Pradip Dharam

5. Logistic Regression on Amazon BoW TFIDF

forked from 4. Naive Bayes on Amazon BoW TFIDF by Pradip Dharam (+0/−0)

last run 2 hours ago · IPython Notebook HTML
using data from multiple data sources · Private Make Public

5] Apply Logistic Regression on Amazon product reviews data-set [M]

Implement Logistic Regression on Amazon reviews dataset

1. Please do train, cross validation and test OR train and CV and be combined and k' fold validation can be performed.
2. Use grid search cross validation (GridSearchCV) and random search cross validation (RandomSearchCV) to find optimal lamda (hyper parameter)
3. Please try L1 and L2 regulariser. Elastic net can be skipped since its not implemented in sklearn. We have high dimensional data
4. Try L1 regilarization, keep increasing lambda and report error as well as sparsity (number of non zero elements in the weight vector w star) As lambda increases for L1 regularization, show whats happening with error and sparsity. As lsmbda increases drastically; model underfits, error also increases, sparsity also increases, performance drops (precision and recall will drop). This point we need to check in case we use low latency application where results are required in miliseconds
5. Get feature importance. Check for multi-coliniarity (pertubiation test) - simply add some noise to the data which can be epsilon belongs to N(0, 0.01), compare the weights before and after adding noice, if weights changes a lot, we can declare that there is multicoliniarity problem, if weights dont change much its not a big problem. Also get most important features

PLEASE DONT FORGET TO DO COLUMN STANDARDIZATION OTHERWISE ALL LOGISTIC REGRESSION FAILS, do it at very start

This excercise id important since in real world, LR is applied many many times. This is often baseline model, this is the first model which will be applied in most real world cases

OPTIONAL:

1. Get accuracy, precision, recall, f1 score, confusion matrix, TNR, FNR, FPR, TPR

SOLUTION:

Importing required lib's

In [1]:

```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

from sklearn.cross_validation import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn import preprocessing

from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, classification_report, log_loss
from sklearn.metrics import precision_score, recall_score, accuracy_score
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: Thi
s module was deprecated in version 0.18 in favor of the model_selection module into which all
the refactored classes and functions are moved. Also note that the interface of the new CV ite
rators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/opt/conda/lib/python3.6/site-packages/sklearn/grid_search.py:42: DeprecationWarning: This mod
ule was deprecated in version 0.18 in favor of the model_selection module into which all the r
efactored classes and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

Creting connection to work with '../input/database.sqlite' sqllite database file

In [2]:

```
# using the SQLite Table to read data.
con = sqlite3.connect('../input/amazon-fine-food-reviews/database.sqlite')
```

#

TEXT PREPROCESSING Begins Here

Filtering only positive and negative reviews i.e. not taking into consideration those reviews with Score=3

In [3]:
```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

Duplicate entries like below needs tobe removed and instead we need to beep one record

In [4]:
```
# Query to show Duplicate entries
display= pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 AND UserId="AR5J8UI46CUR
R" ORDER BY ProductID""", con)
display
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha	2	2	5	119957760(

				Krishnan				
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600

Removing duplicates

- Sorting the data first
- filtering the records on features "UserId","ProfileName","Time" and "Text" then just keeping first occurence of rest of the features

In [5]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind=
'quicksort', na_position='last')

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first',
inplace=False)
final.shape
```

Out[5]:
```
(364173, 10)
```

Checking to see how much % of data still remains after duplicate records removal

In [6]:
```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[6]:
```
69.25890143662969
```

Indentified invad records as per business logic which needs to removed

- As per business logic, records which are valid where HelpfulnessNumerator is less than HelpfulnessDenomirator and invalid records should be removed

In [7]:
```
display= pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3 AND Id=44737 OR Id=64422 O
RDER BY ProductID""", con)
display
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	122489280(
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	121288320(

Removing invalid records as per above comment

In [8]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
final.shape
```

Out[8]:
```
(364171, 10)
```

Identifying whether review text contains HTML tags

- One of the review text has HTML tags which needs to be removed as part of data pre processing

In [9]:
```
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this point, I consider mys
elf a connoisseur of children's books and this is one of the best. Santa Clause put this under
the tree. Since then, we've read it perpetually and he loves it.<br /><br />First, this book t
aught him the months of the year.<br /><br />Second, it's a pleasure to read. Well suited to
1.5 y/o old to 4+.<br /><br />Very few children's books are worth owning. Most should be borro
wed from the library. This book, however, deserves a permanent spot on your shelf. Sendak's be
st.
```

Avoiding stop words removal

- Removing stopwords like below will manipulates the meaning of scentense and this will spoil the r=end result of the trained model up to some extent **so its bettet to keep stop words
- **Some stop words:** before, after, above, below, up, down, on, off, over under, again, further, all, any, each, no, nor, don't, not nor'

In [10]:
```
from nltk.corpus import stopwords
stopwords.words('english')
# stop = set(stopwords.words('english')) #set of stopwords
# if(cleaned_words.lower() not in stop):
```

Out[10]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
```

```
                "you'll",
                "you'd",
                'your',
                'yours',
                'yourself',
                'yourselves',
                'he',
                'him',
                'his',
                'himself',
                'she',
                "she's",
                'her',
                'hers',
                'herself',
                'it',
                "it's",
                'its',
                'itself',
                'they',
                'them',
                'their',
                'theirs',
                'themselves',
                'what',
                'which',
                'who',
                'whom',
                'this',
                'that',
                "that'll",
                'these',
                'those',
                'am',
                'is',
                'are',
                'was',
                'were',
                'be',
                'been',
                'being',
                'have',
                'has',
                'had',
                'having',
                'do',
```

```
                'does',
                'did',
                'doing',
                'a',
                'an',
                'the',
                'and',
                'but',
                'if',
                'or',
                'because',
                'as',
                'until',
                'while',
                'of',
                'at',
                'by',
                'for',
                'with',
                'about',
                'against',
                'between',
                'into',
                'through',
                'during',
                'before',
                'after',
                'above',
                'below',
                'to',
                'from',
                'up',
                'down',
                'in',
                'out',
                'on',
                'off',
                'over',
                'under',
                'again',
                'further',
                'then',
                'once',
                'here',
                'there',
                'when',
```

```
                'where',
                'why',
                'how',
                'all',
                'any',
                'both',
                'each',
                'few',
                'more',
                'most',
                'other',
                'some',
                'such',
                'no',
                'nor',
                'not',
                'only',
                'own',
                'same',
                'so',
                'than',
                'too',
                'very',
                's',
                't',
                'can',
                'will',
                'just',
                'don',
                "don't",
                'should',
                "should've",
                'now',
                'd',
                'll',
                'm',
                'o',
                're',
                've',
                'y',
                'ain',
                'aren',
                "aren't",
                'couldn',
                "couldn't",
                'didn',
```

```
"didn't",
'doesn',
"doesn't",
'hadn',
"hadn't",
'hasn',
"hasn't",
'haven',
"haven't",
'isn',
"isn't",
'ma',
'mightn',
"mightn't",
'mustn',
"mustn't",
'needn',
"needn't",
'shan',
"shan't",
'shouldn',
"shouldn't",
'wasn',
"wasn't",
'weren',
"weren't",
'won',
"won't",
'wouldn',
"wouldn't"]
```

