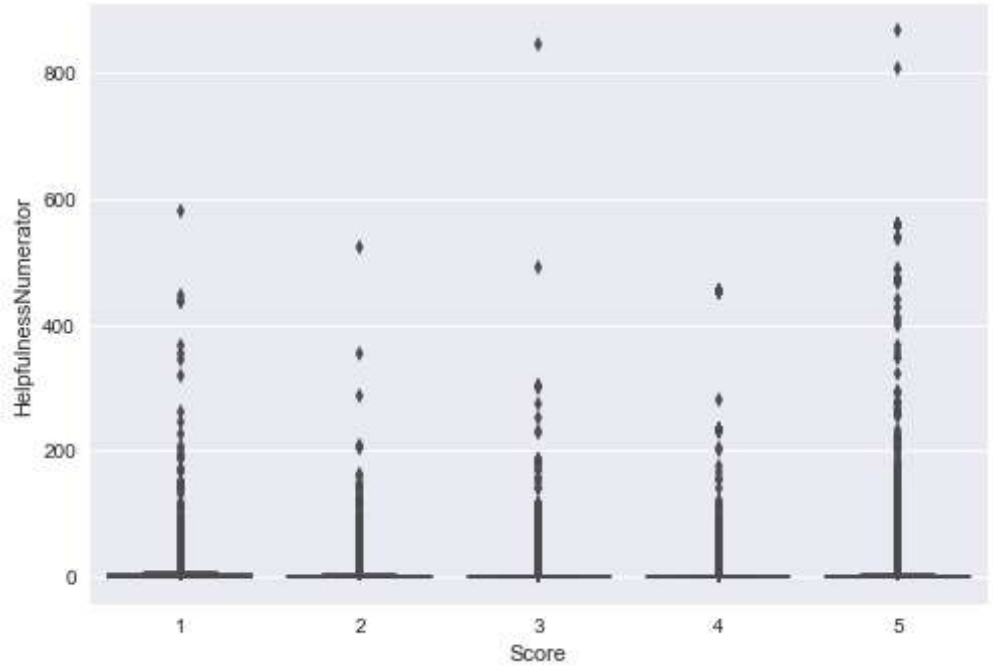


BOX-PLOTS

In [27]:

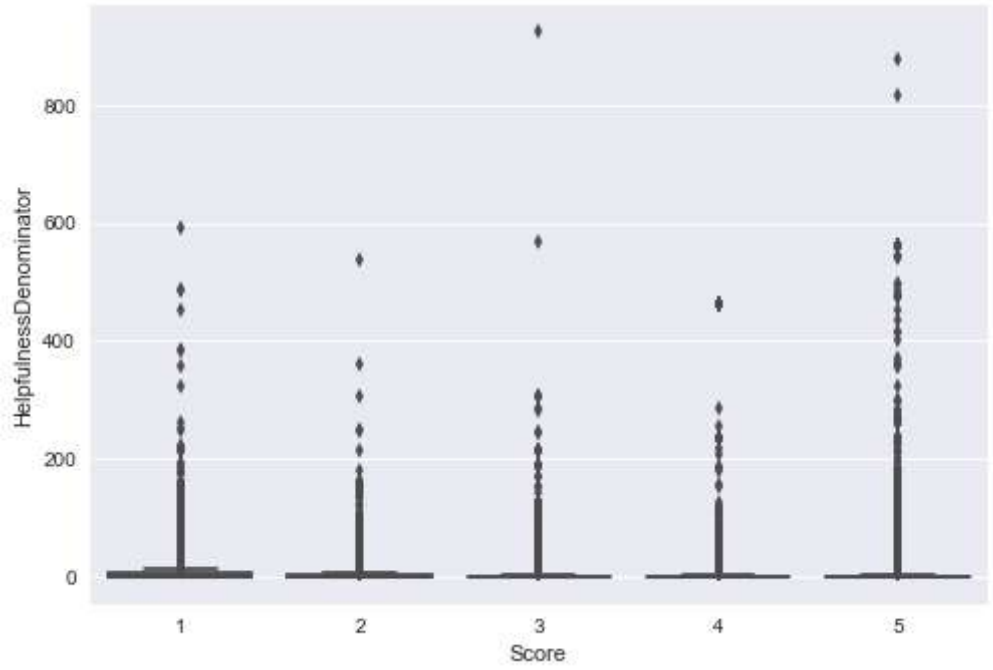
```
#Box-plot can be visualized as a PDF on the side-ways.

sns.boxplot(x='Score',y='HelpfulnessNumerator', data=reviews)
plt.show()
```



