In [1]:

```
# ============================== loading libraries ===============================
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
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.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import classification_report
from prettytable import PrettyTable
import random
from scipy.stats import uniform
from sklearn.metrics import roc_curve, auc
from sklearn.learning_curve import validation_curve
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import precision_score, recall_score,roc_auc_score
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
import joblib

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

#import nltk
#nltk.download('stopwords')
```

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

#from gensim.models import KeyedVectors
#model = KeyedVectors.load_word2vec_format(‘GoogleNews-vectors-negative300.bin.g

#import gensim
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# ================================================================================
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the m
odel_selection module into which all the refactored classes and functions are m
oved. Also note that the interface of the new CV iterators are different from t
hat of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\learning_curve.py:22: Dep
recationWarning: This module was deprecated in version 0.18 in favor of the mod
el_selection module into which all the functions are moved. This module will be
removed in 0.20
  DeprecationWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.
py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module a
nd should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
C:\Users\AbhiShek\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
#Loading preprocessed data on 100K data sets and bifurcated according to TimeSta
fileObject = open("./train_to_file2.pkl",'rb') # we open the file for reading
X_train = pickle.load(fileObject) # load the object from the file

fileObject = open("./x_test_to_file2.pkl",'rb') # we open the file for reading
X_test = pickle.load(fileObject) # load the object from the file

fileObject = open("./y_train_to_file2.pkl",'rb') # we open the file for reading
y_train = pickle.load(fileObject) # load the object from the file

fileObject = open("./y_test_to_file2.pkl",'rb') # we open the file for reading
y_test = pickle.load(fileObject) # load the object from the file
```

In [4]:

```
#Appling BoW to fit and transform
count_vect =  CountVectorizer()
bow_nstd = count_vect.fit(X_train[:,9])
train_bow_nstd = count_vect.transform(X_train[:,9])
test_bow_nstd = count_vect.transform(X_test[:,9])

print("the type of count vectorizer ",type(train_bow_nstd))
print("the number of unique words ", test_bow_nstd.get_shape()[1])

print(train_bow_nstd.shape)
print(test_bow_nstd.shape)
print(y_test.shape)
print(y_train.shape)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the number of unique words  39675
(42000, 39675)
(18000, 39675)
(18000,)
(42000,)
```

In [13]:

```
# Colum Standardization of the BoW non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(train_bow_nstd)
train_bow = std_scal.transform(train_bow_nstd)
test_bow = std_scal.transform(test_bow_nstd)
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [5]:

```
print(count_vect.get_feature_names()[4250:4325])
```

```
['ballgames', 'ballistic', 'ballon', 'balloon', 'balloons', 'ballotin', 'ballot
ins', 'ballpark', 'balls', 'balm', 'balmex', 'baloney', 'balsalmic', 'balsami
c', 'balsamico', 'balsamics', 'baltasar', 'baltimore', 'baluchi', 'bam', 'bambe
rg', 'bambi', 'bamboo', 'bamboos', 'bamboozled', 'bamilton', 'bamm', 'bammer',
'ban', 'banana', 'bananamon', 'bananas', 'bananna', 'banannas', 'bancha', 'ban
d', 'bandages', 'banded', 'bandera', 'bandini', 'bandit', 'bands', 'bane', 'ban
ed', 'bang', 'banged', 'banger', 'bangers', 'banging', 'bangkok', 'bangladesh',
'bangled', 'bangok', 'bangor', 'bangs', 'banh', 'banish', 'banishing', 'banjo',
'bank', 'banking', 'bankrupt', 'banks', 'bannana', 'banned', 'banner', 'bannin
g', 'banquet', 'banquets', 'bans', 'banshee', 'bao', 'baquettes', 'bar', 'bar
b']
```

In [6]:

```
#Using GridSearchCV with L1 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l1 = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scor
model_l1.fit(train_bow, y_train)

GS_OPTIMAL_clf_l1 = model_l1.best_estimator_
print(GS_OPTIMAL_clf_l1)
best_score_model_l1 = model_l1.best_score_
print("\nBest score: ",best_score_model_l1)
test_score_l1 = model_l1.score(test_bow, y_test)
print(test_score_l1)

#Writing data
fileObject = open("./GS_OPTIMAL_clf_l1.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l1,fileObject)   # this writes the object a to the fi
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_model_l1.pkl",'wb')
pickle.dump(best_score_model_l1,fileObject)   # this writes the object a to the f
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l1.pkl",'wb')
pickle.dump(test_score_l1,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

joblib.dump(model_l1,"model_l1.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9581934619956152
0.9533196608405523
```

Out[6]:

```
['model_l1.pkl']
```

In [8]:
```
model_l1 = joblib.load("model_l1.pkl")

fileObject = open("./GS_OPTIMAL_clf_l1.pkl",'rb') # we open the file for reading
GS_OPTIMAL_clf_l1 = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_score_model_l1.pkl",'rb') # we open the file for readi
best_score_model_l1 = pickle.load(fileObject) # load the object from the file

fileObject = open("./test_score_l1.pkl",'rb') # we open the file for reading
test_score_l1 = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l1)
print(best_score_model_l1)
print(test_score_l1)
print(model_l1)
```

```
LogisticRegression(C=0.01, 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)
0.9581934619956152
0.9533196608405523
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'C': [0.0001, 0.01, 1, 100, 10000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [9]:

```
    print("Scores for alphas:")
    print(model_l1.grid_scores_)
    print("Best estimator:")
    print(model_l1.best_estimator_)
    print("Best score:")
    print(model_l1.best_score_)
    print("Best parameters:")
    print(model_l1.best_params_)
```

```
Scores for alphas:
[mean: 0.94177, std: 0.00003, params: {'C': 0.0001}, mean: 0.95819, std: 0.0014
6, params: {'C': 0.01}, mean: 0.95381, std: 0.00061, params: {'C': 1}, mean: 0.
94687, std: 0.00182, params: {'C': 100}, mean: 0.93865, std: 0.00101, params:
{'C': 10000}]
Best estimator:
LogisticRegression(C=0.01, 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)
Best score:
0.9581934619956152
Best parameters:
{'C': 0.01}

C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:761: DeprecationWarning: The grid_scores_ attribute was deprecated in version
0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ att
ribute will not be available from 0.20
  DeprecationWarning)
```

In [10]:

```
#Using GridSearchCV with L2 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l2 = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters, scor
model_l2.fit(train_bow, y_train)

GS_OPTIMAL_clf_l2 = model_l2.best_estimator_
print(GS_OPTIMAL_clf_l2)

best_score_l2 = model_l2.best_score_
print("\nBest score: ",best_score_l2)
test_score_l2 = model_l2.score(test_bow, y_test)
print(test_score_l2)


#Writing data
fileObject = open("./GS_OPTIMAL_clf_l2.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l2,fileObject)   # this writes the object a to the fi
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_l2.pkl",'wb')
pickle.dump(best_score_l2,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l2.pkl",'wb')
pickle.dump(test_score_l2,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

joblib.dump(model_l2,"model_l2.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9578844747026857
0.9519191919191918
```

Out[10]:

```
['model_l2.pkl']
```

In [11]:

```
model_l2 = joblib.load("model_l2.pkl")

fileObject = open("./GS_OPTIMAL_clf_l2.pkl",'rb') # we open the file for reading
GS_OPTIMAL_clf_l2 = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_score_l2.pkl",'rb') # we open the file for reading
best_score_l2 = pickle.load(fileObject) # load the object from the file

fileObject = open("./test_score_l2.pkl",'rb') # we open the file for reading
test_score_l2 = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l2)
print(best_score_l2)
print(test_score_l2)
print(model_l2)
```

```
LogisticRegression(C=0.01, 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)
0.9578844747026857
0.9519191919191918
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=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',
       scoring='f1', verbose=0)
```

In [12]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c1 = LogisticRegression(C=0.01, penalty='l1');
clf_c1.fit(train_bow, y_train);
w = clf_c1.coef_
print(np.count_nonzero(w), w)
```

```
2258 [[-0.01912245  0.          0.          ...  0.          0.
   0.        ]]
```

In [13]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c2 = LogisticRegression(C=0.1, penalty='l1');
clf_c2.fit(train_bow, y_train);
w = clf_c2.coef_
print(np.count_nonzero(w))
```

```
6673
```

In [14]:

```
# More Sparsity (Fewer elements of W* being non-zero) by decreasing Lambda (incre
clf_c3 = LogisticRegression(C=1, penalty='l1');
clf_c3.fit(train_bow, y_train);
w = clf_c3.coef_
print(np.count_nonzero(w))
```

```
7868
```

In [15]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c4 = LogisticRegression(C=10, penalty='l1');
clf_c4.fit(train_bow, y_train);
w = clf_c4.coef_
print(np.count_nonzero(w))
```

```
8288
```