Stemming, HTML removal, cleaning punctuation or special characters, Lemmatization, converting to lover case

- Also removing the words which has length 1
- **This section is commented since this takes time and one time processing. Pre-processing results are stored in csv file and every time we will get data from csv file**

In [11]:

```
"""
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
#from nltk.stem import PorterStemmer
#from nltk.stem.wordnet import WordNetLemmatizer
```

```
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned

#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>1)):
                s=(sno.stem(cleaned_words.lower())).encode('utf8')
                #Comment above line to remove stemming an replce this line with below line
                #s=cleaned_words.lower()
                filtered_sentence.append(s)
                if (final['Score'].values)[i] == 'positive':
                    all_positive_words.append(s) #list of all words used to describe positive rev
iews
                if(final['Score'].values)[i] == 'negative':
                    all_negative_words.append(s) #list of all words used to describe negative rev
iews reviews
            else:
                continue
    #print("filtered_sentence: ",filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**************************************************************************")

    # Trimming b' and ' from the left and right respectively
    str2=str(str1).lstrip('b\'').rstrip('\'')
    #print("str2:",str2)
```

```
    final_string.append(str2)
    i+=1

final['CleanedText']=final_string #adding a column of CleanedText which displays the data after p
re-processing of the review
"""
```

Out[11]:

```
'\nimport re\n# Tutorial about Python regular expressions: https://pymotw.com/2/re/\nimport st
ring\n#from nltk.stem import PorterStemmer\n#from nltk.stem.wordnet import WordNetLemmatizer\n
\nsno = nltk.stem.SnowballStemmer(\'english\') #initialising the snowball stemmer\n\ndef clean
html(sentence): #function to clean the word of any html-tags\n    cleanr = re.compile(\'<.*?>
\')\n    cleantext = re.sub(cleanr, \' \', sentence)\n    return cleantext\ndef cleanpunc(sent
ence): #function to clean the word of any punctuation or special characters\n    cleaned = re.
sub(r\'[?|!|\'|"|#]\',r\'\',sentence)\n    cleaned = re.sub(r\'[.|,|)|(|\\|/]\',r\' \',cleane
d)\n    return  cleaned\n\n#Code for implementing step-by-step the checks mentioned in the pre
-processing phase\n# this code takes a while to run as it needs to run on 500k sentences.\ni=0
\nstr1=\' \'\nfinal_string=[]\nall_positive_words=[] # store words from +ve reviews here\nall_
negative_words=[] # store words from -ve reviews here.\ns=\'\'\nfor sent in final[\'Text\'].va
lues:\n    filtered_sentence=[]\n    #print(sent);\n    sent=cleanhtml(sent) # remove HTml tag
s\n    for w in sent.split():\n        for cleaned_words in cleanpunc(w).split():\n
if((cleaned_words.isalpha()) & (len(cleaned_words)>1)):    \n                s=(sno.stem(clean
ed_words.lower())).encode(\'utf8\')\n                #Comment above line to remove stemming an
replce this line with below line\n                #s=cleaned_words.lower()\n                fi
ltered_sentence.append(s)\n                if (final[\'Score\'].values)[i] == \'positive\': \n
                    all_positive_words.append(s) #list of all words used to describe positive
reviews\n                if(final[\'Score\'].values)[i] == \'negative\':\n
all_negative_words.append(s) #list of all words used to describe negative reviews reviews
            \n            else:\n                continue \n    #print("filtered_sentence: ",fi
ltered_sentence)\n    str1 = b" ".join(filtered_sentence) #final string of cleaned words\n
#print("**********************************************************************")\n    \n    #
Trimming b\' and \' from the left and right respectively\n    str2=str(str1).lstrip(\'b\'\').r
strip(\'\'\')\n    #print("str2:",str2)\n    final_string.append(str2)  \n    i+=1\n
  \nfinal[\'CleanedText\']=final_string #adding a column of CleanedText which displays the dat
a after pre-processing of the review \n'
```

TEXT PREPROCESSING Ends Here

#

Storing pre-processed data to csv file

- **This section is commented since this takes time and one time processing. Pre-processing results are stored in csv file and every time we will get data from csv file**

In [12]:
```
"""
final_text_processed=pd.DataFrame()
final_text_processed['Text'] = final['Text']
final_text_processed['CleanedText'] = final['CleanedText']
final_text_processed['Time'] = final['Time']
final_text_processed['Score'] = final['Score']
final_text_processed.to_csv('final_text_processed.csv')
"""
```

Output

#

Defining below functions

1. Function NaiveBayes10fold_and_MSE

- This function takes training data both X_train and Y_train to train the model
- This function also takes argument algo where in we can provide whether to use brute or kd_tree
- We need to do time based splitting but not random while training 10 fold cross validation kNN model; I am using TimeSeriesSplit and passing 10 as argument and my traing data is already sorted in ascending order.
- I did added print statements to undestand the execution after writing code. Now those print statements are commented.
- This function plots the graph "Number of Neighbors K')" against "Misclassification Error"
- It takes average accuracy of 10 folds for each k and then calculates the mis-classification error as (1-average_accuracy_for_each_k)
- It then select the optimal k such that misclassification error is minimum and returns the same

In [13]:
```
def LogisticRegression5fold_and_MSE(x_train, y_train, x_test, y_test, search_type='GridSearchC
V', regulariser='l2'):
    lambdas_reciprocals = {'C': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10
**4, 10**5]}

    model = GridSearchCV(LogisticRegression(penalty=regulariser), lambdas_reciprocals, scoring
= 'accuracy', cv=5)
```

```
    if (search_type == 'RandomizedSearchCV'):
        model = RandomizedSearchCV(LogisticRegression(penalty=regulariser), lambdas_reciprocals
, scoring = 'accuracy', cv=5)

    print("This is 5 fold ",search_type, " with ", regulariser, " regulariser Logistic Regressi
on" )

    model.fit(x_train, y_train)

    print(model.best_estimator_)
    print(model.score(x_test, y_test))
```

2. Function knn_with_optimal_alpha

- Here, lets test the model over unknown data which is x_test and y_test here
- Train the model using x_train and y_train; then predict using x_test
- Then get the accuracy using prected class labels and y_train and report the accuracy for optimal k

In [14]:

```
def LogisticRegression_with_optimal_C(x_train, y_train, x_test, y_test, optimal_C, regulariser=
'l2'):
    clf = LogisticRegression(C=optimal_C, penalty=regulariser);
    print("This is Logistic Regression with ", regulariser, " regulariser for optimal C=",optim
al_C )
    clf.fit(x_train, y_train);
    pred = clf.predict(x_test)
    acc = accuracy_score(y_test, pred) * 100
    print('Able to achieve ', round(acc,2), '% of test accuracy' )
    print("Classification Report: ")
    print(classification_report(y_test, pred))

    c_metrix = confusion_matrix(y_test,pred)
    tn, fp, fn, tp = confusion_matrix(y_test,pred).ravel()
    sns.heatmap(c_metrix.T, square=True, annot=True, fmt='d', cbar=True,cmap='RdYlBu_r',
        xticklabels=['negative','positive'], yticklabels=['negative','positive'])
    plt.title("Confusion Metrix")
    plt.xlabel('True Label')
    plt.ylabel('Predicted Label');

    print("True Negative Rate: ",(tn/(tn+fp))*100)
    print("False Negative Rate: ",(fn/(fn+tp))*100)
    print("False Positive Rate: ",(fp/(tn+fp))*100)
    print("True Positive Rate: ",(tp/(fn+tp))*100)
```

```
str_model="Logistic Regression"
from prettytable import PrettyTable
model_pretty_table = PrettyTable()
model_pretty_table.field_names = ["Model","Regularizer", "Hyper Parameter C", "Test Error"]

model_pretty_table.add_row([str_model, regulariser, optimal_C, round(100-acc,2)])
print(model_pretty_table)
```

3. Function to get

In [15]:

#

Reading data from pre-processed csv file

- **Preprocessed data was stored in csv file. Reading the same into panda data frame variable data**

In [15]:
```
final_text_processed=pd.read_csv('../input/final-text-processed/final_text_processed.csv')
```

In [16]:
```
final_text_processed.shape
```

Out[16]:
```
(364171, 5)
```

#

Sorting data in ascending order of time since time based splitting needs to be done

In [17]:
```
#Sorting the data on ascending order of Time since time based splitting needs to be done further
final_text_processed=final_text_processed.sort_values(['Time'], ascending=1)
print("Time min:", final_text_processed['Time'].min())
print("Time max:", final_text_processed['Time'].max())
print("\nPrinting time feature to ensure whether data is sorted in ascending order of time")
```

```
final_text_processed['Time']
```

```
Time min: 939340800
Time max: 1351209600

Printing time feature to ensure whether data is sorted in ascending order of time
```

Out[17]:

```
0          939340800
30         940809600
424        944092800
330        944438400
423        946857600
245        947376000
308        948240000
215        948672000
261        951523200
325        959990400
427        959990400
241        961718400
242        962236800
485        965001600
837        965779200
868        965779200
249        966297600
296        970531200
844        975974400
360        977184000
329        978134400
845        982800000
425        992217600
342        997228800
270       1001289600
855       1003795200
353       1004054400
32        1009324800
998       1010275200
331       1012780800
             ...
331313    1351209600
274064    1351209600
941       1351209600
273941    1351209600
331386    1351209600
363346    1351209600
215652    1351209600
```

```
331387    1351209600
273648    1351209600
18387     1351209600
354060    1351209600
354041    1351209600
132404    1351209600
272372    1351209600
1637      1351209600
361679    1351209600
332259    1351209600
353879    1351209600
16845     1351209600
214181    1351209600
16642     1351209600
332848    1351209600
333441    1351209600
353804    1351209600
139151    1351209600
139298    1351209600
333542    1351209600
139690    1351209600
213087    1351209600
57313     1351209600
Name: Time, Length: 364171, dtype: int64
```

In [18]:

```
#import numpy as np
#from sklearn.cross_validation import train_test_split
#a = np.arange(10).reshape((5, 2))
#b = np.arange(5)
#print("Array a: \n",a); print("Array b: \n",b);

#a_train, a_test, b_train, b_test = train_test_split(a, b,  test_size=0.3,  random_state=0)
#print("a_train :\n", a_train)
#print("b_train :\n", b_train)
```

#

Selecting first 100k records for processing as a sample

In [19]:

```
num_of_points=100000
```

```
data = final_text_processed[0:num_of_points]

print("Min and max time values for sample data :")
print("Time min:", data['Time'].min())
print("Time max:", data['Time'].max())

X = data['CleanedText']
Y = np.array([1 if x=='positive' else 0 for x in data['Score']])
XYTime=data['Time']
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
#print(X); print(Y)
from scipy import stats
stats.describe(Y)
```

```
Min and max time values for sample data :
Time min: 939340800
Time max: 1277164800
X Shape :  (100000,)  X Ndim:  1
Y Shape :  (100000,)  Y Ndim:  1
```

Out[19]:

```
DescribeResult(nobs=100000, minmax=(0, 1), mean=0.87729, variance=0.10765333243332435, skewnes
s=-2.299819344191714, kurtosis=3.28916901591841)
```

#

Splitting dataset into train and test with 70:30 ratio**

In [20]:

```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]
XYTime_train = XYTime[:boundry]; XYTime_test = XYTime[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("x train Shape : ",x_train.shape, " x Ndim: ",x_train.ndim); print("y train Shape : ",y_t
rain.shape, " y Ndim: ",y_train.ndim)
print("x test Shape : ",x_test.shape, " x Ndim: ",x_test.ndim); print("y test Shape : ",y_test.
```

```
shape, " y Ndim: ",y_test.ndim)

print("\nMin and max time values for time for both train and test :")
print("Train Time min:", XYTime_train.min())
print("Train Time max:", XYTime_train.max())
print("Test Time min:", XYTime_test.min())
print("Test Time max:", XYTime_test.max())
```

```
Boundry:  70000
x train Shape :  (70000,)  x Ndim:  1
y train Shape :  (70000,)  y Ndim:  1
x test Shape :  (30000,)  x Ndim:  1
y test Shape :  (30000,)  y Ndim:  1

Min and max time values for time for both train and test :
Train Time min: 939340800
Train Time max: 1256083200
Test Time min: 1256083200
Test Time max: 1277164800
```

In [21]:

```
"""
num_of_points=20
x_test_temp = x_test[0:num_of_points]
x_test=x_test_temp
print("x_test shape: ",x_test.shape)

#BoW
count_vect_train = CountVectorizer(ngram_range=(1,2)) #in scikit-learn
final_counts_bow_train = count_vect_train.fit_transform(x_train.values)
final_counts_bow_train.shape
print("final_counts_bow_train shape: ",final_counts_bow_train.shape)

features_count_vect_train = count_vect_train.get_feature_names()

count_vect_test = CountVectorizer(ngram_range=(1,2), vocabulary = features_count_vect_train)
counts_bow_test = count_vect_test.fit(x_train.values)
final_counts_bow_test = counts_bow_test.transform(x_test.values)

print("final_counts_bow_test shape: ",final_counts_bow_test.shape)
test_length=final_counts_bow_test.shape[0]
print("test_length: ",test_length)
print("test_length type: ",type(test_length))

test_length=final_counts_bow_test.shape[0]
```

```
splits=4
pred=[]

for i in range(0,test_length,splits):
    print(i, " ",i+splits-1)
    x_test_split = final_counts_bow_test[i:i+splits]
    print("x_test_split shape: ",x_test_split.shape)
    pred_splitted=knn_optimal.predict(x_test_split)
    #print("pred_splitted: ",pred_splitted)
    #list_pred_splitted=list(pred_splitted)
    print("list_pred_splitted : ", list_pred_splitted)
    print("pred_splitted type: ",type(pred_splitted))
    print("pred_splitted shape: ",pred_splitted.shape)
    pred=np.concatenate([pred,pred_splitted], axis=0)
    print("pred: ",pred)
    print("pred type: ",type(pred))
    print("pred shape: ",pred.shape)
print('-----------------------------------')
print("pred: ",pred)
print("pred type: ",type(pred))
print("pred shape: ",pred.shape)
"""
```

Out[21]:

```
'\nnum_of_points=20\nx_test_temp = x_test[0:num_of_points]\nx_test=x_test_temp\nprint("x_test
shape: ",x_test.shape)\n\n#BoW\ncount_vect_train = CountVectorizer(ngram_range=(1,2)) #in scik
it-learn\nfinal_counts_bow_train = count_vect_train.fit_transform(x_train.values)\nfinal_count
s_bow_train.shape\nprint("final_counts_bow_train shape: ",final_counts_bow_train.shape)\n\nfea
tures_count_vect_train = count_vect_train.get_feature_names()\n\ncount_vect_test = CountVector
izer(ngram_range=(1,2), vocabulary = features_count_vect_train) \ncounts_bow_test = count_vect
_test.fit(x_train.values)\nfinal_counts_bow_test = counts_bow_test.transform(x_test.values)\n
\nprint("final_counts_bow_test shape: ",final_counts_bow_test.shape)\ntest_length=final_counts
_bow_test.shape[0]\nprint("test_length: ",test_length)\nprint("test_length type: ",type(test_l
ength))\n\ntest_length=final_counts_bow_test.shape[0]\nsplits=4\npred=[]\n\nfor i in range(0,t
est_length,splits):\n    print(i, " ",i+splits-1)\n    x_test_split = final_counts_bow_test[i:
i+splits]\n    print("x_test_split shape: ",x_test_split.shape)\n    pred_splitted=knn_optima
l.predict(x_test_split)\n    #print("pred_splitted: ",pred_splitted)\n    #list_pred_splitted=
list(pred_splitted)\n    print("list_pred_splitted : ", list_pred_splitted)\n    print("pred_s
plitted type: ",type(pred_splitted))\n    print("pred_splitted shape: ",pred_splitted.shape)\n
    pred=np.concatenate([pred,pred_splitted], axis=0)\n    print("pred: ",pred)\n    print("pr
ed type: ",type(pred))\n    print("pred shape: ",pred.shape)\nprint(\'-----------------------
-----------\')\nprint("pred: ",pred)\nprint("pred type: ",type(pred))\nprint("pred shape: ",pr
ed.shape)\n'
```

#

Top prominent number of words

In [22]:
```
top_prominent=20
```

#

BAG OF WORDS

Defining CountVectorizer and defining the vocablary (i.e. unique words as per 1,2 gram) from training data

In [23]:
```
#BoW
count_vect_train = CountVectorizer(ngram_range=(1,2)) #in scikit-learn
count_vect_train.fit(x_train.values)
```

Out[23]:
```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

Storing vocablary of training data

In [24]:
```
features_count_vect_train_bow = count_vect_train.get_feature_names()
print("features shape: ", len(features_count_vect_train_bow))
features_count_vect_train_bow[:5]
```

```
features shape:  756453
```

Out[24]:
```
['aa', 'aa and', 'aa cell', 'aa coffe', 'aa doubl']
```

Using the vocablary generated by train data and generating the BoW vector representation or sparce metric representation for training data set

In [25]:
```
final_counts_bow_train = count_vect_train.transform(x_train.values)
final_counts_bow_train.shape
```

Out[25]:
```
(70000, 756453)
```

- **BoW vector representation for TEST DATA needs to be generated using VOCABLARY OF TRAIN DATA since test data is always UNKNOWN in real testing phase.**
- **And vocablary of test data having new words should not be considered while generating vector representation for test data, and vocablary or unique words from test data which are already present in trainign data set will be considered while generating vector representation for test data**

In [26]:
```
final_counts_bow_test = count_vect_train.transform(x_test.values)
final_counts_bow_test.shape
```

Out[26]:
```
(30000, 756453)
```

1. Logistic Regression with GridSearchCV and L2 Regularization for BoW

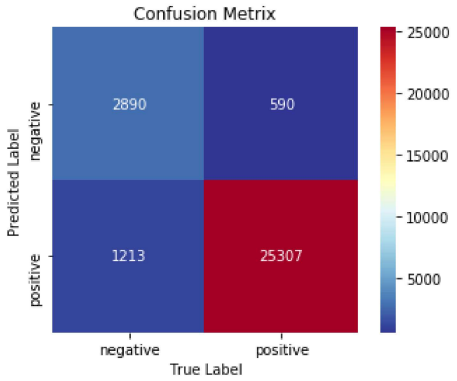
In [27]:
```
LogisticRegression5fold_and_MSE(final_counts_bow_train, y_train, final_counts_bow_test, y_test,
 search_type='GridSearchCV', regulariser='l2')
```

```
This is 5 fold  GridSearchCV  with  l2  regulariser Logistic Regression
LogisticRegression(C=1, 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)
0.9399
```

In [28]:
```
LogisticRegression_with_optimal_C(final_counts_bow_train, y_train, final_counts_bow_test, y_tes
t, 1 , regulariser='l2')
```

```
This is Logistic Regression with  l2  regulariser for optimal C= 1
```

```
Able to achieve  93.99 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.83      0.70      0.76      4103
          1       0.95      0.98      0.97     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  70.43626614672192
False Negative Rate:  2.2782561686681855
False Positive Rate:  29.56373385327809
True Positive Rate:  97.72174383133182
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l2     |         1         |    6.01    |
+---------------------+-------------+-------------------+------------+
```



OBSERVATIONS: Logistic Regression with GridSearchCV and L2 Regularization for BoW

- 1) We have got the best result for C=1 in the above cross validationThe optimal C is 1
- 2) **The test accuracy or accuracy over unknown data for optimal C = 1 is 93.99%**
- 3) **Only 70.43% of data points are predited to be negative out of all negative points. IT LOOKS LIKE OUR MODEL IS ALMOST ACCEPTABLE, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 4) Almost 2.27% of the positive data points are predictes as negative
- 5) **Almost 29.56% of data points which are actually negative and are predicted as positive by the model. IT LOOKS LIKE OUR MODEL IS ALMOST ACCEPTABLE, THIS IS BECAUSE OUR DATASET IS IMBALANCED**
- 6) Almost 97.72% of data points from the test data are predicted to be positive out of all actual positive data points

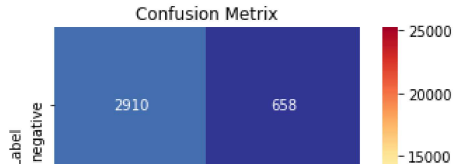
2. Logistic Regression with GridSearchCV and L1 Regularization for BoW

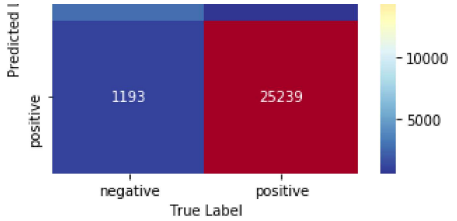
In [29]:
```
LogisticRegression5fold_and_MSE(final_counts_bow_train, y_train, final_counts_bow_test, y_test
, search_type='GridSearchCV', regulariser='l1')
```

```
This is 5 fold  GridSearchCV  with  l1  regulariser Logistic Regression
LogisticRegression(C=1000, 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)
0.9374
```

In [30]:
```
LogisticRegression_with_optimal_C(final_counts_bow_train, y_train, final_counts_bow_test, y_tes
t, 1000, regulariser='l1')
```

```
This is Logistic Regression with  l1  regulariser for optimal C= 1000
Able to achieve  93.83 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.82      0.71      0.76      4103
          1       0.95      0.97      0.96     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  70.92371435534974
False Negative Rate:  2.540834845735027
False Positive Rate:  29.076285644650255
True Positive Rate:  97.45916515426497
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l1     |        1000       |    6.17    |
+---------------------+-------------+-------------------+------------+
```





OBSERVATIONS: Logistic Regression with GridSearchCV and L1 Regularization for BoW

- 1) We have got the best result for C=1000 in the above cross validation. The optimal C is 1000
- 2) **The test accuracy or accuracy over unknown data for optimal C = 1000 is 93.84%**

3. Logistic Regression with RandomizedSearchCV and L2 Regularization for BoW

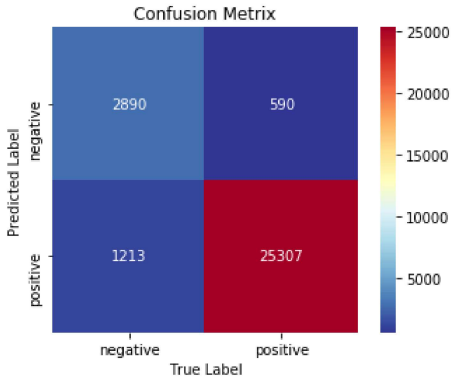
In [31]:
```
LogisticRegression5fold_and_MSE(final_counts_bow_train, y_train, final_counts_bow_test, y_test,
 search_type='RandomizedSearchCV', regulariser='l2')
```

```
This is 5 fold  RandomizedSearchCV  with  l2  regulariser Logistic Regression
LogisticRegression(C=1, 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)
0.9399
```

In [32]:
```
LogisticRegression_with_optimal_C(final_counts_bow_train, y_train, final_counts_bow_test, y_tes
t, 1 , regulariser='l2')
```

```
This is Logistic Regression with  l2  regulariser for optimal C= 1
Able to achieve  93.99 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.83      0.70      0.76      4103
          1       0.95      0.98      0.97     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  70.43626614672192
```

```
False Negative Rate:  2.2782561686681855
False Positive Rate:  29.56373385327809
True Positive Rate:  97.72174383133182
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l2     |         1         |    6.01    |
+---------------------+-------------+-------------------+------------+
```



OBSERVATIONS: Logistic Regression with RandomizedSearchCV and L2 Regularization for BoW

- 1) We have got the best result for C=1 in the above cross validationThe optimal C is 1
- 2) **The test accuracy or accuracy over unknown data for optimal C = 1 is 93.99%**

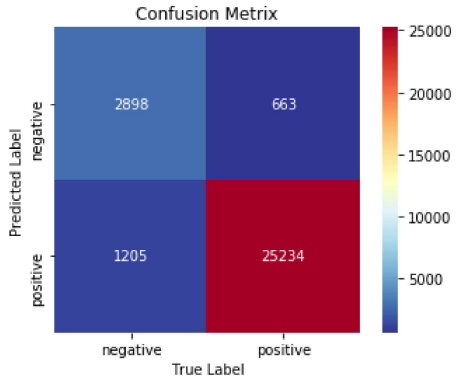
4. Logistic Regression with RandomizedSearchCV and L1 Regularization for BoW

In [33]:
```
LogisticRegression5fold_and_MSE(final_counts_bow_train, y_train, final_counts_bow_test, y_test,
 search_type='RandomizedSearchCV', regulariser='l1')
```

```
This is 5 fold  RandomizedSearchCV  with  l1  regulariser Logistic Regression
LogisticRegression(C=1000, 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)
0.9386666666666666
```

In [34]:
```
LogisticRegression_with_optimal_C(final_counts_bow_train, y_train, final_counts_bow_test, y_tes
t, 1000 , regulariser='l1')
```

```
This is Logistic Regression with  l1  regulariser for optimal C= 1000
Able to achieve  93.77 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.81      0.71      0.76      4103
          1       0.95      0.97      0.96     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  70.63124543017304
False Negative Rate:  2.560142101401707
False Positive Rate:  29.368754569826955
True Positive Rate:  97.43985789859829
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l1     |        1000       |    6.23    |
+---------------------+-------------+-------------------+------------+
```



OBSERVATIONS: Logistic Regression with RandomizedSearchCV and L1 Regularization for BoW

- 1) We have got the best result for C=1000 in the above cross validationThe optimal C is 1000
- 2) **The test accuracy or accuracy over unknown data for optimal C = 1000 is 93.78%**

#

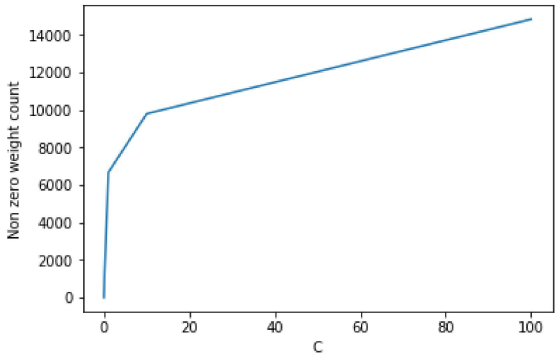
L1 REGULARIZER LR using BAG OF WORDS for increasing values of lambda

Taking increasing values of lambda

In [35]:
```
C = [100, 10, 1, 0.1 , 0.01, 0.001]
final_result = []

for c in C:
    result_temp = []
    clf = LogisticRegression(C=c, penalty='l1');
    clf.fit(final_counts_bow_train, y_train);
    pred = clf.predict(final_counts_bow_test)
    acc = accuracy_score(y_test, pred) * 100
    w = clf.coef_
    non_zero_count = np.count_nonzero(w)
    result_temp.append(c)
    result_temp.append(acc)
    result_temp.append(non_zero_count)
    final_result.append(result_temp)

arr_final_result = np.asarray(final_result)
arr_final_result
```

Out[35]:
```
array([[1.00000000e+02, 9.35433333e+01, 1.48180000e+04],
       [1.00000000e+01, 9.32733333e+01, 9.79000000e+03],
       [1.00000000e+00, 9.37500000e+01, 6.66200000e+03],
       [1.00000000e-01, 9.29400000e+01, 1.14600000e+03],
       [1.00000000e-02, 8.89033333e+01, 1.63000000e+02],
       [1.00000000e-03, 8.63633333e+01, 1.10000000e+01]])
```

In [36]:
```
plt.plot(arr_final_result[:,0], arr_final_result[:,2])
plt.xlabel("C")
plt.ylabel("Non zero weight count")
plt.show()
```



OBSERVATIONS: As C increases (lambda decreases), count of non zero weights increases. In other words, as lambda increases, sparsity increases since low number of non zero weight count.

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

Defining TfidfVectorizer and defining the vocablary (i.e. unique words as per 1,2 gram) from training data, fit function will do that

In [37]:
```
#TF-IDF
tf_idf_vect_train = TfidfVectorizer(ngram_range=(1,2)) #in scikit-learn
tf_idf_vect_train.fit(x_train.values)
```

Out[37]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

Storing vocablary of training data

In [38]:
```
features_count_vect_train_ifidf = tf_idf_vect_train.get_feature_names()
print("features shape: ", len(features_count_vect_train_ifidf))
features_count_vect_train_ifidf[:20]
```

```
features shape:  756453
```

Out[38]:

```
['aa',
 'aa and',
 'aa cell',
 'aa coffe',
 'aa doubl',
 'aa grade',
 'aa offer',
 'aa pod',
 'aa superior',
 'aa this',
 'aa though',
 'aa to',
 'aaa',
 'aaa perfect',
 'aaaaaaaaagghh',
 'aaaaah',
 'aaaaah satisfi',
 'aaaaah they',
 'aaaaahhhhhhhhhhhhhhh',
 'aaaaahhhhhhhhhhhhhhh the']
```

Using the vocablary generated by train data and generating the TF-IDF vector representation or sparce metric representation for training data set

In [39]:

```
final_count_tf_idf_train = tf_idf_vect_train.transform(x_train.values)
final_count_tf_idf_train.shape
```

Out[39]:

```
(70000, 756453)
```

- **TF-IDF vector representation for TEST DATA needs to be generated using VOCABLARY OF TRAIN DATA since test data is always UNKNOWN in real testing phase.**
- **And vocablary of test data having new words should not be considered while generating vector representation for test data, and vocablary or unique words from test data which are already present in trainign data set will be considered while generating vector representation for test data**

In [40]:

```
final_counts_tf_idf_test = tf_idf_vect_train.transform(x_test.values)
final_counts_tf_idf_test.shape
```

```
final_counts_tf_idf_test.shape
```

Out[40]:

```
(30000, 756453)
```

1. Logistic Regression with GridSearchCV and L2 Regularization for TF-IDF

In [41]:

```
LogisticRegression5fold_and_MSE(final_count_tf_idf_train, y_train, final_counts_tf_idf_test, y_
test, search_type='GridSearchCV', regulariser='l2')
```

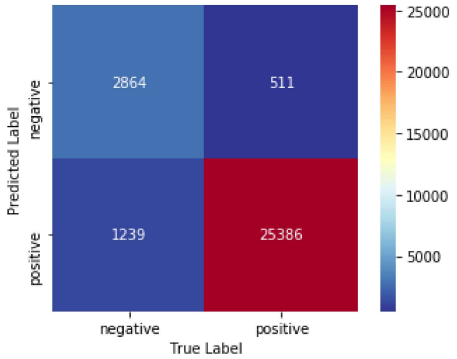
```
This is 5 fold  GridSearchCV  with  l2  regulariser Logistic Regression
LogisticRegression(C=10000, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
0.9413333333333334
```

In [42]:

```
LogisticRegression_with_optimal_C(final_count_tf_idf_train, y_train, final_counts_tf_idf_test,
y_test, 100000 , regulariser='l2')
```

```
This is Logistic Regression with  l2  regulariser for optimal C= 100000
Able to achieve  94.17 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.85      0.70      0.77      4103
          1       0.95      0.98      0.97     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  69.80258347550573
False Negative Rate:  1.9732015291346487
False Positive Rate:  30.197416524494276
True Positive Rate:  98.02679847086534
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l2     |       100000      |    5.83    |
+---------------------+-------------+-------------------+------------+
```

Confusion Metrix



OBSERVATIONS: Logistic Regression with GridSearchCV and L2 Regularization for TF-IDF

- 1) We have got the best result for C=100000 in the above cross validationThe optimal C is 1 00000
- 2) **The test accuracy or accuracy over unknown data for optimal C = 100000 is 94.16%**

2. Logistic Regression with GridSearchCV and L1 Regularization for TF-IDF

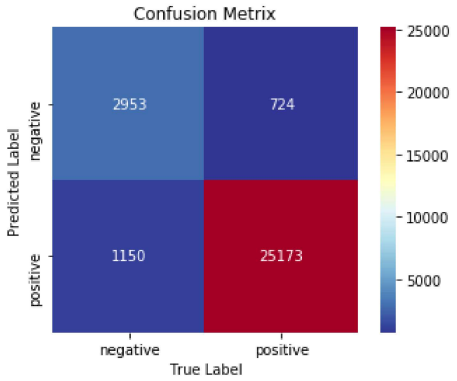
In [43]:
```
LogisticRegression5fold_and_MSE(final_count_tf_idf_train, y_train, final_counts_tf_idf_test, y_
test, search_type='GridSearchCV', regulariser='l1')
```

```
This is 5 fold  GridSearchCV  with  l1  regulariser Logistic Regression
LogisticRegression(C=10, 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)
0.9375333333333333
```

In [44]:
```
LogisticRegression_with_optimal_C(final_count_tf_idf_train, y_train, final_counts_tf_idf_test,
y_test, 10 , regulariser='l1')
```

```
This is Logistic Regression with  l1  regulariser for optimal C= 10
Able to achieve  93.75 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.80      0.72      0.76      4103
```

```
          1       0.96      0.97      0.96     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  71.97172800389959
False Negative Rate:  2.795690620535197
False Positive Rate:  28.02827199610041
True Positive Rate:  97.2043093794648
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l1     |         10        |    6.25    |
+---------------------+-------------+-------------------+------------+
```



OBSERVATIONS: Logistic Regression with GridSearchCV and L1 Regularization for TF-IDF

- 1) We have got the best result for C=10 in the above cross validationThe optimal C is 10
- 2) **The test accuracy or accuracy over unknown data for optimal C = 10 is 93.75%**

3. Logistic Regression with RandomizedSearchCV and L2 Regularization for TF-IDF

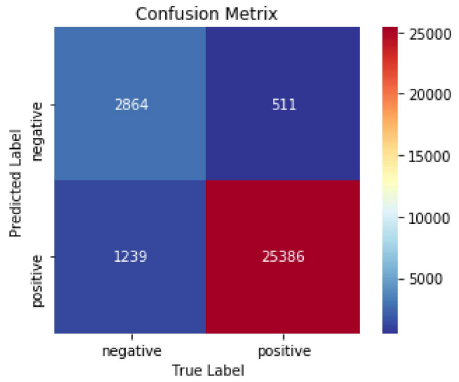
In [45]:
```
LogisticRegression5fold_and_MSE(final_count_tf_idf_train, y_train, final_counts_tf_idf_test, y_
test, search_type='RandomizedSearchCV', regulariser='l2')
```

```
This is 5 fold  RandomizedSearchCV  with  l2  regulariser Logistic Regression
LogisticRegression(C=10000, 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
```

```
            penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
            verbose=0, warm_start=False)
0.9413333333333334
```

In [46]:
```
LogisticRegression_with_optimal_C(final_count_tf_idf_train, y_train, final_counts_tf_idf_test,
y_test, 100000 , regulariser='l2')
```

```
This is Logistic Regression with  l2  regulariser for optimal C= 100000
Able to achieve  94.17 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.85      0.70      0.77      4103
          1       0.95      0.98      0.97     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  69.80258347550573
False Negative Rate:  1.9732015291346487
False Positive Rate:  30.197416524494276
True Positive Rate:  98.02679847086534
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l2     |       100000      |    5.83    |
+---------------------+-------------+-------------------+------------+
```



OBSERVATIONS: Logistic Regression with RandomizedSearchCV and L2 Regularization for TF-IDF

1) We have got the best result for C=100000 in the above cross validationThe optimal C is 100000

- 1) We have got the best result for C=100000 in the above cross validation The optimal C is 100000
- 2) **The test accuracy or accuracy over unknown data for optimal C = 100000 is 94.16%**

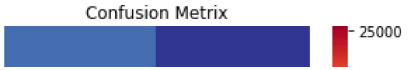
4. Logistic Regression with RandomizedSearchCV and L1 Regularization for TF-IDF

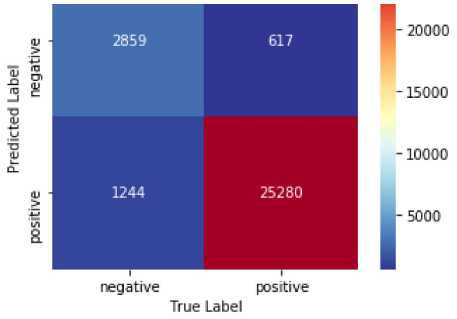
In [47]:
```
LogisticRegression5fold_and_MSE(final_count_tf_idf_train, y_train, final_counts_tf_idf_test, y_
test, search_type='RandomizedSearchCV', regulariser='l1')
```

```
This is 5 fold  RandomizedSearchCV  with  l1  regulariser Logistic Regression
LogisticRegression(C=10, 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)
0.9375333333333333
```

In [48]:
```
LogisticRegression_with_optimal_C(final_count_tf_idf_train, y_train, final_counts_tf_idf_test,
y_test, 10000 , regulariser='l1')
```

```
This is Logistic Regression with  l1  regulariser for optimal C= 10000
Able to achieve  93.8 % of test accuracy
Classification Report:
             precision    recall  f1-score   support

          0       0.82      0.70      0.75      4103
          1       0.95      0.98      0.96     25897

avg / total       0.94      0.94      0.94     30000

True Negative Rate:  69.68072142334877
False Negative Rate:  2.382515349268255
False Positive Rate:  30.319278576651232
True Positive Rate:  97.61748465073174
+---------------------+-------------+-------------------+------------+
|        Model        | Regularizer | Hyper Parameter C | Test Error |
+---------------------+-------------+-------------------+------------+
| Logistic Regression |      l1     |       10000       |    6.2     |
+---------------------+-------------+-------------------+------------+
```





OBSERVATIONS: Logistic Regression with RandomizedSearchCV and L1 Regularization for TF-IDF

- 1) We have got the best result for C=10000 in the above cross validationThe optimal C is 10000
- 2) **The test accuracy or accuracy over unknown data for optimal C = 10000 is 93.77%**

#

Observation:

- **Its required to do upsampling or downsampling to get the very acceptable model or technically the acceptable confusion metrix**

L1 REGULARIZER LR using TF-IDF for increasing values of lambda

Taking increasing values of lambda

In [49]:
```
C = [100, 10, 1, 0.1 , 0.01, 0.001]
final_result = []

for c in C:
    result_temp = []
    clf = LogisticRegression(C=c, penalty='l1');
    clf.fit(final_count_tf_idf_train, y_train);
    pred = clf.predict(final_counts_tf_idf_test)
    acc = accuracy_score(y_test, pred) * 100
    w = clf.coef_
```

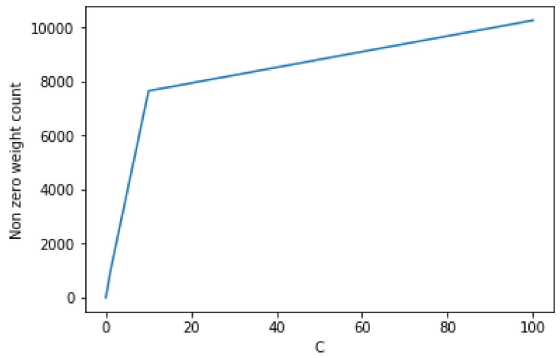
```
    non_zero_count = np.count_nonzero(w)
    result_temp.append(c)
    result_temp.append(acc)
    result_temp.append(non_zero_count)
    final_result.append(result_temp)

arr_final_result = np.asarray(final_result)
arr_final_result
```

Out[49]:

```
array([[1.00000000e+02, 9.36066667e+01, 1.02630000e+04],
       [1.00000000e+01, 9.37533333e+01, 7.65200000e+03],
       [1.00000000e+00, 9.32433333e+01, 9.35000000e+02],
       [1.00000000e-01, 8.83033333e+01, 8.40000000e+01],
       [1.00000000e-02, 8.63233333e+01, 0.00000000e+00],
       [1.00000000e-03, 8.63233333e+01, 0.00000000e+00]])
```

In [50]:

```
plt.plot(arr_final_result[:,0], arr_final_result[:,2])
plt.xlabel("C")
plt.ylabel("Non zero weight count")
plt.show()
```



OBSERVATIONS: As C increases (lambda decreases), count of non zero weights increases. In other words, as lambda increases, sparsity increases since low number of non zero weight count.

#

FEATURE IMPORTANCE BEFORE ADDING NOISE TO CHECK FOR

FEATURE IMPORTANCE BEFORE ADDING NOISE TO CHECK FOR MULTICOLLINIARITY

In [51]:
```
clf = LogisticRegression(C=10, penalty='l1');
clf.fit(final_counts_bow_train, y_train);
```

In [52]:
```
w = clf.coef_
print(np.count_nonzero(w))
print(w.shape)
weights = np.ravel(w)
print(weights.shape)
np.array(features_count_vect_train_bow).shape
```