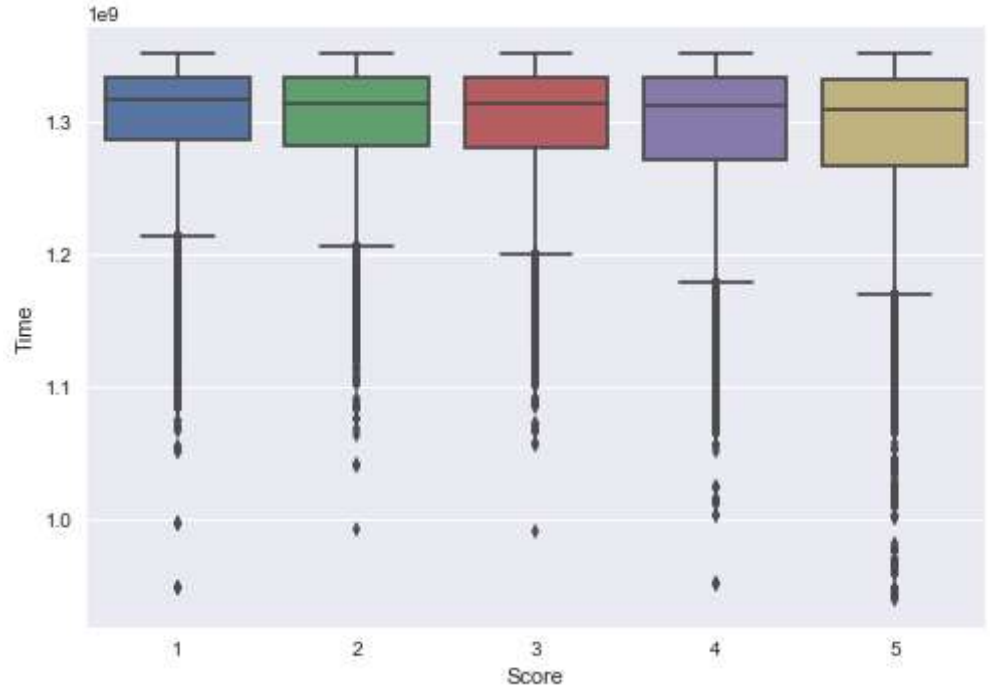
In [28]:

```
sns.boxplot(x='Score',y='HelpfulnessDenominator', data=reviews)
plt.show()
```



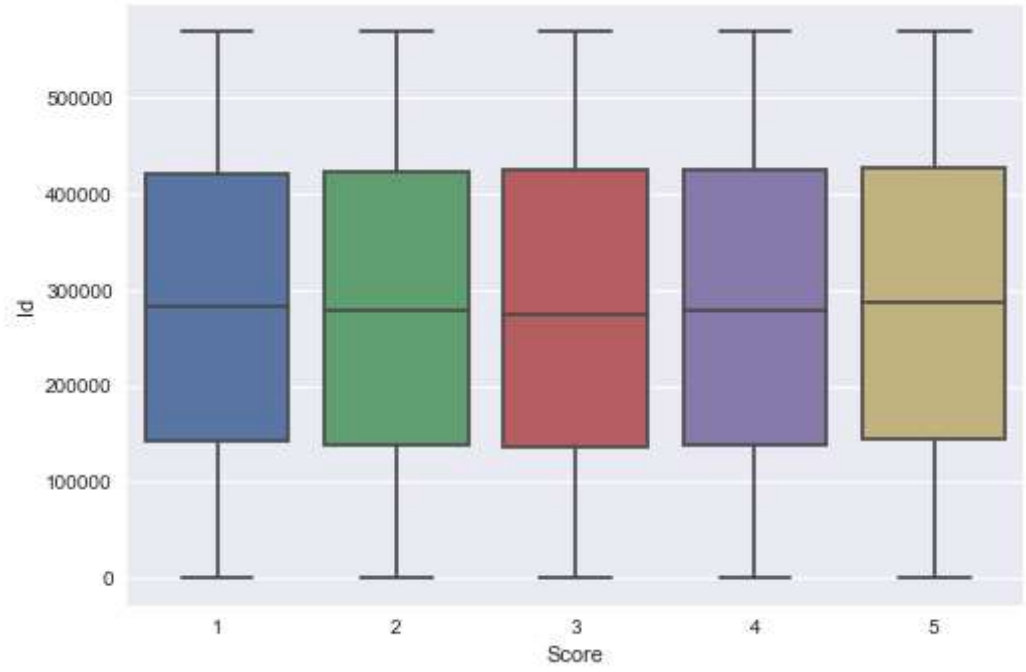
In [29]:

```
sns.boxplot(x='Score',y='Time', data=reviews)
plt.show()
```



In [30]:

```
sns.boxplot(x='Score',y='Id', data=reviews)
plt.show()
```



In [31]:

```
# frequency counts for users|product etc.

def top_n_counts (n, col, col_1):
    gb = reviews.groupby(col)[col_1].count()
    gb = gb.sort_values(ascending=False)
    return gb.head(n)

top_n_counts(15, ['ProductId','UserId'], 'ProductId')
```

Out[31]:

```
ProductId   UserId
B00008CQVA  A29JUMRL1US6YP    11
B000084EZ4  A29JUMRL1US6YP    11
B000WFORH0  A29JUMRL1US6YP    11
B000WFRQQ4  A29JUMRL1US6YP    11
B000WFKWDI  A29JUMRL1US6YP    11
B000WFEN74  A29JUMRL1US6YP    11
B000WFPJIG  A29JUMRL1US6YP    11
B000WFKI82  A29JUMRL1US6YP    11
B000WFU8O6  A29JUMRL1US6YP    11
B000WFN0VO  A29JUMRL1US6YP    11
B000WFUL3E  A29JUMRL1US6YP    11
B000084DWM  A3TVZM3ZIXG8YW    10
B003MWBFXY  A3TVZM3ZIXG8YW    10
B003MA8P02  A3TVZM3ZIXG8YW    10
B003WK0D8O  A3TVZM3ZIXG8YW    10
Name: ProductId, dtype: int64
```