In [16]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c5 = LogisticRegression(C=100, penalty='l1');
clf_c5.fit(train_bow, y_train);
w = clf_c5.coef_
print(np.count_nonzero(w))
print(np.shape(w))
```

```
13114
(1, 39675)
```

In [18]:

```
#perturbation testing
index = []

clf_c1.fit(train_bow, y_train)
w=clf_c1.coef_+0.0000001
#print(np.shape(w))
print("w: ",w)

train_bow.data += 0.001
clf_c1.fit(train_bow, y_train)
w_1=clf_c1.coef_+0.0000001
#print(np.shape(w_1))
print("w_1: ",w_1)

per_array= np.abs((w-w_1)/w)*100
print("per_array: ",per_array)
#print(np.shape(per_array))
```

```
w:  [[-1.91188629e-02  1.00000000e-07  1.00000000e-07 ...  1.00000000e-07
   1.00000000e-07  1.00000000e-07]]
w_1:  [[-1.9107843e-02  1.0000000e-07  1.0000000e-07 ...  1.0000000e-07
   1.0000000e-07  1.0000000e-07]]
per_array:  [[0.05763893 0.         0.         ... 0.         0.         0.
]]
```

In [19]:

```
#between 1st-100th percentile
for i in range(0,101,10):
    Weights = np.percentile(per_array, i)
    print("Weights = ",Weights)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  26138234.36439968
**************************
```

In [20]:

```
#between 90th-100th percentile
for i in range(90,101,1):
    Weights = np.percentile(per_array, i)
    print("Weights = ",Weights)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.021549709583148013
Weights =  0.0640694482884901
Weights =  0.18785448795017512
Weights =  2.1792054013884457
Weights =  88.57803957725572
Weights =  26138234.36439968
**************************
```

In [21]:

```
#between 99th-100th percentile
k=99
for i in range(1,12,1):
    Weights_3 = np.percentile(per_array, k)
    print("Weights = ",Weights_3)
    k+=0.1
print("**************************")
```

```
Weights =  88.57803957725572
Weights =  95.44179855289318
Weights =  99.65272173460104
Weights =  100.01149984024542
Weights =  127.35578706999286
Weights =  221.5114438404176
Weights =  429.08519617188904
Weights =  1049.2464256830942
Weights =  7546.465456957729
Weights =  322191.5821214678
Weights =  26138234.36433728
**************************
```

In [23]:

```
#between 99.5th - 99.6th percentile
Weight_chng = []
#Weight_chng = np.asarray(Weights_4)
k=99.5
for i in range(1,3):
    Weights_3 = np.percentile(per_array, k)
    print("Weights = ",Weights_3)
    Weight_chng.append(Weights_3)
    #Weight_chng = Weights_3
    k+=0.1
print("**************************")

print(Weight_chng)
thrsh = np.abs(Weight_chng[0] - Weight_chng[1]) #taking threshold value for weig
print("Threshold value = ",thrsh)
```

```
Weights =  221.5114438404213
Weights =  429.08519617189756
**************************
[221.5114438404213, 429.08519617189756]
Threshold value =  207.57375233147627
```

In [25]:

```
Weight_chng_3 = []
loop2 = np.shape(train_bow)

for loop1 in range(0,loop2[1]):
    Weight_chng_3[0:loop1] = np.abs((w-w_1)/w)*100
Weight_chng_2 = np.asarray(Weight_chng_3)


loop4 = 0
index2 = []
for loop3 in range(0,loop2[1]):
    if (Weight_chng_2[0,loop3] >= thrsh):
        index2.append(loop3)
        loop4 +=1

leng = len(index2)
print("Threshhold taken for abrupt changes = ",thrsh)
print("Number of features with abrupt changes = ",leng)
print("\n")

feature_names = count_vect.get_feature_names()

for k in index2:
     print("Feature Names: ",feature_names[k])
```

```
Threshhold taken for abrupt changes =  207.57375233147627
Number of features with abrupt changes =  203


Feature Names:  0451155505
Feature Names:  10yrs
Feature Names:  117
Feature Names:  1230
Feature Names:  239
Feature Names:  23lb
Feature Names:  30ct
Feature Names:  30oz
Feature Names:  348
Feature Names:  407
Feature Names:  4ish
Feature Names:  6402
Feature Names:  80gb
Feature Names:  8545
Feature Names:  ablaze
```

In [26]:

```python
# Create logistic regression
logistic = LogisticRegression()

# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter distribution using uniform distribution
C = uniform(loc=0, scale=4)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create randomized search 5-fold cross validation and 100 iterations
clf_rscv = RandomizedSearchCV(logistic, hyperparameters, random_state=1, cv=5)

# Fit randomized search
best_model = clf_rscv.fit(train_bow, y_train)

# View best hyperparameters
best_penalty = best_model.best_estimator_.get_params()['penalty']
print('Best Penalty:',best_penalty )
best_c = best_model.best_estimator_.get_params()['C']
print('Best C:',best_c)

# Predict target vector
predict_test = best_model.predict(test_bow)

#Writing data
fileObject = open("./best_penalty.pkl",'wb')
pickle.dump(best_penalty,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_c.pkl",'wb')
pickle.dump(best_c,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./predict_test.pkl",'wb')
pickle.dump(predict_test,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject
```

```
Best Penalty: l1
Best C: 0.8178089989260697
```

In [27]:

```
fileObject = open("./best_penalty.pkl",'rb') # we open the file for reading
best_penalty = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_c.pkl",'rb') # we open the file for reading
best_c = pickle.load(fileObject) # load the object from the file

fileObject = open("./predict_test.pkl",'rb') # we open the file for reading
predict_test = pickle.load(fileObject) # load the object from the file

print(best_penalty)
print(best_c)
print(predict_test)
```

```
l1
0.8178089989260697
[1 0 1 ... 1 1 1]
```

In [29]:

```
def most_informative_feature_for_binary_classification(vectorizer, classifier, n
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    print("Class 0: Negatives ")
    for coef, feat in topn_class1:
        print (class_labels[0], coef, feat)

    print("\n")
    print("Class 1: Positives ")
    for coef, feat in reversed(topn_class2):
        print (class_labels[1], coef, feat)


most_informative_feature_for_binary_classification(count_vect, clf_c1)
```

```
Class 0: Negatives
0 -0.38459975029538646 not
0 -0.20005237705935414 worst
0 -0.19454206247389794 disappointed
0 -0.15265322646418467 horrible
0 -0.152283957397188 terrible
0 -0.14071948547945395 awful
0 -0.13951639998883827 unfortunately
0 -0.12623767309950973 bland
0 -0.12043856922648584 bad
0 -0.11894286920602978 disappointing


Class 1: Positives
1 0.6033694459260905 great
1 0.44256606189011727 best
1 0.3706455502789717 delicious
1 0.27638293379623935 wonderful
1 0.2604975791630741 good
1 0.2543307903700292 love
1 0.24734545974119804 perfect
1 0.233575414916343 excellent
1 0.2271868789106424 and
1 0.2077492991157471 nice
```

In [ ]:

In [ ]:

```
#TF-IDF
```

In [30]:

```
#tf-idf on train data
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1)) #considering only uni-gram as I
train_tf_idf_nstd = tf_idf_vect.fit_transform(X_train[:,9]) #sparse matrix
test_tfidf_nstd = tf_idf_vect.transform(X_test[:,9])
print(train_tf_idf_nstd.shape)
print(test_tfidf_nstd.shape)
```

```
(42000, 39675)
(18000, 39675)
```

In [31]:

```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(train_tf_idf_nstd)
train_tfidf = std_scal.transform(train_tf_idf_nstd)
test_tfidf = std_scal.transform(test_tfidf_nstd)
```

In [32]:

```
#Using GridSearchCV with L1 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l1_tfidf = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters
model_l1_tfidf.fit(train_tfidf, y_train)

GS_OPTIMAL_clf_l1_tfidf = model_l1.best_estimator_
print(GS_OPTIMAL_clf_l1_tfidf)
best_score_model_l1_tfidf = model_l1.best_score_
print("\nBest score: ",best_score_model_l1_tfidf)
test_score_l1_tfidf = model_l1_tfidf.score(test_tfidf, y_test)
print(test_score_l1_tfidf)

#Writing data
fileObject = open("./GS_OPTIMAL_clf_l1_tfidf.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l1_tfidf,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_model_l1_tfidf.pkl",'wb')
pickle.dump(best_score_model_l1_tfidf,fileObject)   # this writes the object a t
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l1_tfidf.pkl",'wb')
pickle.dump(test_score_l1_tfidf,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

joblib.dump(model_l1_tfidf,"model_l1_tfidf.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9581934619956152
0.9543866102630127
```

Out[32]:

