Information about data:

```
                    ->We have the amazon reviews dataset from kaggle

                    ->Reviews are given for the product

                    ->The features of the data were:

                          Id

                          ProductId- unique identifier for the product

                          UserId- unqiue identifier for the user

                          ProfileName

                          HelpfullnessNumerator- number of users who fou
nd the review helpful

                          HelpfulnessDenominator- number of users who in
dicated whether they found the review
                                                helpful or not

                          Score-rating between 1 and 5

                          Time-timestamp for the review

                          Summary- brief summary of the review

                          Text- text of the review

                    -> Based on the score of the review we review we cla
ssify them into positive and negative

                    Number of reviews: 568,454
```

Objective:

```
-> Cleaning the dataset by classifying them into positive and negati
ve reviews based on the
    rating provided and removing the duplicates

-> Converting the text data to vectors by using Bag of
words,Tfidf,word2vec,Average word2vec

-> Applying logistic regression to determine the best lambda
```

```
-> Using grid search and random search to determine the best lambda

-> Using both L1 and L2 regularizations and checking the sparsity wi
th different values
```

Importing the required libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
from sklearn.feature_extraction.text import
TfidfTransformer,TfidfVectorizer,CountVectorizer
import sqlite3
import nltk
import string
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn import cross_validation
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/cross_validation.py:41: DeprecationWarning: This module wa
s deprecated in version 0.18 in favor of the model_selection module into wh
ich all the refactored classes and functions are moved. Also note that the
interface of the new CV iterators are different from that of this module. T
his module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

->loading the data and information about the data

-> Shape of the data

-> Dimensionality of the data

-> Attributes if the data

-> Sample of the data

In [2]:

```
con = sqlite3.connect("database.sqlite")
data = pd.read_sql_query("SELECT * FROM Reviews WHERE Score != 3",con)
print(data.shape)
print(data.ndim)
print(data.columns)
print(data.head(5))
```

```
(525814, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
```

```
1   2  B00813GRG4  A1D87F6ZCVE5NK                          dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                            Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

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

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...

1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...

3         Cough Medicine  If you are looking for the secret ingredient i...

4            Great taffy  Great taffy at a great price.  There was a wid...
```

DATA PRE-PROCESSING:

```
-> Classifying the reviews into positive and negative based on the s
core for the review

-> Considering reviews with score 3 as neutral and 1,2 as negative a
nd 4,5 as positive
```

In [3]:

```
def change(n):
    if n>3:
        return 'positive'
    return 'negative'

rating = data['Score']

rating = rating.map(change)

data['Score'] = rating

data.head(6)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0
5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0

DATA PRE-PROCESSING:

    Removing Duplicates

In [4]:

```
user = pd.read_sql_query("""SELECT * FROM Reviews WHERE UserId= "AR5J8UI46C
URR" ORDER BY ProductId """,con)
print(user)
```

```
       Id   ProductId          UserId       ProfileName  HelpfulnessNumerator
\
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2

1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2

2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2

3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
```

```
3   75791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                      2

4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                      2


   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

Observation:

    -> Here we can see that for the same time span we got five reviews,
    practically which is not possible

    ->This happened because when the user given review for a product it
    is applied to all the flavors of the product

In [5]:

```
sorteddata = data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='quicksort',na_position='last')
```

In [6]:

```
finaldata = sorteddata.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='first',inplace=False)
```

Information about the modified data:

    -> Shape of the data

    -> Dimensionality of the data

    -> Attributes if the data

    -> Sample of modified data

In [7]:

```
print(finaldata.shape)
print(finaldata.ndim)
print(finaldata.columns)
print(finaldata.head(5))
```

```
(364173, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
            Id   ProductId          UserId                ProfileName  \
138706  150524  0006641040   ACITT7DI6IDDL            shari zychinski
138688  150506  0006641040  A2IW4PEEKO2R0U                      Tracy
138689  150507  0006641040  A1S4A3IQ2MU7V4        sally sue "sally sue"
138690  150508  0006641040    AZGXZ2UUK6X  Catherine Hallberg "(Kate)"
138691  150509  0006641040  A3CMRKGE0P909G                     Teresa

        HelpfulnessNumerator  HelpfulnessDenominator     Score        Time
\
138706                     0                       0  positive   939340800
138688                     1                       1  positive  1194739200
138689                     1                       1  positive  1191456000
138690                     1                       1  positive  1076025600
138691                     3                       4  positive  1018396800

                                          Summary  \
138706                  EVERY book is educational
138688  Love the book, miss the hard cover version
138689               chicken soup with rice months
138690       a good swingy rhythm for reading aloud
138691                 A great way to learn the months

                                                     Text
138706  this witty little book makes my son laugh at l...
138688  I grew up reading these Sendak books, and watc...
138689  This is a fun way for children to learn their ...
138690  This is a great little book to read aloud- it ...
138691  This is a book of poetry about the months of t...
```