In [32]:

```
top_n_counts(15, ['UserId'], 'UserId').plot(kind='bar', figsize=(20,10), colormap='wint
er')

print (top_n_counts(15, ['UserId'], 'UserId'))
```

```
UserId
A3OXHLG6DIBRW8    448
A1YUL9PCJR3JTY    421
AY12DBB0U420B     389
A281NPSIMI1C2R    365
A1Z54EM24Y40LL    256
A1TMAVN4CEM8U8    204
A2MUGFV2TDQ47K    201
A3TVZM3ZIXG8YW    199
A3PJZ8TU8FDQ1K    178
AQQLWCMRNDFGI     176
A2SZLNSI5KOQJT    175
A29JUMRL1US6YP    172
AZV26LP92E6WU     167
AY1EF0GOH80EK     162
A31N6KB160O508    162
Name: UserId, dtype: int64
```

In [33]:

```
#Need to understand what this is .. why there are multiple rows for same user, product
 and comment?
```

In [34]:

```
reviews[(reviews['ProductId'] == 'B001BDDTB2') & (reviews['UserId'] == 'AF3BYMPWKWO8F'
)][['Text','Score']]
```

Out[34]:

	Text	Score
20268	According to the manufacturer's website, this ...	1
20343	While several reviewers have alluded to the he...	3
20345	According to the manufacturer's website, this ...	1
20347	According to the manufacturer's website, this ...	1
20350	According to the manufacturer's website, this ...	1
20376	According to the manufacturer's website, this ...	1
20411	According to the manufacturer's website, this ...	1
20415	According to the manufacturer's website, this ...	1
20416	According to the manufacturer's website, this ...	1

In [35]:

```
# Time series for monthly review counts
reviews['datetime'] = pd.to_datetime(reviews["Time"], unit='s')
reviews_grp = reviews.groupby([reviews.datetime.dt.year, reviews.datetime.dt.month, rev
iews.Score]).count()['ProductId'].unstack().fillna(0)


reviews_grp.plot(figsize=(20,10), rot=45, colormap='jet')
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1611697b198>
```

In [36]:

```
reviews.shape

def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

Score = reviews['Score']
Score = Score.map(partition)

reviews['Review'] = reviews['Score'].map(partition)

print(reviews.head())
```

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

    datetime    Review
0 2011-04-27  positive
1 2012-09-07  negative
2 2008-08-18  positive
3 2011-06-13  negative
4 2012-10-21  positive
```

In [3]:

```
pos = reviews.loc[reviews['Review'] == 'positive']
pos = pos[0:25000]

neg = reviews.loc[reviews['Review'] == 'negative']
neg = neg[0:25000]
```

```
---------------------------------------------------------------------------
----
KeyError                                  Traceback (most recent call l
ast)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py
 in get_loc(self, key, method, tolerance)
   2441             try:
-> 2442                 return self._engine.get_loc(key)
   2443             except KeyError:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc (panda
s\_libs\index.c:5280)()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc (panda
s\_libs\index.c:5126)()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObj
ectHashTable.get_item (pandas\_libs\hashtable.c:20523)()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObj
ectHashTable.get_item (pandas\_libs\hashtable.c:20477)()

KeyError: 'Review'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call l
ast)
<ipython-input-3-5b048c18a662> in <module>()
----> 1 pos = reviews.loc[reviews['Review'] == 'positive']
      2 pos = pos[0:25000]
      3
      4 neg = reviews.loc[reviews['Review'] == 'negative']
      5 neg = neg[0:25000]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __ge
titem__(self, key)
   1962             return self._getitem_multilevel(key)
   1963         else:
-> 1964             return self._getitem_column(key)
   1965
   1966     def _getitem_column(self, key):

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in _get
item_column(self, key)
   1969         # get column
   1970         if self.columns.is_unique:
-> 1971             return self._get_item_cache(key)
   1972
   1973         # duplicate columns & possible reduce dimensionality

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in _g
et_item_cache(self, item)
   1643         res = cache.get(item)
   1644         if res is None:
-> 1645             values = self._data.get(item)
   1646             res = self._box_item_values(item, values)
   1647             cache[item] = res

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\internals.py in
get(self, item, fastpath)
```

```
   3588
   3589             if not isnull(item):
-> 3590                 loc = self.items.get_loc(item)
   3591             else:
   3592                 indexer = np.arange(len(self.items))[isnull(sel
f.items)]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py
 in get_loc(self, key, method, tolerance)
   2442                 return self._engine.get_loc(key)
   2443             except KeyError:
-> 2444                 return self._engine.get_loc(self._maybe_cast_in
dexer(key))
   2445
   2446         indexer = self.get_indexer([key], method=method, tolera
nce=tolerance)

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc (panda
s\_libs\index.c:5280)()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc (panda
s\_libs\index.c:5126)()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObj
ectHashTable.get_item (pandas\_libs\hashtable.c:20523)()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObj
ectHashTable.get_item (pandas\_libs\hashtable.c:20477)()

KeyError: 'Review'
```

Encoding score to Positive or negative based on value of each sample

In [38]:

```
scores = reviews['Score']
reviews['Score'] = reviews['Score'].apply(lambda x : 'pos' if x > 3 else 'neg')
```

In [39]:

```
scores.mean()
```

Out[39]:

```
4.183198640523243
```

Distribution of labels in the dataset

In [40]:

```
reviews.groupby('Score')['Summary'].count()
```

Out[40]:

```
Score
neg    124651
pos    443777
Name: Summary, dtype: int64
```

In [41]:

```
reviews.groupby('Score')['Summary'].count().plot(kind='bar',color=['r','g'],title='Labe
l Distribution',figsize=(10,6))
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1611697b198>
```

In [42]:

```
print ('Percentage of negative reviews %.2f %%' % ((reviews.groupby('Score')['Summary']
.count()['neg'])*100.0/len(reviews)))
print ('Percentage of positive reviews %.2f %%' % ((reviews.groupby('Score')['Summary']
.count()['pos'])*100.0/len(reviews)))
```

```
Percentage of negative reviews 21.93 %
Percentage of positive reviews 78.07 %
```

Splitting the dataset based on labels

In [43]:

```
def splitPosNeg(Summaries):
    neg = reviews.loc[Summaries['Score']=='neg']
    pos = reviews.loc[Summaries['Score']=='pos']
    return [pos,neg]
```

In [44]:

```
[pos,neg] = splitPosNeg(reviews)
```

Preprocessing

In [45]:

```
#Using lemmatization as it was giving better results than stemming.
#Other steps include removing punctuation amd upper case to lower case conversion.
```

In [4]:

```
df = reviews.filter(['Score','Summary','Text'], axis = 1)
df.iloc[:5]
```

Out[4]:

	Score	Summary	Text
0	5	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	4	"Delight" says it all	This is a confection that has been around a fe...
3	2	Cough Medicine	If you are looking for the secret ingredient i...
4	5	Great taffy	Great taffy at a great price. There was a wid...

Splitting the data into train and test

In [5]:

```
# finding the number of rows in our dataset
rows = df.shape[0]
rows
```

Out[5]:

```
568454
```

In [6]:

```
train_size = int(rows * 0.70)
test_size = rows - train_size
```

In [8]:

```
rows == train_size + test_size
```

Out[8]:

```
True
```

In [9]:

```
train_data = pd.DataFrame(columns=df.columns)
test_data  = pd.DataFrame(columns=df.columns)
```

In [10]:

```
print("Train Data : {}".format(train_data))
print("\nTest Data: {}".format(test_data))
print("\ntest_size : {} train_size : {}".format(test_size, train_size))
```

```
Train Data : Empty DataFrame
Columns: [Score, Summary, Text]
Index: []

Test Data: Empty DataFrame
Columns: [Score, Summary, Text]
Index: []

test_size : 170537 train_size : 397917
```

In [11]:

```
import random

# indices list contains all indices from 0 to rows..
indices =list(range(rows))
random.shuffle(indices)

# get random indices from shuffled indices array for train and test data..
trnind = indices[:train_size]
tstind = indices[train_size:rows]

print(trnind[:10])
print(tstind[:10])
```

```
[149199, 434835, 295673, 558526, 528731, 3052, 108434, 6172, 556116, 19313
5]
[211582, 209315, 421364, 135451, 459254, 185340, 179675, 236503, 264069, 5
05989]
```

In [12]:

```
# get the training data with that trnindices array
train_data = df.iloc[trnind]
train_data.shape
```

Out[12]:

```
(397917, 3)
```

In [13]:

```
# get the test data with that tstindices array
test_data = df.iloc[tstind]
test_data.shape
```

Out[13]:

```
(170537, 3)
```

In [14]:

```
train_data.iloc[:3]
```

Out[14]:

	Score	Summary	Text
149199	1	Really awful.	We bought this specifically to add body to hom...
434835	1	Are you serious?	A total rip off! The picture made it look like...
295673	5	Smokehouse 100% Natural Duck Breast Tenders	My dogs love most all of Smokehouse products. ...

In [15]:

```
"""
Removing html tags
Removing all special characters
Convert all text into small characters
Removing stop words
Stemming words using Porter Stemming algorithm
"""
```

Out[15]:

```
'\nRemoving html tags\nRemoving all special characters\nConvert all text i
nto small characters\nRemoving stop words\nStemming words using Porter Ste
mming algorithm\n'
```

In [16]:

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

ps = PorterStemmer()
```

In [17]:

```
def preProcessString(text):
    # remove all html tags
    text = re.sub('<.*?>', ' ', str(text))

    # remove all special characters
    text = re.sub('[^A-Za-z0-9]+', ' ', text)

    # converting all text into small letters and store them as words for furthur proces
sing
    text_list = text.lower().split()

    # removing stopwords from the text
    english_stop_words = set(stopwords.words('english'))
    # we have used set instead of list because, set uses hashing to store the words. So
 lookup is O(1).
    # where as for list the look up time is O(n) (ie., make things faster in list compr
ehension below)
    text_list = [word for word in text_list if word not in english_stop_words]

    # stemming the words (removing prefix and postfix) using Porter stemming algorith
m..
    text_list = [ps.stem(word) for word in text_list]

    return ' '.join(text_list)
```

In [18]:

```
train_data['Summary'] = train_data['Summary'].apply(preProcessString)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [19]:

```
train_data['Text'] = train_data['Text'].apply(preProcessString)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [20]:

```
test_data['Summary'] = test_data['Summary'].apply(preProcessString)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [21]:

```
test_data['Text'] = test_data['Text'].apply(preProcessString)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [22]:

```
# combine these two datasets so that we can save them as a csv file locally..
edited_reviews = pd.concat([train_data, test_data])
```

In [23]:

```
edited_reviews.to_csv('preprocessed_reviews_SST.csv', index=False)
```

In [7]:

```
reviews_sst = pd.read_csv('preprocessed_reviews_SST.csv')
```

In [8]:

```
rows = reviews_sst.shape[0]
rows
```

Out[8]:

```
568454
```

In [4]:

```
train_size = int(rows*0.70)
test_size = rows - train_size

print(train_size)
print(test_size)
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
<ipython-input-4-f0fc54d38ded> in <module>()
----> 1 train_size = int(rows*0.70)
      2 test_size = rows - train_size
      3
      4 print(train_size)
      5 print(test_size)

NameError: name 'rows' is not defined
```

In [27]:

```
# creating new dataframes for train aand test data
train_data = pd.DataFrame(columns=reviews_sst.columns)
test_data  = pd.DataFrame(columns=reviews_sst.columns)

print("Train Data : {}".format(train_data))
print("\nTest Data: {}".format(test_data))
print("\ntest_size : {} train_size : {}".format(test_size, train_size))
```

```
Train Data : Empty DataFrame
Columns: [Score, Summary, Text]
Index: []

Test Data: Empty DataFrame
Columns: [Score, Summary, Text]
Index: []

test_size : 170537 train_size : 397917
```

In [28]:

```
import random

# indices list contains all indices from 0 to rows..
indices =list(range(rows))
random.shuffle(indices)

# get random indices from shuffled indices array for train and test data..
trnind = indices[:train_size]
tstind = indices[train_size:rows]

print(trnind[:10])
print(tstind[:10])
```

```
[206595, 360310, 530620, 552859, 401237, 385775, 32858, 174765, 127201, 15
7194]
[183705, 355117, 463088, 360108, 196104, 220070, 369515, 32117, 18687, 155
479]
```

In [29]:

```
# split the data into train and test based on the indices that we saperated
train_data = reviews_sst.iloc[trnind]
test_data = reviews_sst.iloc[tstind]

train_data[:1]
```

Out[29]:

	Score	Summary	Text
206595	5	great oil	wonder oil hint coconut flavor nervou first re...

In [30]:

```
# Counter is subclass of dict
from collections import Counter
# we will update this counter whenever needed
counter = Counter()

counter.clear()

# get summary and text from train data
summary = train_data['Summary']
print(summary.shape)
text = train_data['Text']
print(text.shape)
```

```
(397917,)
(397917,)
```

In [31]:

```
# get word count from summaries of all train data and update to the counter
for line in summary.iteritems():
    counter.update(str(line[1]).split())

# get word count from text data of train data and update the counter
for line in text.iteritems():
    counter.update(str(line[1]).split())

len(counter)
```

Out[31]:

```
73726
```

In [32]:

```
counter.most_common(2)
```

Out[32]:

```
[('like', 213265), ('tast', 207066)]
```

In [33]:

```
# reviews_sst.shape[0]
rows
```

Out[33]:

```
568454
```

In [34]:

```
reviews_sst.shape
```

Out[34]:

```
(568454, 3)
```

In [35]:

```
reviews_sst.sample()
```

Out[35]:

	Score	Summary	Text
197803	5	emeril big easi bold k cup	love emeril big bold strong smooth would recom...

In [36]:

```
import os
import pandas as pd
import numpy as np
import scipy as sp
import seaborn as sns
import matplotlib.pyplot as plt
import json
from IPython.display import Image
from IPython.core.display import HTML
```

In [37]:

```
retval=os.chdir("..")
```

In [5]:

```
reviews_sst.head()
```

Out[5]:

	Score	Summary	Text
0	1	realli aw	bought specif add bodi homemad yogurt seen goo...
1	1	seriou	total rip pictur made look like alot differ ca...
2	5	smokehous 100 natur duck breast tender	dog love smokehous product duck breast tender ...
3	5	outstand product	tri share famili han thumb cut good lite oliv ...
4	2	impress	buy flavor would call cola realli smell like c...

In [10]:

```
kept_cols=['Score', 'Summary', 'Text']
```

In [6]:

```
reviews['smry_txt'] = reviews['Summary'].astype(str) + ' ' + reviews['Text']
dataset = reviews.loc[reviews['Score'] != 3][['smry_txt','Score']]

dataset.loc[dataset['Score']==1, 'Score'] = 'negative'
dataset.loc[dataset['Score']==2, 'Score'] = 'negative'
dataset.loc[dataset['Score']==4, 'Score'] = 'positive'
dataset.loc[dataset['Score']==5, 'Score'] = 'positive'

dataset.head()
```

Out[6]:

	smry_txt	Score
0	Good Quality Dog Food I have bought several of...	positive
1	Not as Advertised Product arrived labeled as J...	negative
2	"Delight" says it all This is a confection tha...	positive
3	Cough Medicine If you are looking for the secr...	negative
4	Great taffy Great taffy at a great price. The...	positive

Logistic Regression

In [7]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
```

GridSearchCV with l2 regularizer

In [8]:

```
count_vec_total = CountVectorizer()

x_data = count_vec_total.fit_transform(dataset.smry_txt)
y_data = dataset['Score']

y_data.head()

print(dataset.iloc[0]['smry_txt'], end='\n\n')
print(count_vec_total.inverse_transform(x_data[0]), end='\n\n')
print(x_data[0])
```

```
Good Quality Dog Food I have bought several of the Vitality canned dog foo
d products and have found them all to be of good quality. The product look
s more like a stew than a processed meat and it smells better. My Labrador
is finicky and she appreciates this product better than  most.