```
['model_l1_tfidf.pkl']
```

In [33]:
```
model_l1_tfidf = joblib.load("model_l1_tfidf.pkl")

fileObject = open("./GS_OPTIMAL_clf_l1_tfidf.pkl",'rb') # we open the file for re
GS_OPTIMAL_clf_l1_tfidf = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_score_model_l1_tfidf.pkl",'rb') # we open the file for
best_score_model_l1_tfidf = pickle.load(fileObject) # load the object from the fi

fileObject = open("./test_score_l1_tfidf.pkl",'rb') # we open the file for readin
test_score_l1_tfidf = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l1_tfidf)
print(best_score_model_l1_tfidf)
print(test_score_l1_tfidf)
print(model_l1_tfidf)
```

```
LogisticRegression(C=0.01, 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)
0.9581934619956152
0.9543866102630127
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'C': [0.0001, 0.01, 1, 100, 10000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [34]:

```
    print("Scores for alphas:")
    print(model_l1_tfidf.grid_scores_)
    print("Best estimator:")
    print(model_l1_tfidf.best_estimator_)
    print("Best score:")
    print(model_l1_tfidf.best_score_)
    print("Best parameters:")
    print(model_l1_tfidf.best_params_)
```

```
Scores for alphas:
[mean: 0.91073, std: 0.00353, params: {'C': 0.0001}, mean: 0.95892, std: 0.0009
0, params: {'C': 0.01}, mean: 0.95363, std: 0.00089, params: {'C': 1}, mean: 0.
94321, std: 0.00090, params: {'C': 100}, mean: 0.93585, std: 0.00059, params:
{'C': 10000}]
Best estimator:
LogisticRegression(C=0.01, 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)
Best score:
0.9589154256304367
Best parameters:
{'C': 0.01}

C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:761: DeprecationWarning: The grid_scores_ attribute was deprecated in version
0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ att
ribute will not be available from 0.20
  DeprecationWarning)
```

In [35]:

```
#Using GridSearchCV with L2 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l2_tfidf = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
model_l2_tfidf.fit(train_tfidf, y_train)

GS_OPTIMAL_clf_l2_tfidf = model_l2_tfidf.best_estimator_
print(GS_OPTIMAL_clf_l2_tfidf)

best_score_l2_tfidf = model_l2_tfidf.best_score_
print("\nBest score: ",best_score_l2_tfidf)
test_score_l2_tfidf = model_l2_tfidf.score(test_tfidf, y_test)
print(test_score_l2_tfidf)


#Writing data
fileObject = open("./GS_OPTIMAL_clf_l2_tfidf.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l2_tfidf,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_l2_tfidf.pkl",'wb')
pickle.dump(best_score_l2_tfidf,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l2_tfidf.pkl",'wb')
pickle.dump(test_score_l2_tfidf,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

joblib.dump(model_l2_tfidf,"model_l2_tfidf.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9535812754753286
0.9443043491796643
```

Out[35]:

```
['model_l2_tfidf.pkl']
```

In [36]:
```
model_l2_tfidf = joblib.load("model_l2_tfidf.pkl")

fileObject = open("./GS_OPTIMAL_clf_l2_tfidf.pkl",'rb') # we open the file for re
GS_OPTIMAL_clf_l2_tfidf = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_score_l2_tfidf.pkl",'rb') # we open the file for readi
best_score_l2_tfidf = pickle.load(fileObject) # load the object from the file

fileObject = open("./test_score_l2_tfidf.pkl",'rb') # we open the file for readi
test_score_l2_tfidf = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l2_tfidf)
print(best_score_l2_tfidf)
print(test_score_l2_tfidf)
```

```
LogisticRegression(C=0.01, 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)
0.9535812754753286
0.9443043491796643
```

In [37]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c1_tfidf = LogisticRegression(C=0.01, penalty='l1');
clf_c1_tfidf.fit(train_tfidf, y_train);
w_tfidf = clf_c1_tfidf.coef_
print(np.count_nonzero(w_tfidf), w_tfidf)
```

```
2579 [[-0.02152811  0.          0.          ...  0.          0.
   0.        ]]
```

In [38]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c2_tfidf = LogisticRegression(C=0.1, penalty='l1');
clf_c2_tfidf.fit(train_tfidf, y_train);
w_tfidf = clf_c2_tfidf.coef_
print(np.count_nonzero(w_tfidf))
```

```
7812
```

In [39]:
```
# More Sparsity (Fewer elements of W* being non-zero) by decreasing Lambda (incre
clf_c3_tfidf = LogisticRegression(C=1, penalty='l1');
clf_c3_tfidf.fit(train_tfidf, y_train);
w_tfidf = clf_c3_tfidf.coef_
print(np.count_nonzero(w_tfidf))
```

```
8892
```

In [40]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c4_tfidf = LogisticRegression(C=10, penalty='l1');
clf_c4_tfidf.fit(train_tfidf, y_train);
w_tfidf = clf_c4_tfidf.coef_
print(np.count_nonzero(w_tfidf))
```

```
9433
```

In [41]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c5_tfidf = LogisticRegression(C=100, penalty='l1');
clf_c5_tfidf.fit(train_tfidf, y_train);
w_tfidf = clf_c5_tfidf.coef_
print(np.count_nonzero(w_tfidf))
print(np.shape(w_tfidf))
```

```
14786
(1, 39675)
```

In [42]:

```
#perturbation testing
index = []

clf_c1_tfidf.fit(train_tfidf, y_train)
w_tfidf=clf_c1_tfidf.coef_+0.0000001
#print(np.shape(w_tfidf))
print("w: ",w_tfidf)

train_tfidf.data += 0.001
clf_c1_tfidf.fit(train_tfidf, y_train)
w_1_tfidf=clf_c1_tfidf.coef_+0.0000001
#print(np.shape(w_1_tfidf))
print("w_1: ",w_1_tfidf)

per_array_tfidf= np.abs((w_tfidf-w_1_tfidf)/w_tfidf)*100
print("per_array: ",per_array_tfidf)
#print(np.shape(per_array_tfidf))
```

```
w:  [[-2.15256299e-02  1.00000000e-07  1.00000000e-07 ...  1.00000000e-07
   1.00000000e-07  1.00000000e-07]]
w_1:  [[-2.15360595e-02  1.00000000e-07  1.00000000e-07 ...  1.00000000e-07
   1.00000000e-07  1.00000000e-07]]
per_array:  [[0.04845198 0.         0.         ... 0.         0.         0.
]]
```

In [43]:

```
#between 1st-100th percentile
for i in range(0,101,10):
    Weights_tfidf = np.percentile(per_array_tfidf, i)
    print("Weights = ",Weights_tfidf)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  15248600.020817772
**************************
```

In [44]:

```
#between 90th-100th percentile
for i in range(90,101,1):
    Weights_tfidf = np.percentile(per_array_tfidf, i)
    print("Weights = ",Weights_tfidf)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.0
Weights =  0.011594057239780128
Weights =  0.03282140473185597
Weights =  0.07021305209216433
Weights =  0.174231255525424l
Weights =  4.852635580005572
Weights =  98.20156920398308
Weights =  15248600.020817772
**************************
```

In [46]:

```
#between 99th-100th percentile
k=99
for i in range(1,12,1):
    Weights_3 = np.percentile(per_array_tfidf, k)
    print("Weights = ",Weights_3)
    k+=0.1
print("**************************")
```

```
Weights =  98.20156920398308
Weights =  99.84137589278305
Weights =  101.4841646890115
Weights =  184.29799787353645
Weights =  399.6296222604461
Weights =  912.0257726016774
Weights =  3532.9671239793374
Weights =  11218.159084083469
Weights =  89352.30241447946
Weights =  1304188.0753517554
Weights =  15248600.020748941
**************************
```

In [47]:

```
#between 99.5th - 99.6th percentile
Weight_chng = []
#Weight_chng = np.asarray(Weights_4)
k=99.5
for i in range(1,3):
    Weights_3 = np.percentile(per_array_tfidf, k)
    print("Weights = ",Weights_3)
    Weight_chng.append(Weights_3)
    #Weight_chng = Weights_3
    k+=0.1
print("**************************")

print(Weight_chng)
thrsh_tfidf = np.abs(Weight_chng[0] - Weight_chng[1]) #taking threshold value fo
print("Threshold= ",thrsh_tfidf)
```

```
Weights =  912.0257726017675
Weights =  3532.967123980692
**************************
[912.0257726017675, 3532.967123980692]
Threshold=  2620.9413513789245
```