```
9344
(1, 756453)
(756453,)
```

Out[52]:
```
(756453,)
```

In [53]:
```
def showProminentWords(feature_names, feature_weights, top_prominent):
    import pandas as pd

    df=pd.DataFrame({
    "PROMINENT_FEATURES": feature_names,
    "WEIGHTS": feature_weights})
    df=df.sort_values('WEIGHTS',ascending=False)
    print("TOP ",top_prominent, " IMPORTANT FEATURES for POSITIVE class")
    print("_____________________________________________________")
    print(df.iloc[:top_prominent][["PROMINENT_FEATURES","WEIGHTS"]])

    df=df.sort_values('WEIGHTS',ascending=True)
    print("\n\nTOP ",top_prominent, " IMPORTANT FEATURES for NEGATIVE class")
    print("_____________________________________________________")
    print(df.iloc[:top_prominent][["PROMINENT_FEATURES","WEIGHTS"]])

top_prominent = 20
feature_weights = clf.coef_
showProminentWords(features_count_vect_train_bow, weights, top_prominent)
```

```
TOP  20  IMPORTANT FEATURES for POSITIVE class
```

```
-----------------------------------------------------
       PROMINENT_FEATURES    WEIGHTS
434318     not disappoint  10.272739
43481          arriv dent   8.152318
602449           star off   7.598595
596608        spectacular   6.965911
99736            buy here   6.534884
347989          just what   6.467601
342644         jar includ   6.071453
624226          syrup was   6.069489
168422        definit buy   6.045118
324205          indic the   5.802309
247834          four star   5.709924
508415         price isnt   5.587656
61945       be disappoint   5.469895
505270        prefer this   5.367360
588221               solv   5.335423
435732          not stale   5.304224
179955      disappoint is   5.299251
577800            skeptic   5.248685
435227        not overpow   5.149364
611736        strong your   5.042532


TOP  20  IMPORTANT FEATURES for NEGATIVE class
-----------------------------------------------------
       PROMINENT_FEATURES   WEIGHTS
411070        morn realli -9.461187
431635          no wonder -8.695903
477907     packet instead -8.602708
75666       bigger packag -8.335684
200551         either for -8.267680
107773        candi still -8.130486
22305          amazon say -8.129818
707680    veri dissapoint -7.897101
691766           two star -7.895495
436099          not worth -7.880821
707494        veri bitter -7.834433
307632             horrid -7.764544
27468           and flour -7.426802
744128              worst -7.310567
411530          most bean -7.221011
187186     dont recommend -7.084593
175          abit sweeter -7.042487
679943         top better -7.032700
742251       wont purchas -7.028054
```

```
678804          too happi -6.813376
```

#

ADDING NOISE TO CHECK FOR MULTICOLLINIARITY

Defining noise - Random number will be generated from defined normal distribution

In [54]:
```
#Normal distribution with Mean 0, Std Dev 0.1
noise = np.random.normal(0, 0.1 )
noise
```

Out[54]:
```
-0.19745074015213385
```

Adding noise

In [55]:
```
final_counts_bow_train.dtype='float64'
```

In [56]:
```
final_counts_bow_train[final_counts_bow_train!=0] += noise
```

FEATURE IMPORTANCE AFTER ADDING NOISE TO CHECK FOR MULTICOLLINIARITY

In [57]:
```
clf = LogisticRegression(C=10, penalty='l1');
clf.fit(final_counts_bow_train, y_train);
```

In [58]:
```
w = clf.coef_
print(np.count_nonzero(w))
print(w.shape)
weights = np.ravel(w)
print(weights.shape)
```

```
np.array(features_count_vect_train_bow).shape
```

```
8088
(1, 756453)
(756453,)
```

Out[58]:

```
(756453,)
```

In [59]:

```
def showProminentWords(feature_names, feature_weights, top_prominent):
    import pandas as pd

    df=pd.DataFrame({
    "PROMINENT_FEATURES": feature_names,
    "WEIGHTS": feature_weights})
    df=df.sort_values('WEIGHTS',ascending=False)
    print("TOP ",top_prominent, " IMPORTANT FEATURES for POSITIVE class")
    print("_____________________________________________________")
    print(df.iloc[:top_prominent][["PROMINENT_FEATURES","WEIGHTS"]])

    df=df.sort_values('WEIGHTS',ascending=True)
    print("\n\nTOP ",top_prominent, " IMPORTANT FEATURES for NEGATIVE class")
    print("_____________________________________________________")
    print(df.iloc[:top_prominent][["PROMINENT_FEATURES","WEIGHTS"]])

top_prominent = 20
feature_weights = clf.coef_
showProminentWords(features_count_vect_train_bow, weights, top_prominent)
```

```
TOP  20  IMPORTANT FEATURES for POSITIVE class
-----------------------------------------------------
       PROMINENT_FEATURES    WEIGHTS
691766           two star  28.813072
200551         either for  26.743280
436099          not worth  26.302948
411070        morn realli  26.215628
431635          no wonder  24.491796
477907     packet instead  24.415337
107773        candi still  24.204372
27468           and flour  23.193149
653485        them slight  22.859961
207873          especi my  22.385239
707680    veri dissapoint  22.252091
744128              worst  22.141785
```

```
707494        veri bitter  22.129996
22305          amazon say  21.656190
637598         than twice  21.320399
679943         top better  20.985428
452131         old famili  20.907866
707496         veri bland  20.814828
463740           orang as  20.697328
187186     dont recommend  20.648542


TOP  20  IMPORTANT FEATURES for NEGATIVE class
--------------------------------------------------
       PROMINENT_FEATURES    WEIGHTS
43481          arriv dent -30.639513
434318     not disappoint -27.489736
596608        spectacular -25.167354
347989          just what -22.120163
630307          taught to -21.501125
624226          syrup was -21.478816
635159            test it -20.782779
168422        definit buy -18.573476
505270        prefer this -18.443223
420131           my small -18.407660
577800            skeptic -18.405562
588221               solv -17.159429
61945       be disappoint -16.749680
435227        not overpow -16.508367
734487       will purchas -16.305270
111783           carri it -16.186998
247834          four star -16.100397
10902          again they -15.705524
338985             it sad -15.533753
105668         can amazon -15.329034
```

OBSERVATION - MULTICOLLINIARITY CHECK

- **I have manually compared the top 20 features FOR ALMOST 4 RUNS OF THE NOTEBOOK for both positive and negative features BEFORE and AFTER noise.**
- **Since we are RANDOMLY generating the noise. And after adding noise; for across notebook runs I got different results. As per my observation, most of the times, few of the prominent features or nonr of the prominent features were matching before and after adding noise.**
- **Hence, there is almost MULTICOLLINEARITY ISSUE since its my guess that feature are dependent on each other by GRAMATICAL ORDER OF WORDS and somehow by the noice this ORDER gets changed AND hence prominent features to the good extent**

Did you find this Kernel useful?
Show your appreciation with an upvote

0

Comments (0)

Sort by Select...

Click here to enter a comment...