[array(['most', 'this', 'appreciates', 'she', 'finicky', 'is', 'labrador',
       'my', 'better', 'smells', 'it', 'meat', 'processed', 'than', 'ste
w',
       'like', 'more', 'looks', 'product', 'be', 'to', 'all', 'them',
       'found', 'and', 'products', 'canned', 'vitality', 'the', 'of',
       'several', 'bought', 'have', 'food', 'dog', 'quality', 'good'],
      dtype='<U124')]

  (0, 74176)    1
  (0, 108779)   1
  (0, 9541)     1
  (0, 97878)    1
  (0, 47541)    1
  (0, 61725)    1
  (0, 65358)    1
  (0, 75122)    1
  (0, 20970)    2
  (0, 100122)   1
  (0, 61875)    1
  (0, 71077)    1
  (0, 87084)    1
  (0, 108307)   2
  (0, 103477)   1
  (0, 66977)    1
  (0, 73996)    1
  (0, 67944)    1
  (0, 87176)    2
  (0, 19907)    1
  (0, 109606)   1
  (0, 7492)     1
  (0, 108457)   1
  (0, 49181)    1
  (0, 8547)     3
  (0, 87199)    1
  (0, 25908)    1
  (0, 115789)   1
  (0, 108377)   2
  (0, 78426)    2
  (0, 97531)    1
  (0, 23056)    1
  (0, 55714)    2
  (0, 48731)    2
  (0, 40110)    2
  (0, 88814)    2
  (0, 52757)    2
```

In [9]:

```
#   GridsearchCV will build classifier specified (logistic regression),
# with the optimal C value, with 3-fold cross validation by defalut.
# So, we don't need to build a classifier again with this model..
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
log_reg_l2 = LogisticRegression(penalty='l2')
log_reg_l2_grid_clf = GridSearchCV(log_reg_l2, param_grid=param_grid,n_jobs=-1, verbose
=40)
log_reg_l2_grid_clf
```

Out[9]:

```
GridSearchCV(cv=None, error_score='raise',
       estimator=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),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=40)
```

lets split the data into train and test data

In [10]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data)

y_train.shape
```

Out[10]:

```
(394360,)
```

In [11]:

```
from datetime import datetime
startTime = datetime.now()

#lets run GridsearchCV on our data
log_reg_l2_grid_clf.fit(x_train, y_train)

print(datetime.now() - startTime)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   29.7s
[Parallel(n_jobs=-1)]: Done   2 out of  15 | elapsed:   32.4s remaining:
3.5min
[Parallel(n_jobs=-1)]: Done   3 out of  15 | elapsed:   35.0s remaining:
2.3min
[Parallel(n_jobs=-1)]: Done   4 out of  15 | elapsed:  1.5min remaining:
4.1min
[Parallel(n_jobs=-1)]: Done   5 out of  15 | elapsed:  1.6min remaining:
3.1min
[Parallel(n_jobs=-1)]: Done   6 out of  15 | elapsed:  1.7min remaining:
2.6min
[Parallel(n_jobs=-1)]: Done   7 out of  15 | elapsed:  4.6min remaining:
5.2min
[Parallel(n_jobs=-1)]: Done   8 out of  15 | elapsed:  4.6min remaining:
4.0min
[Parallel(n_jobs=-1)]: Done   9 out of  15 | elapsed:  5.4min remaining:
3.6min
[Parallel(n_jobs=-1)]: Done  10 out of  15 | elapsed: 10.8min remaining:
5.4min
[Parallel(n_jobs=-1)]: Done  11 out of  15 | elapsed: 11.2min remaining:
4.1min
[Parallel(n_jobs=-1)]: Done  12 out of  15 | elapsed: 11.8min remaining:
2.9min
[Parallel(n_jobs=-1)]: Done  13 out of  15 | elapsed: 12.4min remaining:
1.9min
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 15.4min remaining:
  0.0s
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 15.4min finished

0:21:50.554847
```

In [12]:

```
log_reg_l2_grid_clf.best_params_ ,log_reg_l2_grid_clf.best_estimator_, log_reg_l2_grid_
clf.best_score_
```

Out[12]:

```
({'C': 1},
 LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tru
e,
           intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
           penalty='l2', random_state=None, solver='liblinear', tol=0.000
1,
           verbose=0, warm_start=False),
 0.95510447307029112)
```

In [13]:

```
y_true, y_pred = y_test, log_reg_l2_grid_clf.predict(x_test)
```

In [14]:

```
from sklearn.metrics import precision_recall_fscore_support, classification_report

print(classification_report(y_test, y_pred))
print(precision_recall_fscore_support(y_test, y_pred, average='micro'))
```

```
             precision    recall  f1-score   support

   negative       0.89      0.83      0.86     20247
   positive       0.97      0.98      0.98    111207

avg / total       0.96      0.96      0.96    131454

(0.95755169108585514, 0.95755169108585514, 0.95755169108585514, None)
```