In [48]:

```
Weight_chng_3 = []
loop2 = np.shape(train_tfidf)

for loop1 in range(0,loop2[1]):
    Weight_chng_3[0:loop1] = np.abs((w_tfidf-w_1_tfidf)/w_tfidf)*100
Weight_chng_2 = np.asarray(Weight_chng_3)


loop4 = 0
index2 = []
for loop3 in range(0,loop2[1]):
    if (Weight_chng_2[0,loop3] >= thrsh_tfidf):
        index2.append(loop3)
        loop4 +=1

leng = len(index2)
print("Threshhold taken for abrupt changes = ",thrsh_tfidf)
print("Number of abrupt changes in features = ",leng)
print("\n")

feature_names_tfidf = tf_idf_vect.get_feature_names()

for k in index2:
     print("Feature Names: ",feature_names_tfidf[k])
```

```
Threshhold taken for abrupt changes =  2620.9413513789245
Number of abrupt changes in features =  164


Feature Names:  1200cal
Feature Names:  1230
Feature Names:  1601
Feature Names:  1pod
Feature Names:  2pod
Feature Names:  39g
Feature Names:  600x
Feature Names:  6402
Feature Names:  adjacent
Feature Names:  afi
Feature Names:  againgst
Feature Names:  airbag
Feature Names:  animatronic
Feature Names:  assertion
Feature Names:  atis
```

In [49]:

```
# Create logistic regression
logistic_tfidf = LogisticRegression()

# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter distribution using uniform distribution
C = uniform(loc=0, scale=4)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create randomized search 5-fold cross validation and 100 iterations
clf_rscv_tfidf = RandomizedSearchCV(logistic_tfidf, hyperparameters, random_stat

# Fit randomized search
best_model_tfidf = clf_rscv_tfidf.fit(train_tfidf, y_train)

# View best hyperparameters
best_penalty_tfidf = best_model_tfidf.best_estimator_.get_params()['penalty']
print('Best Penalty:',best_penalty_tfidf )
best_c_tfidf = best_model_tfidf.best_estimator_.get_params()['C']
print('Best C:',best_c_tfidf)

# Predict target vector
predict_test_tfidf = best_model_tfidf.predict(test_tfidf)

#Writing data
fileObject = open("./best_penalty_tfidf.pkl",'wb')
pickle.dump(best_penalty_tfidf,fileObject)   # this writes the object a to the f
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_c_tfidf.pkl",'wb')
pickle.dump(best_c_tfidf,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./predict_test_tfidf.pkl",'wb')
pickle.dump(predict_test_tfidf,fileObject)   # this writes the object a to the f
fileObject.close() # here we close the fileObject

joblib.dump(clf_rscv_tfidf,"clf_rscv_tfidf.pkl")
```

```
Best Penalty: l1
Best C: 0.8178089989260697
```

Out[49]:

```
['clf_rscv_tfidf.pkl']
```

In [50]:

```
clf_rscv_tfidf = joblib.load("clf_rscv_tfidf.pkl")

fileObject = open("./best_penalty_tfidf.pkl",'rb') # we open the file for readin
best_penalty_tfidf = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_c_tfidf.pkl",'rb') # we open the file for reading
best_c_tfidf = pickle.load(fileObject) # load the object from the file

fileObject = open("./predict_test_tfidf.pkl",'rb') # we open the file for readin
predict_test_tfidf = pickle.load(fileObject) # load the object from the file

print(best_penalty_tfidf)
print(best_c_tfidf)
print(predict_test_tfidf)
print(clf_rscv_tfidf)
```

```
l1
0.8178089989260697
[1 1 1 ... 1 1 1]
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fi
t_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_iter=10, n_jobs=1,
          param_distributions={'C': <scipy.stats._distn_infrastructure.rv_froze
n object at 0x00000265052F1C88>, 'penalty': ['l1', 'l2']},
          pre_dispatch='2*n_jobs', random_state=1, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [51]:

```
def most_informative_feature_for_binary_classification(vectorizer, classifier, n
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    print("Class 0: Negatives ")
    for coef, feat in topn_class1:
        print (class_labels[0], coef, feat)

    print("\n")
    print("Class 1: Positives ")
    for coef, feat in reversed(topn_class2):
        print (class_labels[1], coef, feat)


most_informative_feature_for_binary_classification(tf_idf_vect, clf_c1_tfidf)
```

```
Class 0: Negatives
0 -0.3315682303599515 not
0 -0.1945346986016476 worst
0 -0.1698186644362145 disappointed
0 -0.14349274391074407 terrible
0 -0.1393662466434362 awful
0 -0.13609875601554092 horrible
0 -0.12209055088035092 bland
0 -0.1158811810118024 unfortunately
0 -0.11559974001641002 disappointing
0 -0.11053618714782869 didn


Class 1: Positives
1 0.6262720880595144 great
1 0.47801893243293403 best
1 0.34020487284389883 delicious
1 0.2875742262024318 love
1 0.2677890429058033 good
1 0.2542115453687983 wonderful
1 0.24881977881035244 perfect
1 0.22087164709341847 excellent
1 0.2135477925493301 and
1 0.20866968011684606 nice
```

In [ ]:

```

```

In [ ]:

```
# Avg W2V
```

In [52]:

```
fileObject = open("./final_to_file2.pkl",'rb') # we open the file for reading
final = pickle.load(fileObject) # load the object from the file
```

In [53]:
```python
#w2v
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())

print(type(list_of_sent))
print(final['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
<class 'list'>
witti littl book make son laugh loud recit car drive along alway sing refrain h
es learn whale india droop love new word book introduc silli classic book will
bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car', 'dri
ve', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whale', 'india', 'dr
oop', 'love', 'new', 'word', 'book', 'introduc', 'silli', 'classic', 'book', 'w
ill', 'bet', 'son', 'still', 'abl', 'recit', 'memori', 'colleg']
```

In [54]:
```python
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [55]:
```python
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
#print(len(sent_vectors[0]))
print(type(sent_vectors))
```

```
60000
<class 'list'>
```

In [56]:
```python
# create design matrix X and target vector y
X = np.array(sent_vectors[::]) # end index is exclusive
y = np.array(final['Score']) # showing you two ways of indexing a pandas df
```

In [57]:

```
X_train_nstd = X[0:42000:1]
X_test_nstd = X[42000:60000:1]

y_train_nstd = y[0:42000:1]
y_test_nstd =y[42000:60000:1]

print(X_train_nstd.shape)
print(X_test_nstd.shape)
print(y_train_nstd.shape)
print(y_test_nstd.shape)
```

```
(42000, 50)
(18000, 50)
(42000,)
(18000,)
```

In [58]:

```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(X_train_nstd)
train_avgw2v = std_scal.transform(X_train_nstd)
test_avgw2v = std_scal.transform(X_test_nstd)
```

In [59]:

```
#Using GridSearchCV with L1 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l1_avgw2v = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameter
model_l1_avgw2v.fit(train_avgw2v, y_train)

GS_OPTIMAL_clf_l1_avgw2v = model_l1_avgw2v.best_estimator_
print(GS_OPTIMAL_clf_l1_avgw2v)
best_score_model_l1_avgw2v = model_l1_avgw2v.best_score_
print("\nBest score: ",best_score_model_l1_avgw2v)
test_score_l1_avgw2v = model_l1_avgw2v.score(test_avgw2v, y_test)
print(test_score_l1_avgw2v)

#Writing data
fileObject = open("./GS_OPTIMAL_clf_l1_avgw2v.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l1_avgw2v,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_model_l1_avgw2v.pkl",'wb')
pickle.dump(best_score_model_l1_avgw2v,fileObject)   # this writes the object a
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l1_avgw2v.pkl",'wb')
pickle.dump(test_score_l1_avgw2v,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

joblib.dump(model_l1_avgw2v,"model_l1_avgw2v.pkl")
```

```
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)

Best score:  0.951193222202625
0.9430497513955404
```

Out[59]:

```
['model_l1_avgw2v.pkl']
```

In [60]:

```
model_l1_avgw2v = joblib.load("model_l1_avgw2v.pkl")

fileObject = open("./GS_OPTIMAL_clf_l1_avgw2v.pkl",'rb') # we open the file for
GS_OPTIMAL_clf_l1_avgw2v = pickle.load(fileObject) # load the object from the fi

fileObject = open("./best_score_model_l1_avgw2v.pkl",'rb') # we open the file fo
best_score_model_l1_avgw2v = pickle.load(fileObject) # load the object from the

fileObject = open("./test_score_l1_avgw2v.pkl",'rb') # we open the file for read
test_score_l1_avgw2v = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l1_avgw2v)
print(best_score_model_l1_avgw2v)
print(test_score_l1_avgw2v)
print(model_l1_avgw2v)
```

```
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)
0.951193222202625
0.9430497513955404
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'C': [0.0001, 0.01, 1, 100, 10000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [61]:

```
print("Scores for alphas:")
print(model_l1_avgw2v.grid_scores_)
print("Best estimator:")
print(model_l1_avgw2v.best_estimator_)
print("Best score:")
print(model_l1_avgw2v.best_score_)
print("Best parameters:")
print(model_l1_avgw2v.best_params_)
```

```
Scores for alphas:
[mean: 0.93630, std: 0.00075, params: {'C': 0.0001}, mean: 0.95011, std: 0.0007
6, params: {'C': 0.01}, mean: 0.95117, std: 0.00089, params: {'C': 1}, mean: 0.
95114, std: 0.00079, params: {'C': 100}, mean: 0.95119, std: 0.00083, params:
{'C': 10000}]
Best estimator:
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)
Best score:
0.951193222202625
Best parameters:
{'C': 10000}
```

```
C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:761: DeprecationWarning: The grid_scores_ attribute was deprecated in version
0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ att
ribute will not be available from 0.20
  DeprecationWarning)
```

In [62]:

```
#Using GridSearchCV with L2 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l2_avgw2v = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
model_l2_avgw2v.fit(train_avgw2v, y_train)

GS_OPTIMAL_clf_l2_avgw2v = model_l2_avgw2v.best_estimator_
print(GS_OPTIMAL_clf_l2_avgw2v)

best_score_l2_avgw2v = model_l2_avgw2v.best_score_
print("\nBest score: ",best_score_l2_avgw2v)
test_score_l2_avgw2v = model_l2_avgw2v.score(test_avgw2v, y_test)
print(test_score_l2_avgw2v)


#Writing data
fileObject = open("./GS_OPTIMAL_clf_l2_avgw2v.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l2_avgw2v,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_l2_avgw2v.pkl",'wb')
pickle.dump(best_score_l2_avgw2v,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l2_avgw2v.pkl",'wb')
pickle.dump(test_score_l2_avgw2v,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

joblib.dump(model_l2_avgw2v,"model_l2_avgw2v.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9512152832878302
0.9428162756018139
```

Out[62]:

```
['model_l2_avgw2v.pkl']
```

In [63]:
```
model_l2_avgw2v = joblib.load("model_l2_avgw2v.pkl")

fileObject = open("./GS_OPTIMAL_clf_l2_avgw2v.pkl",'rb') # we open the file for r
GS_OPTIMAL_clf_l2_avgw2v = pickle.load(fileObject) # load the object from the fi

fileObject = open("./best_score_l2_avgw2v.pkl",'rb') # we open the file for read
best_score_l2_avgw2v = pickle.load(fileObject) # load the object from the file

fileObject = open("./test_score_l2_avgw2v.pkl",'rb') # we open the file for read
test_score_l2_avgw2v = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l2_avgw2v)
print(best_score_l2_avgw2v)
print(test_score_l2_avgw2v)
print(model_l2_avgw2v)
```

```
LogisticRegression(C=0.01, 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)
0.9512152832878302
0.9428162756018139
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=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',
       scoring='f1', verbose=0)
```

In [64]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c1_avgw2v = LogisticRegression(C=0.01, penalty='l1');
clf_c1_avgw2v.fit(train_avgw2v, y_train);
w_avgw2v = clf_c1_avgw2v.coef_
print(np.count_nonzero(w_avgw2v), w_avgw2v)
```

```
40 [[ 6.98516932e-02 -1.75376265e-02  7.29592654e-01  2.55101350e-01
   2.10392056e-01  0.00000000e+00  1.47063075e-01 -6.89877947e-02
  -3.09280345e-01  1.84751934e-01 -5.65786826e-04  1.66184680e-01
  -5.52520415e-01 -3.22191197e-02  0.00000000e+00  1.66029593e-01
   4.21394040e-02 -2.52941418e-01  8.03197724e-02  2.71895196e-01
   8.85422133e-02 -6.56343452e-02  4.22168426e-02  0.00000000e+00
   0.00000000e+00  4.37951817e-01  5.41808579e-02 -1.09322279e-02
  -1.47742760e-02  5.00737528e-01  0.00000000e+00  0.00000000e+00
   6.44972917e-02  2.40154904e-01 -4.26238007e-01  5.62423153e-01
  -7.14173349e-02  0.00000000e+00  1.59133104e-01  2.86625452e-01
  -1.82744650e-01  0.00000000e+00  0.00000000e+00 -4.77508154e-01
   6.99983461e-01 -5.49134982e-02 -2.54480896e-01  0.00000000e+00
  -7.57504541e-02  5.78774249e-02]]
```

In [65]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c2_avgw2v = LogisticRegression(C=0.1, penalty='l1');
clf_c2_avgw2v.fit(train_avgw2v, y_train);
w_avgw2v = clf_c2_avgw2v.coef_
print(np.count_nonzero(w_avgw2v))
```

```
48
```

In [66]:

```
# More Sparsity (Fewer elements of W* being non-zero) by decreasing Lambda (incre
clf_c3_avgw2v = LogisticRegression(C=1, penalty='l1');
clf_c3_avgw2v.fit(train_avgw2v, y_train);
w_avgw2v = clf_c3_avgw2v.coef_
print(np.count_nonzero(w_avgw2v))
```

```
50
```

In [67]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c4_avgw2v = LogisticRegression(C=10, penalty='l1');
clf_c4_avgw2v.fit(train_avgw2v, y_train);
w_avgw2v = clf_c4_avgw2v.coef_
print(np.count_nonzero(w_avgw2v))
```

```
50
```

In [70]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c5_avgw2v = LogisticRegression(C=100, penalty='l1');
clf_c5_avgw2v.fit(train_avgw2v, y_train);
w_avgw2v = clf_c5_avgw2v.coef_
print(np.count_nonzero(w_avgw2v))
print(np.shape(w_avgw2v))
```

```
50
(1, 50)
```

In [72]:

```
#perturbation testing
index = []

clf_c1_avgw2v.fit(train_avgw2v, y_train)
w_avgw2v=clf_c1_avgw2v.coef_+0.0000001
#print(np.shape(w_avgw2v))
print("w: ",w_avgw2v)

loop6 = np.shape(train_avgw2v)

for loop7 in range(1,loop6[0],1):
    for loop8 in range(1,loop6[1],1):
        train_avgw2v[loop7,loop8] += 0.001

#train_avgw2v.data += 0.001
clf_c1_avgw2v.fit(train_avgw2v, y_train)
w_1_avgw2v=clf_c1_avgw2v.coef_+0.0000001
#print(np.shape(w_1_avgw2v))
print("w_1: ",w_1_avgw2v)

per_array_avgw2v = np.abs((w_avgw2v-w_1_avgw2v)/w_avgw2v)*100
print("per_array: ",per_array_avgw2v)
#print(np.shape(per_array_avgw2v))
```

```
w:  [[ 6.97723263e-02 -1.62481494e-02  7.29203572e-01  2.55036264e-01
   2.10491821e-01  1.00000000e-07  1.47084969e-01 -6.90965759e-02
  -3.09184901e-01  1.84549089e-01 -5.87179379e-04  1.66430631e-01
  -5.52430393e-01 -3.17011651e-02  1.00000000e-07  1.66005218e-01
   4.24870071e-02 -2.52712497e-01  8.11346011e-02  2.71993720e-01
   8.85462135e-02 -6.54917838e-02  4.26425917e-02  1.00000000e-07
   1.00000000e-07  4.37358803e-01  5.42821133e-02 -1.07624277e-02
  -1.46827992e-02  5.01007415e-01  1.00000000e-07  1.00000000e-07
   6.45495848e-02  2.40418689e-01 -4.26611849e-01  5.62436824e-01
  -7.11630133e-02  1.00000000e-07  1.59461786e-01  2.86855042e-01
  -1.82845092e-01  1.00000000e-07  1.00000000e-07 -4.76998374e-01
   7.00167664e-01 -5.44243859e-02 -2.54344027e-01  1.00000000e-07
  -7.57513483e-02  5.75005556e-02]]
w_1:  [[ 6.94018498e-02 -1.67001213e-02  7.28926089e-01  2.54966245e-01
   2.10294663e-01  1.00000000e-07  1.47230369e-01 -6.90171417e-02
  -3.08737782e-01  1.84445573e-01 -6.42103935e-04  1.65832761e-01
  -5.52503103e-01 -3.13057348e-02  1.00000000e-07  1.66366063e-01
   4.22758918e-02 -2.52831417e-01  8.13456084e-02  2.71914615e-01
   8.86148865e-02 -6.52940206e-02  4.26214262e-02  1.00000000e-07
   1.00000000e-07  4.37867748e-01  5.44317082e-02 -1.05514007e-02
  -1.49627624e-02  5.01089230e-01  1.00000000e-07  1.00000000e-07
   6.45388391e-02  2.40266603e-01 -4.26582631e-01  5.62306447e-01
  -7.16477086e-02  1.00000000e-07  1.59155496e-01  2.86888371e-01
  -1.82592037e-01  1.00000000e-07  1.00000000e-07 -4.77144524e-01
   6.99625330e-01 -5.42614199e-02 -2.54260112e-01  1.00000000e-07
  -7.53208263e-02  5.76363149e-02]]
per_array:  [[5.30979183e-01 2.78168227e+00 3.80528934e-02 2.74543198e-02
  9.36655079e-02 0.00000000e+00 9.88544833e-02 1.14961143e-01
  1.44612162e-01 5.60914040e-02 9.35396535e+00 3.59230511e-01
  1.31616873e-02 1.24736836e+00 0.00000000e+00 2.17369948e-01
  4.96893705e-01 4.70573933e-02 2.60070668e-01 2.90834462e-02
  7.75561984e-02 3.01966432e-01 4.96345197e-02 0.00000000e+00
  0.00000000e+00 1.16367779e-01 2.75587671e-01 1.96077445e+00
```