BAG OF WORDS:

```
-> TIME BASED SPLITTING:

    -> Converting the text data to vectors

    -> Using L2 regularization with random search and grid search to
determine the "C"

    -> Using L1 regularization with random search and grid search to
determine the  "C"

    -> Checking the sparsity with different values using L1 regulari
zation
```

In [8]:

```
sorted_count_vect = finaldata.sort_values("Time",axis=0,ascending=True,kind
='quicksort',na_position='last',inplace=False)
```

Sample of the time based sliced data

In [9]:

```
sorted_count_vect.head(5)
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0

In [10]:

```
sorted_count_vect.shape
```

Out[10]:

```
(364173, 10)
```

In [11]:

```
cv = CountVectorizer()
cv_data = cv.fit_transform(sorted_count_vect["Text"].values)
```

In [12]:

```
print(type(cv_data))
print(cv_data.shape)
```

```
print(cv_data.ndim)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(364173, 115282)
2
```

Bag Of Words:L2 REGULARIZATION WITH GRID SEARCH AND RANDOM SEARCH

In [13]:

```
from sklearn.linear_model import LogisticRegression
```

In [14]:

```
log_reg = LogisticRegression(penalty='l2')
```

In [15]:

```
xtrain = cv_data[0:250000]
xtest = cv_data[250000:]
ytrain = sorted_count_vect['Score'][0:250000]
ytest = sorted_count_vect['Score'][250000:]
```

In [16]:

```
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(250000, 115282)
(114173, 115282)
(250000,)
(114173,)
```

Bag Of Words:GRID SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [17]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [18]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5)
```

In [19]:

```
model.fit(xtrain,ytrain)
```

Out[19]:

```
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, f
it_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': [0.0001, 0.01, 1, 100, 10000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
```

```
        pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
        scoring='accuracy', verbose=0)
```

In [20]:

```
model.best_estimator_
```

Out[20]:

```
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)
```

In [21]:

```
model.score(xtest,ytest)
```

Out[21]:

```
0.9242903313392834
```

In [22]:

```
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
```

```
0.9242903313392834
```

In [23]:

```
np.count_nonzero(v)
```

Out[23]:

```
94292
```

In [24]:

```
clf2 = LogisticRegression(C= 0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
v = clf2.coef_
np.count_nonzero(v)
```

Out[24]:

```
94292
```

In [25]:

```
clf2 = LogisticRegression(C= 0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
v = clf2.coef_
np.count_nonzero(v)
```

Out[25]:

```
94292
```

Observation:

```
BAG OF WORDS:
```

```
-> L2 Regularization with Grid Search Results:

    -> The grid search resulted with best "C" value as 1

    -> There is no change in the sparsity level when the values
of "C" = [1,0.1,0.01]
```

Bag Of Words:RANDOM SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [26]:

```
import scipy
```

In [27]:

```
scipy.stats.randint.rvs(0,10,size=5)
```

Out[27]:

```
array([0, 8, 8, 9, 0])
```

In [31]:

```
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
log_reg_l2 = LogisticRegression(penalty='l2')
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scorin
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
{'C': array([4, 1, 3, 2, 1])}
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)
```

In [32]:

```
print(model.best_estimator_)
print(model.score(xtest,ytest))
```

```
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)
0.9242903313392834
```

In [33]:

```
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

```
0.9242903313392834
```

Out[33]:

94292

In [34]:

```
clf2 = LogisticRegression(C=0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.9240450894694893

Out[34]:

94292

In [35]:

```
clf2 = LogisticRegression(C=0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.9144981738239338

Out[35]:

94292

Observation:

```
BAG OF WORDS:

    -> By using L2 Regularization with Random Search the best 'C' va
lue = 1

    -> Experimented different 'C' values to check the sparsity

    -> By using L2 Regularization for different values 'C' values th
ere is no improvement in the sparsity

    -> All the 'C' values resulted in the same sparsity

    -> By increasing 'C' value we can observe slightly decrement in
accuracy score
```

Bag Of Words:L1 REGULARIZATION WITH GRID SEARCH AND RANDOM SEARCH

In [36]:

```
from sklearn.linear_model import LogisticRegression
```

In [37]:

```
log_reg = LogisticRegression(penalty='l1')
```

In [38]:

```
xtrain = cv_data[0:250000]
xtest = cv_data[250000:]
ytrain = sorted_count_vect['Score'][0:250000]
ytest = sorted_count_vect['Score'][250000:]
```

In [39]:

```
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(250000, 115282)
(114173, 115282)
(250000,)
(114173,)
```

Bag Of Words:GRID SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [40]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [41]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
print(model.score(xtest,ytest))
print(model.best_estimator_)
```

```
LogisticRegression(C=1, 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)
0.9232305361162446
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [42]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.9232743293072793
10764
```

In [43]:

```
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
```

```
print(np.count_nonzero(v))
```

```
0.9206730137598206
2208
```

In [44]:

```
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8979531062510401
444
```

Observation:

```
BAG OF WORDS:

    -> L1 Regularization with Grid Search Results:

        -> The grid search resulted with best "C" value as 1

        -> When the value of "C" = 1 it resulted in a sparsity of 10
764

        -> When the value of "C" = 0.1 it resulted in a sparsity of
2208

        -> When the value of "C" = 0.01 it resulted in a sparsity of
444
```

Bag Of Words:RANDOM SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [45]:

```
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
log_reg_l2 = LogisticRegression(penalty='l1')
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scorin
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
{'C': array([3, 3, 4, 1, 4])}
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [46]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
```

```
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.9232392947544515
10773
```

In [47]:

```
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.9206905310362344
2205
```

In [48]:

```
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.8979531062510401
444
```

Observation:

```
BAG OF WORDS:

    -> Implemented logistic regression with L1 Regularization on Bag
Of Words with Random Search

    -> Both the methods have returned the best "C" value = 1

    -> Experimented different 'C' values to check the sparsity

    -> By using L1 Regularization we can observe the change in spars
ity level while the value of "C" changes

    -> When "C" = 1 it resulted with a sparsity of 10773

    -> When "C" = 0.1 it resulted with a sparsity of 2205

    -> When "C" = 0.01 it resulted with a sparsity of 444
```

TFIDF:

```
-> TIME BASED SPLITTING:

    -> Converting the text data to vectors
```

```
    -> Using L2 regularization with random search and grid search to
determine the "C"

    -> Using L1 regularization with random search and grid search to
determine the "C"

    -> Checking the sparsity with different values using L1 regulari
zation
```

In [11]:

```
tfidf_data = finaldata.sort_values("Time",axis=0,ascending=True,kind='quick
sort',na_position='last',inplace=False)
```

In [12]:

```
tfidf_data.shape
```

Out[12]:

```
(364173, 10)
```

In [13]:

```
tfidf_vect = TfidfVectorizer(ngram_range=(1,2))
```

In [14]:

```
tfidf_vect_data = tfidf_vect.fit_transform(tfidf_data['Text'].values)
```

In [15]:

```
tfidf_vect_data.shape
```

Out[15]:

```
(364173, 2910206)
```

In [16]:

```
xtrain = tfidf_vect_data[0:250000]
xtest = tfidf_vect_data[250000:]
ytrain = tfidf_data["Score"][0:250000]
ytest = tfidf_data['Score'][250000:]
```

In [17]:

```
print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(250000, 2910206)
(250000,)
(114173, 2910206)
(114173,)
```

TFIDF:L2 REGULARIZATION WITH GRID SEARCH AND RANDOM SEARCH

In [57]:

```
from sklearn.linear_model import LogisticRegression
```

In [58]:

```
log_reg = LogisticRegression(penalty='l2')
```

In [59]:

```
print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(250000, 2910206)
(250000,)
(114173, 2910206)
(114173,)
```

TFIDF:GRID SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [60]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [61]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
model.score(xtest,ytest)
```