In [15]:

```
from sklearn.metrics import roc_curve, auc, roc_auc_score

y_pred = np.where(y_pred == 'positive', 1, 0)
y_true = np.where(y_true == 'positive', 1, 0)

fpr, tpr, threshold = roc_curve(y_true, y_pred)
print("FPR: {}, \nTPR: {}, \nAUC : {}".format(fpr[1], tpr[1], auc(fpr,tpr)))
```

```
FPR: 0.1690620832715958,
TPR: 0.980603738973266,
AUC : 0.9057708278508351
```

2. GridSearchCV with l1 regularizer

In [16]:

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
log_reg_l1 = LogisticRegression(penalty='l1')
log_reg_l1_grid_clf = GridSearchCV(log_reg_l1, param_grid=param_grid,n_jobs=-1, verbose
=40)
log_reg_l1_grid_clf
```

Out[16]:

```
GridSearchCV(cv=None, error_score='raise',
       estimator=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),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=40)
```

In [17]:

```
from datetime import datetime
startTime = datetime.now()

#Lets run GridsearchCV on our data
log_reg_l1_grid_clf.fit(x_train, y_train)

print(datetime.now() - startTime)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits

[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   23.9s
[Parallel(n_jobs=-1)]: Done   2 out of  15 | elapsed:   30.1s remaining:
3.3min
[Parallel(n_jobs=-1)]: Done   3 out of  15 | elapsed:   31.9s remaining:
2.1min
[Parallel(n_jobs=-1)]: Done   4 out of  15 | elapsed:   37.5s remaining:
1.7min
[Parallel(n_jobs=-1)]: Done   5 out of  15 | elapsed:   39.0s remaining:
1.3min
[Parallel(n_jobs=-1)]: Done   6 out of  15 | elapsed:   44.6s remaining:
1.1min
[Parallel(n_jobs=-1)]: Done   7 out of  15 | elapsed:  1.6min remaining:
1.9min
[Parallel(n_jobs=-1)]: Done   8 out of  15 | elapsed:  1.7min remaining:
1.5min
[Parallel(n_jobs=-1)]: Done   9 out of  15 | elapsed:  1.7min remaining:
1.2min
[Parallel(n_jobs=-1)]: Done  10 out of  15 | elapsed:  1.9min remaining:
57.8s
[Parallel(n_jobs=-1)]: Done  11 out of  15 | elapsed:  2.1min remaining:
45.7s
[Parallel(n_jobs=-1)]: Done  12 out of  15 | elapsed:  2.3min remaining:
34.6s
[Parallel(n_jobs=-1)]: Done  13 out of  15 | elapsed:  2.8min remaining:
25.5s
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:  3.0min remaining:
  0.0s
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:  3.0min finished

0:03:33.333429
```

In [18]:

```
log_reg_l1_grid_clf.best_params_ ,log_reg_l1_grid_clf.best_estimator_, log_reg_l1_grid_
clf.best_score_
```

Out[18]:

```
({'C': 1},
 LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tru
e,
           intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
           penalty='l1', random_state=None, solver='liblinear', tol=0.000
1,
           verbose=0, warm_start=False),
 0.95403692058018053)
```

In [19]:

```
y_true, y_pred = y_test, log_reg_l1_grid_clf.predict(x_test)
```

In [20]:

```
print(classification_report(y_test, y_pred))
print(precision_recall_fscore_support(y_test, y_pred, average='micro'))
```

```
             precision    recall  f1-score   support

   negative       0.88      0.83      0.85     20247
   positive       0.97      0.98      0.97    111207

avg / total       0.96      0.96      0.96    131454

(0.95654753754164956, 0.95654753754164956, 0.95654753754164956, None)
```

In [21]:

```
from sklearn.metrics import roc_curve, auc, roc_auc_score

y_pred = np.where(y_pred == 'positive', 1, 0)
y_true = np.where(y_true == 'positive', 1, 0)

fpr, tpr, threshold = roc_curve(y_true, y_pred)
print("FPR: {}, \nTPR: {}, \nAUC : {}".format(fpr[1], tpr[1], auc(fpr,tpr)))
```

```
FPR: 0.17098829456215736,
TPR: 0.979767460681432,
AUC : 0.9043895830596373
```

Feature Importance

In [22]:

```
best_estimator = log_reg_l1_grid_clf.best_estimator_
```

In [23]:

```
weights_vectors = pd.DataFrame()
weights_vectors['feature'] = count_vec_total.get_feature_names()
weights_vectors['weight'] = np.transpose(best_estimator.coef_)
# convert positive and negative weights to absolute values
weights_vectors['weight'] = weights_vectors['weight'].map(lambda x: abs(x))
```

In [24]:

```
sorted_weights = weights_vectors.sort_values(by='weight', ascending=False)
sum(sorted_weights['weight'] == 0)
```

Out[24]:

```
108454
```

Important features are

In [25]:

```
imp_features = sorted_weights[sorted_weights['weight'] >= 0.0001]['feature'].values
```

In [26]:

```
imp_features, len(imp_features)
```

Out[26]:

```
(array(['excellently', 'suman', 'maxes', ..., 'plus', 'development', 'char
t'], dtype=object),
 12567)
```

top 100 most important features are..

In [27]:

```
imp_features[:100]
```

Out[27]:

```
array(['excellently', 'suman', 'maxes', 'blowout', 'hubs', 'b000sqn3og',
       'saquin', 'vagan', 'ghastly', 'hestitate', 'ramada', 'upgrading',
       'looming', 'abottle', 'yirgacheffe', 'sedentary', 'fragmentatized',
       'bumpy', 'coincident', 'pallets', 'leap', 'von', 'incurred',
       'replentishment', 'undeniably', 'unwearable', 'becausei', 'energy
s',
       'emeraldforest', 'chedder', 'b000et93n2', 'juiciest', 'storebrand',
       'b003crknpk', 'consignment', 'seperatly', 'totalling', 'abominabl
e',
       'decar', 'blehhhh', 'glumpy', 'shippments', 'b000etc9ii', '280mg',
       'weakest', 'cubbard', 'liva', 'deducting', 'devastated', 'chasen',
       'botch', 'nome', 'competed', 'styrofoamy', 'heroes', 'repack',
       'overrated', 'simpson', 'unlemoned', 'phyllis', 'hazards',
       '55pound', 'healhy', 'upswing', 'danhill', 'stitch', 'subjectivel
y',
       'ethylene', 'tranquility', 'wrinkly', 'brûl', 'publicized',
       'unmatched', 'biotic', 'skewed', 'commands', 'jp', 'dissapointing',
       'keratin', 'totaller', 'urinated', 'todd', 'lifespans', '15mo',
       'chagrin', 'contaminating', 'diffuser', 'truffiere', 'tribute',
       'webb', 'ellen', 'ensues', 'ehhhh', 'abbreviated', 'tooooooo',
       'freind', 'hfs', 'skyrocketd', 'sham', 'trucks'], dtype=object)
```

Training and Testing Split

In [6]:

```
my_rand_state=0
test_size=0.25
```

In [7]:

```
from sklearn.model_selection import train_test_split
```

In [11]:

```
X = (reviews_sst[kept_cols].iloc[:,1]).tolist()
y = (reviews_sst[kept_cols].iloc[:,0]).tolist()
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
                                                    random_state=my_rand_state)
```

Text

In [13]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [14]:

```
#set max_features to minimize training time
#also, I didn't apply LDA-based dimensionality reduction
tfidf=TfidfVectorizer(lowercase=False,max_features=200)
```

Classification Models

In [15]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [16]:

```
nb_clf=GaussianNB()
priors=[None]
```

In [17]:

```
qda_clf=QuadraticDiscriminantAnalysis()
reg_param=[0.0, 0.25, 0.5, 0.75]
```

In [18]:

```
log_clf=LogisticRegression(penalty='l2')
C=[0.001 , 0.01, 10, 100,1000]
```

In [19]:

```
rf_clf=RandomForestClassifier()
n_estimators=[100,200]
max_features=[.1,.3,.5]
```

In [20]:

```
class_weight=['balanced']
class_weight.extend([{1: w} for w in [1, 2, 10]])
```

Creating Pipelines

In [21]:

```
from imblearn import pipeline#needed if mixing imblearn with sklearn classes
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
```

In [22]:

```
n_jobs=4
```

In [23]:

```
n_folds=10
skfold = StratifiedKFold(n_splits=n_folds,random_state=my_rand_state, shuffle=False)
```

In [24]:

```
from sklearn.base import BaseEstimator, TransformerMixin
```

In [25]:

```
class DenseTransformer(BaseEstimator, TransformerMixin):

    def transform(self, X, y=None, **fit_params):
        return X.todense()

    def fit_transform(self, X, y=None, **fit_params):
        self.fit(X, y, **fit_params)
        return self.transform(X)

    def fit(self, X, y=None, **fit_params):
        return self
```

Naive Bayes Estimators

In [26]:

```
nb_clf_b = pipeline.Pipeline(steps=[('tfidf',tfidf),('to_dense', DenseTransformer()),(
'clf',nb_clf)])
nb_clf_est_b = GridSearchCV(estimator=nb_clf_b,cv=skfold,
              scoring='roc_auc',n_jobs=n_jobs,
              param_grid=dict(clf__priors=priors))
```

Logistic Estimators

In [28]:

```
log_clf_b = pipeline.Pipeline(steps=[('tfidf',tfidf),('clf',log_clf)])
log_clf_est_b = GridSearchCV(estimator=log_clf_b,cv=skfold,
              scoring='roc_auc',n_jobs=n_jobs,
              param_grid=dict(clf__C=C,
                              clf__class_weight=class_weight))
```

Random Forest Estimators

In [29]:

```
rf_clf_b = pipeline.Pipeline(steps=[('tfidf',tfidf),('clf',rf_clf)])
rf_clf_est_b = GridSearchCV(estimator=rf_clf_b,cv=skfold,
              scoring='roc_auc',n_jobs=n_jobs,
              param_grid=dict(clf__n_estimators=n_estimators,
                              clf__max_features=max_features,
                              clf__class_weight=class_weight))
```

Fitting Estimators

In [30]:

```
from sklearn.externals import joblib
```