```
 1.90674268e+00 1.63300882e-02 0.00000000e+00 0.00000000e+00
 1.66470814e-02 6.32589700e-02 6.84870977e-03 2.31807841e-02
 6.81105644e-01 0.00000000e+00 1.92077742e-01 1.16187059e-02
 1.38398881e-01 0.00000000e+00 0.00000000e+00 3.06395597e-02
 7.74576575e-02 2.99435570e-01 3.29927259e-02 0.00000000e+00
 5.68335688e-01 2.36100801e-01]]
```

In [73]:

```
#between 1st-100th percentile
for i in range(0,101,10):
    Weights_avgw2v = np.percentile(per_array_avgw2v, i)
    print("Weights = ",Weights_avgw2v)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.005478967816758141
Weights =  0.02122067331055677
Weights =  0.036028826404350865
Weights =  0.07035831377608062
Weights =  0.11552379731928807
Weights =  0.2229892035533302
Weights =  0.3134192475483672
Weights =  0.737731915585059
Weights =  9.353965345650037
**************************
```

In [74]:

```
#between 90th-100th percentile
for i in range(90,101,1):
    Weights_avgw2v = np.percentile(per_array_avgw2v, i)
    print("Weights = ",Weights_avgw2v)
print("**************************")
```

```
Weights =  0.737731915585059
Weights =  1.0152006481460711
Weights =  1.3001183083054997
Weights =  1.6232117224068074
Weights =  1.9099845839152287
Weights =  1.936460151841278
Weights =  1.9936107616722316
Weights =  2.3958555929333674
Weights =  2.9131279293731396
Weights =  6.133546637511588
Weights =  9.353965345650037
**************************
```

In [75]:

```
#between 99th-100th percentile
k=99
for i in range(1,12,1):
    Weights_3 = np.percentile(per_array_avgw2v, k)
    print("Weights = ",Weights_3)
    k+=0.1
print("**************************")

#No abrupt chnages
```

```
Weights =  6.133546637511588
Weights =  6.4555885083254285
Weights =  6.777630379139269
Weights =  7.099672249953109
Weights =  7.421714120766903
Weights =  7.743755991580695
Weights =  8.065797862394536
Weights =  8.387839733208377
Weights =  8.709881604022216
Weights =  9.031923474836011
Weights =  9.35396534564985
**************************
```

In [76]:

```
# Create logistic regression
logistic_avgw2v = LogisticRegression()

# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter distribution using uniform distribution
C = uniform(loc=0, scale=4)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create randomized search 5-fold cross validation and 100 iterations
clf_rscv_avgw2v = RandomizedSearchCV(logistic_avgw2v, hyperparameters, random_sta

# Fit randomized search
best_model_avgw2v = clf_rscv_avgw2v.fit(train_avgw2v, y_train)

# View best hyperparameters
best_penalty_avgw2v = best_model_avgw2v.best_estimator_.get_params()['penalty']
print('Best Penalty:',best_penalty_avgw2v )
best_c_avgw2v = best_model_avgw2v.best_estimator_.get_params()['C']
print('Best C:',best_c_avgw2v)

# Predict target vector
predict_test_avgw2v = best_model_avgw2v.predict(test_avgw2v)

#Writing data
fileObject = open("./best_penalty_avgw2v.pkl",'wb')
pickle.dump(best_penalty_avgw2v,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_c_avgw2v.pkl",'wb')
pickle.dump(best_c_avgw2v,fileObject)   # this writes the object a to the file
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./predict_test_avgw2v.pkl",'wb')
pickle.dump(predict_test_avgw2v,fileObject)   # this writes the object a to the
fileObject.close() # here we close the fileObject

joblib.dump(clf_rscv_avgw2v,"clf_rscv_avgw2v.pkl")
```

```
Best Penalty: l1
Best C: 0.9183088549193021
```

Out[76]:

```
['clf_rscv_avgw2v.pkl']
```

In [78]:

```
clf_rscv_avgw2v = joblib.load("clf_rscv_avgw2v.pkl")

fileObject = open("./best_penalty.pkl",'rb') # we open the file for reading
best_penalty = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_c.pkl",'rb') # we open the file for reading
best_c = pickle.load(fileObject) # load the object from the file

fileObject = open("./predict_test.pkl",'rb') # we open the file for reading
predict_test = pickle.load(fileObject) # load the object from the file

print(best_penalty_avgw2v)
print(best_c_avgw2v)
print(predict_test_avgw2v)
print(clf_rscv_avgw2v)
```

```
l1
0.9183088549193021
[1 1 1 ... 1 1 1]
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fi
t_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_iter=10, n_jobs=1,
          param_distributions={'C': <scipy.stats._distn_infrastructure.rv_froze
n object at 0x000002651DFBC208>, 'penalty': ['l1', 'l2']},
          pre_dispatch='2*n_jobs', random_state=1, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [ ]:

```
```

In [ ]:

```
# tfidf-W-w2v
```

In [79]:

```
fileObject = open("./final_to_file2.pkl",'rb') # we open the file for reading
final = pickle.load(fileObject) # load the object from the file
```

In [80]:
```
#w2v
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())

print(type(list_of_sent))
print(final['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
<class 'list'>
witti littl book make son laugh loud recit car drive along alway sing refrain h
es learn whale india droop love new word book introduc silli classic book will
bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car', 'dri
ve', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whale', 'india', 'dr
oop', 'love', 'new', 'word', 'book', 'introduc', 'silli', 'classic', 'book', 'w
ill', 'bet', 'son', 'still', 'abl', 'recit', 'memori', 'colleg']
```

In [81]:
```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [82]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [83]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
row=0;
for sent in (list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [84]:

```
print(len(tfidf_sent_vectors))
print(np.shape(tfidf_sent_vectors))
print(type(tfidf_sent_vectors))
```

```
60000
(60000, 50)
<class 'list'>
```

In [85]:

```
# create design matrix X and target vector y
X = np.array(sent_vectors[::]) # end index is exclusive
y = np.array(final['Score']) # showing you two ways of indexing a pandas df
```

In [86]:

```
X_train_nstd = X[0:42000:1]
X_test_nstd = X[42000:60000:1]

y_train_nstd = y[0:42000:1]
y_test_nstd =y[42000:60000:1]

print(X_train_nstd.shape)
print(X_test_nstd.shape)
print(y_train_nstd.shape)
print(y_test_nstd.shape)
```

```
(42000, 50)
(18000, 50)
(42000,)
(18000,)
```

In [87]:

```
# Column Standardization of the tfidf non-standard vector
std_scal = StandardScaler(with_mean=False)
std_scal.fit(X_train_nstd)
train_tfidfww2v = std_scal.transform(X_train_nstd)
test_tfidfww2v = std_scal.transform(X_test_nstd)
```

In [88]:
```
#Using GridSearchCV with L1 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l1_tfidfww2v = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parame
model_l1_tfidfww2v.fit(train_tfidfww2v, y_train)

GS_OPTIMAL_clf_l1_tfidfww2v = model_l1_tfidfww2v.best_estimator_
print(GS_OPTIMAL_clf_l1_tfidfww2v)
best_score_model_l1_tfidfww2v = model_l1_tfidfww2v.best_score_
print("\nBest score: ",best_score_model_l1_tfidfww2v)
test_score_l1_tfidfww2v = model_l1_tfidfww2v.score(test_tfidfww2v, y_test)
print(test_score_l1_tfidfww2v)

#Writing data
fileObject = open("./GS_OPTIMAL_clf_l1_tfidfww2v.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l1_tfidfww2v,fileObject)   # this writes the object a
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_model_l1_tfidfww2v.pkl",'wb')
pickle.dump(best_score_model_l1_tfidfww2v,fileObject)   # this writes the object
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l1_tfidfww2v.pkl",'wb')
pickle.dump(test_score_l1_tfidfww2v,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

joblib.dump(model_l1_tfidfww2v,"model_l1_tfidfww2v.pkl")
```

```
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)