```
LogisticRegression(C=100, 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)
```

Out[61]:

```
0.9466598933197866
```

In [62]:

```
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

```
0.933425590989113
```

Out[62]:

```
2270091
```

In [64]:

```
clf2 = LogisticRegression(C=0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
```

```
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.881180314084766

Out[64]:

2270091

In [65]:

```
clf2 = LogisticRegression(C=0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.8257468928730961

Out[65]:

2270091

Observation:

```
TFIDF:

    -> L2 Regularization with Grid Search Results:

        -> The grid search resulted with best "C" value as 100

        -> There is no change in the sparsity level when the values
of "C" = [1,0.1,0.01]
```

TFIDF:RANDOM SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [66]:

```
import scipy
```

In [67]:

```
scipy.stats.randint.rvs(1,10,size=5)
```

Out[67]:

array([7, 3, 6, 2, 3])

In [76]:

```
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
```

{'C': array([1, 3, 3, 3, 1])}

In [77]:

```
log_reg_l2 = LogisticRegression(penalty='l2')
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scoring='accuracy',cv=5,n_iter=5)
```

```
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [78]:

```
print(model.best_estimator_)
print(model.score(xtest,ytest))
```

```
LogisticRegression(C=3, 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)
0.9418514009441812
```

In [79]:

```
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.933425590989113
2270091
```

In [80]:

```
clf2 = LogisticRegression(C=0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

```
0.881180314084766
```

Out[80]:

```
2270091
```

In [81]:

```
clf2 = LogisticRegression(C=0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.8257468928730961
2270091
```

Observation:

```
TFIDF
```

```
    -> By using L2 Regularization with  Random Search the best 'C' v
alue = 3

    -> Experimented different 'C' values to check the sparsity

    -> By using L2 Regularization for different values 'C' values th
ere is no improvement in the sparsity

    -> All the 'C' values resulted in the same sparsity
```

TFIDF:GRID SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [82]:

```
from sklearn.linear_model import LogisticRegression
```

In [83]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [84]:

```
log_reg = LogisticRegression(penalty='l1')
```

In [92]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
print(model.score(xtest,ytest))
```

```
LogisticRegression(C=10000, 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)
0.9421842291960446
```

In [97]:

```
clf2 = LogisticRegression(C=10000,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.9423418846837693
127776
```

In [86]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.9397142932216899
```

```
2985
```

In [87]:

```python
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.9009923537088453
370
```

In [88]:

```python
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.827673793278621
17
```

Observation:

```
     TFIDF:

           -> L1 Regularization with Grid Search Results:

           -> The grid search resulted with best "C" value as 10000

           -> When the value of "C" = 10000 it resulted in a sparsity o
   f 127776

           -> When the value of "C" = 1 it resulted in a sparsity of 29
   85

           -> When the value of "C" = 0.1 it resulted in a sparsity of
   370

           -> When the value of "C" = 0.01 it resulted in a sparsity of
   17
```

TFIDF:RANDOM SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [90]:

```python
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
```

```
{'C': array([3, 1, 3, 3, 2])}
```

In [91]:

```python
log_reg_l2 = LogisticRegression(penalty='l1')
```

```
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scorin
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [96]:

```
clf2 = LogisticRegression(C=3,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.9460117540924737
10130
```

In [93]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.9397055345834829
2987
```

In [94]:

```
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.9009923537088453
370
```

In [95]:

```
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.827673793278621
17
```

Observation:

```
TFIDF:
```

```
    -> Implemented logistic regression with L1 regularization on TFI
DF with Random Search

    -> Both the methods have returned the best "C" value = 3

    -> Experimented different 'C' values to check the sparsity

    -> By using L1 Regularization we can observe the change in spars
ity level while the value of "C" changes

    -> When "C" = 3 it resulted with a sparsity of 10130

    -> When "C" = 1 it resulted with a sparsity of 2987

    -> When "C" = 0.1 it resulted with a sparsity of 370

    -> When "C" = 0.01 it resulted with a sparsity of 17
```

WORD2VEC:

```
Constructing the word2vec representation of each word in the corpus
```

In [18]:

```
import gensim
from gensim.models import word2vec
```

In [19]:

```
import re
def cleanhtml(sentence):
    clean = re.compile("<.*?>")
    cleantext = re.sub(clean," ",sentence)
    return cleantext
def cleanpunct(sentence):
    cleanr = re.sub(r"[?|!|\|'|#|.|,|)|(|/]",r' ',sentence)
    return cleanr
```

In [20]:

```
sorted_w2vec = finaldata.sort_values("Time",axis=0,ascending=True,kind='qui
cksort',na_position='last',inplace=False)
```

In [21]:

```
print(sorted_w2vec.shape)
print(sorted_w2vec.ndim)
print(sorted_w2vec.columns)
print(sorted_w2vec.head(5))
```

```
(364173, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
```

```
      dtype='object')
             Id   ProductId          UserId              ProfileName  \
138706  150524  0006641040   ACITT7DI6IDDL          shari zychinski
138683  150501  0006641040   AJ46FKXOVC7NR       Nicholas A Mesiano
417839  451856  B00004CXX9   AIUWLEQ1ADEG5         Elizabeth Medina
346055  374359  B00004CI84  A344SMIA5JECGM          Vincent P. Ross
417838  451855  B00004CXX9   AJH6LUC1UT1ON  The Phantom of the Opera

        HelpfulnessNumerator  HelpfulnessDenominator     Score       Time  `
138706                     0                       0  positive  939340800
138683                     2                       2  positive  940809600
417839                     0                       0  positive  944092800
346055                     1                       2  positive  944438400
417838                     0                       0  positive  946857600

                                                  Summary  \
138706                        EVERY book is educational
138683  This whole series is great way to spend time w...
417839                              Entertainingl Funny!
346055                           A modern day fairy tale
417838                                         FANTASTIC!

                                                     Text
138706  this witty little book makes my son laugh at l...
138683  I can remember seeing the show when it aired o...
417839  Beetlejuice is a well written movie ..... ever...
346055  A twist of rumplestiskin captured on film, sta...
417838  Beetlejuice is an excellent and funny movie. K...
```

In [22]:

```
i=0
sentences_list=[]
for sent in sorted_w2vec['Text'].values:
    filtered_sentences = []
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleanedwords in cleanpunct(w).split():
            if(cleanedwords.isalpha()):
                filtered_sentences.append(cleanedwords.lower())
    sentences_list.append(filtered_sentences)
```

In [23]:

```
print(len(sentences_list))
print(type(sentences_list))
```

```
364173
<class 'list'>
```

In [24]:

```
w2vmodel =
gensim.models.Word2Vec(sentences_list,min_count=4,size=100,workers=4)
```

-> Most similar word

-> Similarity between the words

-> Dimensionality representation of a word

In [25]:

```
print(w2vmodel.most_similar("where"))
print(w2vmodel.similarity("where",'who'))
print(w2vmodel.wv['hello'])
```

```
[('when', 0.555910050868988), ('wherever', 0.5385503768920898), ('somewhere
', 0.5213028192520142), ('everywhere', 0.517043948173523), ('anywhere', 0.5
11245846748352), ('why', 0.49936985969543457), ('whenever',
0.4894091486930847), ('what', 0.4891246259212494), ('until',
0.4887976348400116), ('tx', 0.46112507581710815)]
0.24810759684835426
[-1.1313014   1.0932155   0.11378156  0.5984697  -0.40777805  0.36542675
 -0.10039888 -0.6144885   0.1795656   0.3624454  -0.22921501  0.19641885
 -0.2573651   0.9770184  -0.06685726 -0.09147077 -0.00734534 -0.04665074
 -0.26663142  0.64368945 -0.11834418  0.28151038  0.2889424  -0.21643315
 -0.1667287  -0.36938614  0.6879662  -0.10011698  0.94558644  0.5527474
  0.0572133   0.2621847  -0.18907733  0.66910386 -0.11617593  0.12504373
  0.25503594  0.1557429  -0.01187207 -0.54485494 -0.07472737  0.8321764
  0.18299055  0.9956394  -0.36131296 -0.05197132  0.72592556  0.25233057
  0.45833567 -0.28377217  0.6090648   0.20261562  0.07647496 -0.8056886
 -0.35348964 -0.06347021 -0.0497982  -0.77958965  0.3941121   0.14914612
 -0.21553643 -0.20195153  0.036475    0.6339579  -0.00773144 -0.27580798
 -0.31185317  0.99352306  0.28453943 -0.11844904  0.029054    0.27830666
 -0.2959344  -0.07851963  0.13070121 -0.17581376 -0.45458436 -0.36290792
 -0.7638931   0.627594   -0.19319664 -0.04386482 -0.31185874 -0.0699504
  0.03932377 -0.17857397  0.24290591  0.4074607  -0.02756741 -0.6117686
 -0.2612249   0.5490796   0.20474638  0.05279277  0.4699347   0.21727744
  0.74875426 -0.68860763  0.5961362   0.41836035]
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `m
ost_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() i
nstead).
  """Entry point for launching an IPython kernel.
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:2: DeprecationWarning: Call to deprecated `s
imilarity` (Method will be removed in 4.0.0, use self.wv.similarity() inste
ad).
```

Observation:

```
-> We have constructed the vector representation of each word

-> Using this model to construct vector representation of each sente
nce in average word2vec and tfidf-word2vec
```

AVERAGE WORD2VEC:

```
-> TIME BASED SPLITTING:

    -> Converting the text data to vectors with the help of word2vec
```

```
    -> Using L2 regularization with random search and grid search to
determine the "C"

    -> Using L1 regularization with random search and grid search to
determine the "C"

    -> Checking the sparsity with different values using L1 regulari
zation
```

In [107]:

```
sent_vectors = []
for sent in sentences_list:
    sent_vec = np.zeros(100)
    cnt=0
    for word in sent:
        try:
            vec = w2vmodel.wv[word]
            sent_vec += vec
            cnt += 1
        except:
            pass
    sent_vec /= cnt
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[88888]))
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:13: RuntimeWarning: invalid value
encountered in true_divide
  del sys.path[0]
```

```
364173
100
```

In [113]:

```
np.isnan(sent_vectors).any()
```

Out[113]:

```
False
```

In [110]:

```
type(sent_vectors)
```

Out[110]:

```
list
```

In [111]:

```
sent_vectors = np.nan_to_num(sent_vectors)
```

In [112]:

```
type(sent_vectors)
```

Out[112]:

```
numpy.ndarray
```

In [114]:

```
sent_vectors.shape
```

Out[114]:

```
(364173, 100)
```

AVERAGE WORD2VEC:L2 REGULARIZATION WITH GRID SEARCH AND RANDOM SEARCH

In [115]:

```
from sklearn.linear_model import LogisticRegression
```

In [116]:

```
log_reg = LogisticRegression(penalty='l2')
```

In [122]:

```
xtrain = sent_vectors[0:250000]
xtest = sent_vectors[250000:]
ytrain = sorted_w2vec['Score'][0:250000]
ytest = sorted_w2vec['Score'][250000:]
```

In [127]:

```
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(250000, 100)
(114173, 100)
(250000,)
(114173,)
```

AVERAGE WORD2VEC:GRID SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [126]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [145]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5,n_jobs=4)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
model.score(xtest,ytest)
```

```
LogisticRegression(C=10000, 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)
```

Out[145]:

Out[145]:

0.9422017464724585

In [128]:

```python
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.8896761931454897

Out[128]:

100

In [129]:

```python
clf2 = LogisticRegression(C=0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.8892995717025917

Out[129]:

100

In [130]:

```python
clf2 = LogisticRegression(C=0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
```

0.8884762597111401

Out[130]:

100

Observation:

```
AVERAGE WORD2VEC:

    -> L2 Regularization with Grid Search Results:

        -> Since we constructed the vector representation of each wo
rd in 100 dimensions

        -> Now each sentence is also in 100 dimensions

        -> The grid search resulted with best "C" value as 10000

        -> There is no change in the sparsity level when the values
of "C" = [1,0.1,0.01]
```

AVERAGE WORD2VEC:RANDOM SEARCH IMPLEMENTATION FOR L2 REGULARIZATION CLASSIFIER

In [151]:

```
import scipy
```

In [152]:

```
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
```

```
{'C': array([3, 4, 2, 1, 4])}
```

In [153]:

```
log_reg_l2 = LogisticRegression(penalty='l2')
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scorin
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
LogisticRegression(C=4, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [154]:

```
print(model.best_estimator_)
print(model.score(xtest,ytest))
```

```
LogisticRegression(C=4, 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)
0.9426221611063912
```

In [131]:

```
clf2 = LogisticRegression(C=1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.8896849517836967
100
```

In [132]:

```
clf2 = LogisticRegression(C=0.1,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.8892995717025917
```

```
100
```

In [133]:

```
clf2 = LogisticRegression(C=0.01,penalty='l2')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
np.count_nonzero(v)
print(np.count_nonzero(v))
```

```
0.8884762597111401
100
```

Observation:

AVERAGE WORD2VEC:

```
    -> By using L2 Regularization with  Random Search the best 'C' v
alue = 4

    -> Since we constructed the vector representation of each word i
n 100 dimensions

    -> Now each sentence is also in 100 dimensions

    -> Experimented different 'C' values to check the sparsity

    -> By using L2 Regularization for different values 'C' values th
ere is no improvement in the sparsity

    -> All the 'C' values resulted in the same sparsity
```

AVERAGE WORD2VEC:GRID SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [146]:

```
from sklearn.linear_model import LogisticRegression
```

In [147]:

```
parameters =  [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
```

In [148]:

```
log_reg = LogisticRegression(penalty='l1')
```

In [149]:

```
model = GridSearchCV(log_reg,parameters,scoring='accuracy',cv=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
print(model.score(xtest,ytest))
```

```
LogisticRegression(C=10000, 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)
0.9418338836677673
```

In [136]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8894747444667304
100
```

In [137]:

```
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8893258476172125
96
```

In [138]:

```
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8875478440612053
82
```

Observation:

```
AVERAGE WORD2VEC:

        -> L1 Regularization with Grid Search Results:

        -> Since we constructed the vector representation of each wo
rd in 100 dimensions

        -> Now each sentence is also in 100 dimensions

        -> The grid search resulted with best "C" value as 10000

        -> When the value of "C" = 10000 it resulted in a sparsity o
f 100

        -> When the value of "C" = 1 it resulted in a sparsity of 10
0
```

```
        -> When the value of "C" = 0.1 it resulted in a sparsity of
96

        -> When the value of "C" = 0.01 it resulted in a sparsity of
82
```

AVERAGE WORD2VEC:RANDOM SEARCH IMPLEMENTATION FOR L1 REGULARIZATION CLASSIFIER

In [155]:

```
param_grid = {'C': scipy.stats.randint.rvs(1,5,size=5)}
print(param_grid)
```

```
{'C': array([3, 4, 1, 2, 1])}
```

In [156]:

```
log_reg_l2 = LogisticRegression(penalty='l1')
model = RandomizedSearchCV(log_reg_l2,param_distributions=param_grid,scorin
g='accuracy',cv=5,n_iter=5)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
LogisticRegression(C=4, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [139]:

```
clf2 = LogisticRegression(C=1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8894747444667304
100
```

In [140]:

```
clf2 = LogisticRegression(C=0.1,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8893170889790055
98
```

In [141]:

```
clf2 = LogisticRegression(C=0.01,penalty='l1')
clf2.fit(xtrain,ytrain)
print(clf2.score(xtest,ytest))
v = clf2.coef_
print(np.count_nonzero(v))
```

```
0.8875478440612053
82
```

Observation:

```
AVERAGE WORD2VEC:

    -> Implemented logistic regression with L1 regularization on Ave
rage word2vec with Random Search

    -> Since we constructed the vector representation of each word i
n 100 dimensions

    -> Now each sentence is also in 100 dimensions

    ->  The best "C" value = 4

    -> Experimented different 'C' values to check the sparsity

    -> By using L1 Regularization we can observe the change in spars
ity level while the value of "C" changes

    -> When "C" = 1 it resulted with a sparsity of 100

    -> When "C" = 0.1 it resulted with a sparsity of 98

    -> When "C" = 0.01 it resulted with a sparsity of 82
```

Conclusion:

```
-> Grid search and Random search are applied to determine the best h
yper parameter

-> These cross validation techniques can be applied to any estimator

-> L2 regularization does not provide any sparsity with the differen
t hyper parameter tuning using both grid
search and Random search

-> L1 regularization provides sparsity and it increases its sparsity
if we tune the hyper parameter

-> The main reason for providing sparsity in L1 than L2 is due to co
nstant slope in L1

-> By using sparsity we can neglect the low weighted features

-> These type of models are very useful in the low latency applicati
ons for quick response
```