Best score:  0.9511945209425645
0.9430532255604698
```

Out[88]:
```
['model_l1_tfidfww2v.pkl']
```

In [89]:
```
model_l1_tfidfww2v = joblib.load("model_l1_tfidfww2v.pkl")

fileObject = open("./GS_OPTIMAL_clf_l1_tfidfww2v.pkl",'rb') # we open the file fo
GS_OPTIMAL_clf_l1_tfidfww2v = pickle.load(fileObject) # load the object from the

fileObject = open("./best_score_model_l1_tfidfww2v.pkl",'rb') # we open the file
best_score_model_l1_tfidfww2v = pickle.load(fileObject) # load the object from th

fileObject = open("./test_score_l1_tfidfww2v.pkl",'rb') # we open the file for re
test_score_l1_tfidfww2v = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l1_tfidfww2v)
print(best_score_model_l1_tfidfww2v)
print(test_score_l1_tfidfww2v)
print(model_l1_tfidfww2v)
```

```
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)
0.9511945209425645
0.9430532255604698
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'C': [0.0001, 0.01, 1, 100, 10000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [90]:

```
    print("Scores for alphas:")
    print(model_l1_tfidfww2v.grid_scores_)
    print("Best estimator:")
    print(model_l1_tfidfww2v.best_estimator_)
    print("Best score:")
    print(model_l1_tfidfww2v.best_score_)
    print("Best parameters:")
    print(model_l1_tfidfww2v.best_params_)
```

```
Scores for alphas:
[mean: 0.93630, std: 0.00075, params: {'C': 0.0001}, mean: 0.95006, std: 0.0007
9, params: {'C': 0.01}, mean: 0.95116, std: 0.00090, params: {'C': 1}, mean: 0.
95115, std: 0.00087, params: {'C': 100}, mean: 0.95119, std: 0.00087, params:
{'C': 10000}]
Best estimator:
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)
Best score:
0.9511945209425645
Best parameters:
{'C': 10000}

C:\Users\AbhiShek\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:761: DeprecationWarning: The grid_scores_ attribute was deprecated in version
0.18 in favor of the more elaborate cv_results_ attribute. The grid_scores_ att
ribute will not be available from 0.20
  DeprecationWarning)
```

In [91]:

```
#Using GridSearchCV with L2 Regularizer
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
model_l2_tfidfww2v = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parame
model_l2_tfidfww2v.fit(train_tfidfww2v, y_train)

GS_OPTIMAL_clf_l2_tfidfww2v = model_l2_tfidfww2v.best_estimator_
print(GS_OPTIMAL_clf_l2_tfidfww2v)

best_score_l2_tfidfww2v = model_l2_tfidfww2v.best_score_
print("\nBest score: ",best_score_l2_tfidfww2v)
test_score_l2_tfidfww2v = model_l2_tfidfww2v.score(test_tfidfww2v, y_test)
print(test_score_l2_tfidfww2v)


#Writing data
fileObject = open("./GS_OPTIMAL_clf_l2_tfidfww2v.pkl",'wb')
pickle.dump(GS_OPTIMAL_clf_l2_tfidfww2v,fileObject)   # this writes the object a
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_score_l2_tfidfww2v.pkl",'wb')
pickle.dump(best_score_l2_tfidfww2v,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./test_score_l2_tfidfww2v.pkl",'wb')
pickle.dump(test_score_l2_tfidfww2v,fileObject)   # this writes the object a to
fileObject.close() # here we close the fileObject

joblib.dump(model_l2_tfidfww2v,"model_l2_tfidfww2v.pkl")
```

```
LogisticRegression(C=0.01, 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)

Best score:  0.9512152832878302
0.9428162756018139
```

Out[91]:

```
['model_l2_tfidfww2v.pkl']
```

In [92]:
```
model_l2_tfidfww2v = joblib.load("model_l2_tfidfww2v.pkl")

fileObject = open("./GS_OPTIMAL_clf_l2_tfidfww2v.pkl",'rb') # we open the file fo
GS_OPTIMAL_clf_l2_tfidfww2v = pickle.load(fileObject) # load the object from the

fileObject = open("./best_score_l2_tfidfww2v.pkl",'rb') # we open the file for re
best_score_l2_tfidfww2v = pickle.load(fileObject) # load the object from the file

fileObject = open("./test_score_l2_tfidfww2v.pkl",'rb') # we open the file for re
test_score_l2_tfidfww2v = pickle.load(fileObject) # load the object from the file

print(GS_OPTIMAL_clf_l2_tfidfww2v)
print(best_score_l2_tfidfww2v)
print(test_score_l2_tfidfww2v)
print(model_l2_tfidfww2v)
```

```
LogisticRegression(C=0.01, 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)
0.9512152832878302
0.9428162756018139
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=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',
       scoring='f1', verbose=0)
```

In [93]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c1_tfidfww2v = LogisticRegression(C=0.01, penalty='l1');
clf_c1_tfidfww2v.fit(train_tfidfww2v, y_train);
w_tfidfww2v = clf_c1_tfidfww2v.coef_
print(np.count_nonzero(w_tfidfww2v), w_tfidfww2v)
```

```
40 [[ 6.93794145e-02 -1.73492025e-02  7.28943019e-01  2.55132574e-01
   2.09901870e-01  0.00000000e+00  1.46763413e-01 -6.93507855e-02
  -3.09706229e-01  1.84616079e-01 -6.30689661e-04  1.66091419e-01
  -5.52489566e-01 -3.21086184e-02  0.00000000e+00  1.66002546e-01
   4.23860170e-02 -2.53232469e-01  8.08049484e-02  2.72283439e-01
   8.86037163e-02 -6.60818803e-02  4.25747691e-02  0.00000000e+00
   0.00000000e+00  4.36667743e-01  5.41086919e-02 -1.09877426e-02
  -1.49720287e-02  5.00919688e-01  0.00000000e+00  0.00000000e+00
   6.46153761e-02  2.40456495e-01 -4.26418006e-01  5.62057391e-01
  -7.12039084e-02  0.00000000e+00  1.59255535e-01  2.86914651e-01
  -1.83090537e-01  0.00000000e+00  0.00000000e+00 -4.77443924e-01
   7.00013832e-01 -5.46056966e-02 -2.54629268e-01  0.00000000e+00
  -7.64290523e-02  5.80087623e-02]]
```

In [94]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c2_tfidfww2v = LogisticRegression(C=0.1, penalty='l1');
clf_c2_tfidfww2v.fit(train_tfidfww2v, y_train);
w_tfidfww2v = clf_c2_tfidfww2v.coef_
print(np.count_nonzero(w_tfidfww2v))
```

```
48
```

In [95]:

```
# More Sparsity (Fewer elements of W* being non-zero) by decreasing Lambda (incre
clf_c3_tfidfww2v = LogisticRegression(C=1, penalty='l1');
clf_c3_tfidfww2v.fit(train_tfidfww2v, y_train);
w_tfidfww2v = clf_c3_tfidfww2v.coef_
print(np.count_nonzero(w_tfidfww2v))
```

```
50
```

In [96]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c4_tfidfww2v = LogisticRegression(C=10, penalty='l1');
clf_c4_tfidfww2v.fit(train_tfidfww2v, y_train);
w_tfidfww2v = clf_c4_tfidfww2v.coef_
print(np.count_nonzero(w_tfidfww2v))
```

```
50
```

In [97]:

```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lambda (decre
clf_c5_tfidfww2v = LogisticRegression(C=100, penalty='l1');
clf_c5_tfidfww2v.fit(train_tfidfww2v, y_train);
w_tfidfww2v = clf_c5_tfidfww2v.coef_
print(np.count_nonzero(w_tfidfww2v))
print(np.shape(w_tfidfww2v))
```

```
50
(1, 50)
```

In [98]:

```
#perturbation testing
index = []

clf_c1_tfidfww2v.fit(train_tfidfww2v, y_train)
w_tfidfww2v = clf_c1_tfidfww2v.coef_+0.0000001
print(np.shape(w_tfidfww2v))
print("w: ",w_tfidfww2v)

loop6 = np.shape(train_tfidfww2v)

for loop7 in range(1,loop6[0],1):
    for loop8 in range(1,loop6[1],1):
        train_tfidfww2v[loop7,loop8] += 0.001

clf_c1_tfidfww2v.fit(train_tfidfww2v, y_train)
w_1_tfidfww2v=clf_c1_tfidfww2v.coef_+0.0000001
print(np.shape(w_1_tfidfww2v))
print("w_1: ",w_1_tfidfww2v)

per_array_tfidfww2v = np.abs((w_tfidfww2v - w_1_tfidfww2v)/w_tfidfww2v)*100
print("per_array: ",per_array_tfidfww2v)
print(np.shape(per_array_tfidfww2v))
```

```
(1, 50)
w:  [[ 6.93841162e-02 -1.75219511e-02  7.29012364e-01  2.55046877e-01
   2.10196350e-01  1.00000000e-07  1.47150673e-01 -6.92345133e-02
  -3.09702337e-01  1.84726613e-01 -6.31923614e-04  1.65994761e-01
  -5.52373019e-01 -3.22562695e-02  1.00000000e-07  1.66169004e-01
   4.20675291e-02 -2.53614782e-01  8.02750689e-02  2.72308659e-01
   8.81918090e-02 -6.57760672e-02  4.22928492e-02  1.00000000e-07
   1.00000000e-07  4.37054975e-01  5.38621594e-02 -1.10032359e-02
  -1.51373895e-02  5.00821609e-01  1.00000000e-07  1.00000000e-07
   6.43265243e-02  2.40324697e-01 -4.26136320e-01  5.61957845e-01
  -7.16834923e-02  1.00000000e-07  1.59202317e-01  2.87171240e-01
  -1.82982446e-01  1.00000000e-07  1.00000000e-07 -4.77407559e-01
   6.99653039e-01 -5.49652577e-02 -2.54529581e-01  1.00000000e-07
  -7.64389059e-02  5.81360681e-02]]
(1, 50)
w_1:  [[ 6.96362789e-02 -1.63896846e-02  7.29398931e-01  2.55057764e-01
   2.10307305e-01  1.00000000e-07  1.46906016e-01 -6.89805845e-02
  -3.09177183e-01  1.84508927e-01 -6.15306544e-04  1.66257423e-01
  -5.52532524e-01 -3.18834332e-02  1.00000000e-07  1.65999787e-01
   4.25068945e-02 -2.52912522e-01  8.12291535e-02  2.71935069e-01
   8.86308791e-02 -6.57248516e-02  4.27226556e-02  1.00000000e-07
   1.00000000e-07  4.37039460e-01  5.43304253e-02 -1.07483601e-02
  -1.47384670e-02  5.01045085e-01  1.00000000e-07  1.00000000e-07
   6.45517457e-02  2.40510911e-01 -4.26519131e-01  5.62351571e-01
  -7.11169460e-02  1.00000000e-07  1.59384300e-01  2.86907594e-01
  -1.83046824e-01  1.00000000e-07  1.00000000e-07 -4.77111942e-01
   7.00111452e-01 -5.44333218e-02 -2.54289785e-01  1.00000000e-07
  -7.58409398e-02  5.74970568e-02]]
per_array:  [[3.63430017e-01 6.46198865e+00 5.30261226e-02 4.26864617e-03
  5.27866384e-02 0.00000000e+00 1.66262812e-01 3.66766113e-01
  1.69567478e-01 1.17842238e-01 2.62960106e+00 1.58235692e-01
  2.88761936e-02 1.15585698e+00 0.00000000e+00 1.01833935e-01
  1.04442886e+00 2.76900259e-01 1.18851915e+00 1.37193884e-01
  4.97858192e-01 7.78637128e-02 1.01626257e+00 0.00000000e+00
  0.00000000e+00 3.54998666e-03 8.69378291e-01 2.31637131e+00
  2.63534510e+00 4.46218843e-02 0.00000000e+00 0.00000000e+00
  3.50122235e-01 7.74841963e-02 8.98330157e-02 7.00632862e-02
  7.90344129e-01 0.00000000e+00 1.14309436e-01 9.18077268e-02
  3.51823095e-02 0.00000000e+00 0.00000000e+00 6.19214315e-02
  6.55201318e-02 9.67767640e-01 9.42115154e-02 0.00000000e+00
  7.82279836e-01 1.09916507e+00]]
(1, 50)
```

In [99]:

```
#between 1st-100th percentile
for i in range(0,101,10):
    Weights = np.percentile(per_array_tfidfww2v, i)
    print("Weights = ",Weights)
print("**************************")
```

```
Weights =  0.0
Weights =  0.0
Weights =  0.002839989324653288
Weights =  0.05033721217351338
Weights =  0.07451583225277021
Weights =  0.09802272519118367
Weights =  0.16144653994680164
Weights =  0.36443084572609147
Weights =  0.8890561608963095
Weights =  1.159123193004208
Weights =  6.461988648219178
**************************
```

In [100]:

```
#between 90th-100th percentile
for i in range(90,101,1):
    Weights = np.percentile(per_array_tfidfww2v, i)
    print("Weights = ",Weights)
print("**************************")
```

```
Weights =  1.159123193004208
Weights =  1.175127659595327
Weights =  1.27874732445494
Weights =  1.831394880440543
Weights =  2.335165092074807
Weights =  2.4886476691911144
Weights =  2.6298308180299297
Weights =  2.63264539874696
Weights =  2.7118779696103976
Weights =  4.586933308914787
Weights =  6.461988648219178
**************************
```

In [101]:

```
#between 99th-100th percentile
k=99
for i in range(1,12,1):
    Weights_3 = np.percentile(per_array_tfidfww2v, k)
    print("Weights = ",Weights_3)
    k+=0.1
print("**************************")
#No abrupt change in weights
```

```
Weights =  4.586933308914787
Weights =  4.774438842845224
Weights =  4.96194437677566
Weights =  5.149449910706097
Weights =  5.336955444636506
Weights =  5.524460978566914
Weights =  5.7119665124973515
Weights =  5.899472046427787
Weights =  6.086977580358224
Weights =  6.274483114288633
Weights =  6.4619886482190685
**************************
```

In [102]:

```
# Create logistic regression
logistic_tfidfww2v = LogisticRegression()

# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter distribution using uniform distribution
C = uniform(loc=0, scale=4)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create randomized search 5-fold cross validation and 100 iterations
clf_rscv_tfidfww2v = RandomizedSearchCV(logistic_tfidfww2v, hyperparameters, rand

# Fit randomized search
best_model_tfidfww2v = clf_rscv_tfidfww2v.fit(train_tfidfww2v, y_train)

# View best hyperparameters
best_penalty_tfidfww2v = best_model_tfidfww2v.best_estimator_.get_params()['pena
print('Best Penalty:',best_penalty_tfidfww2v )
best_c_tfidfww2v = best_model_tfidfww2v.best_estimator_.get_params()['C']
print('Best C:',best_c_tfidfww2v)

# Predict target vector
predict_test_tfidfww2v = best_model_tfidfww2v.predict(test_tfidfww2v)

#Writing data
fileObject = open("./best_penalty_tfidfww2v.pkl",'wb')
pickle.dump(best_penalty_tfidfww2v,fileObject)   # this writes the object a to t
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./best_c_tfidfww2v.pkl",'wb')
pickle.dump(best_c_tfidfww2v,fileObject)   # this writes the object a to the fil
fileObject.close() # here we close the fileObject

# open the file for writing an array to fileto be save on disk from X_test data
fileObject = open("./predict_test_tfidfww2v.pkl",'wb')
pickle.dump(predict_test_tfidfww2v,fileObject)   # this writes the object a to t
fileObject.close() # here we close the fileObject

joblib.dump(clf_rscv_tfidfww2v,"clf_rscv_tfidfww2v.pkl")
```

```
Best Penalty: l1
Best C: 1.668088018810296
```

Out[102]:

```
['clf_rscv_tfidfww2v.pkl']
```

In [103]:

```
clf_rscv_tfidfww2v = joblib.load("clf_rscv_tfidfww2v.pkl")

fileObject = open("./best_penalty_tfidfww2v.pkl",'rb') # we open the file for rea
best_penalty_tfidfww2v = pickle.load(fileObject) # load the object from the file

fileObject = open("./best_c_tfidfww2v.pkl",'rb') # we open the file for reading
best_c_tfidfww2v = pickle.load(fileObject) # load the object from the file

fileObject = open("./predict_test_tfidfww2v.pkl",'rb') # we open the file for rea
predict_test_tfidfww2v = pickle.load(fileObject) # load the object from the file

print(best_penalty_tfidfww2v)
print(best_c_tfidfww2v)
print(predict_test_tfidfww2v)
print(clf_rscv_tfidfww2v)
```

```
l1
1.668088018810296
[1 1 1 ... 1 1 1]
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fi
t_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_iter=10, n_jobs=1,
          param_distributions={'C': <scipy.stats._distn_infrastructure.rv_froze
n object at 0x00000265197208D0>, 'penalty': ['l1', 'l2']},
          pre_dispatch='2*n_jobs', random_state=1, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [ ]: