Logistic Regression on Amazon Food Reviews

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. index
2. Id
3. ProductId - unique identifier for the product
4. UserId - unqiue identifier for the user
5. ProfileName
6. HelpfulnessNumerator - number of users who found the review helpful
7. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
8. Score - rating between 1 and 5
9. Time - timestamp for the review
10. Summary - brief summary of the review
11. Text - text of the review
12. ProcessedText - Cleaned & Preprocessed Text of the review

Objective: Given Amazon Food reviews, convert all the reviews into a vector using three techniques:
1. Average W2V.
2. Average TFIDF-W2V.
3. GloVe.

Then perform following tasks under each technique:
Task 1. Split train and test data in a ratio of 80:20.
Task 2. Perform GridSearch Cross Validation and Random Search Cross Validation to find optimal Value of λ.

Task 3. Apply Logistic Regression using both L1 and L2 regularizations and report accuracy.
Task 4. Use L1 regularization for different values of λ and report error and sparsity for each value of λ.
Task 5. Check for multi-collinearity of features and find top-10 most important features

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Loading the data

SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently. Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [10]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot

from gensim.models import Word2Vec
import gensim
import csv
import re
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler

from sklearn.cross_validation import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
```

```
C:\Users\GauravP\Anaconda3\lib\site-packages\gensim\utils.py:862: UserWarning: detected Windows; aliasing chunkize to c
hunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
C:\Users\GauravP\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprec
ated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are mov
ed. Also note that the interface of the new CV iterators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\GauravP\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: This module was deprecated
in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. T
his module will be removed in 0.20.
  DeprecationWarning)
```

In [53]:
```
connection = sqlite3.connect('FinalAmazonFoodReviewsDataset.sqlite')
```

In [54]:
```
data = pd.read_sql_query("SELECT * FROM Reviews", connection)
```

In [55]:
```
data.head()
```

Out[55]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	1303862400	Good Quality Dog Food	se
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	1346976000	Not as Advertised	lal
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	1219017600	"Delight" says it all	F . cc a
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	Positive	1350777600	Great taffy	t
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	Positive	1342051200	Nice Taffy	.

In [56]:
```
data.shape
```

Out[56]: (364171, 12)

In [57]:
```
data["Score"].value_counts()
```

Out[57]:
```
Positive    307061
Negative     57110
Name: Score, dtype: int64
```

In [58]:
```
def changingScores(score):
    if score == "Positive":
        return 1
    else:
        return -1
```

In [59]:
```
# changing score
# Positive = 1
# Negative = -1
actualScore = list(data["Score"])
positiveNegative = list(map(changingScores, actualScore)) #map(function, list of numbers)
data['Score'] = positiveNegative
```

In [60]:
```
data.head()
```

Out[60]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I b seve V ca
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	-1	1346976000	Not as Advertised	Pr a label J S Pea
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Thi confe tha aro
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	taff
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	1	1342051200	Nice Taffy	I wil fo or tl

In [61]:
```
allPositiveReviews = data[(data["Score"] == 1)]
```

In [63]:
```
allPositiveReviews.shape
```

Out[63]:
```
(307061, 12)
```

In [64]:
```
positiveReviews_2500 = allPositiveReviews[:2500]
```

In [65]:
```
positiveReviews_2500.shape
```

Out[65]:
```
(2500, 12)
```

In [66]:
```
positiveReviews_2500.head()
```

Out[66]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I bo seve V ca
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Thi confe tha arou
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	(taffy I T v
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	1	1342051200	Nice Taffy	I wil foi orc th
5	6	7	B006K2ZZ7K	A1SP2KVKFXXRU1	David C. Sullivan	0	0	1	1340150400	Great! Just as good as the expensive brands!	salty taffy fl and v

In [68]:
```
allNegativeReviews = data[(data["Score"] == -1)]
```

In [69]:
```
allNegativeReviews.shape
```
Out[69]:
```
(57110, 12)
```
In [70]:
```
negativeReviews_2500 = allNegativeReviews[:2500]
```
In [71]:
```
negativeReviews_2500.shape
```
Out[71]:
```
(2500, 12)
```

In [72]:
```
negativeReviews_2500.head()
```

Out[72]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	T
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	-1	1346976000	Not as Advertised	Prod arri labe Jun Sal Peanu
11	12	13	B0009XLVG0	A327PCT23YH90	LT	1	1	-1	1339545600	My Cats Are Not Fans of the New Food	My c ha be hap eat Felic Pla
15	16	17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	0	-1	1348099200	poor taste	I l eat th and tl are go watc
25	26	27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	1	-1	1332633600	Nasty No flavor	- cand just re No fla . J pla
45	47	51	B001EO5QW8	A108P30XVUFKXY	Roberto A	0	7	-1	1203379200	Don't like it	T oatm is good. mus so

In [73]:
```
frames_5000 = [positiveReviews_2500, negativeReviews_2500]
```

In [74]:
```
FinalPositiveNegative = pd.concat(frames_5000)
```

In [75]:
```
FinalPositiveNegative.shape
```

Out[75]:
```
(5000, 12)
```

In [76]:
```
#Sorting FinalDataframe by "Time"
FinalSortedPositiveNegative_5000 = FinalPositiveNegative.sort_values('Time', axis=0, ascending=True, inplace=False)
```

In [77]:
```
FinalSortedPositiveNegativeScore_5000 = FinalSortedPositiveNegative_5000["Score"]
```

In [78]:
```
FinalSortedPositiveNegative_5000.shape
```

Out[78]:
```
(5000, 12)
```

In [79]:
```
FinalSortedPositiveNegativeScore_5000.shape
```

Out[79]:
```
(5000,)
```

In [80]:
```
Data = FinalSortedPositiveNegative_5000
```

In [81]:
```
Data_Labels = FinalSortedPositiveNegativeScore_5000
```

In [82]:
```
print(Data.shape)
print(Data_Labels.shape)
```

```
(5000, 12)
(5000,)
```

In [83]:
```
Data.head()
```

Out[83]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
772	1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	1	961718400	Great Product	Thi a gooc ar p
771	1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	962236800	WOW Make your own 'slickers' !	rec ship and h
2418	3481	3783	B00016UX0K	AF1PV3DIC0XM7	Robert Ashton	1	2	1	1081555200	Classic Condiment	Mae S Sau becc a st
4417	5897	6386	B000084EK9	A1Z54EM24Y40LL	c2	0	0	-1	1090972800	This stuff is bad!	I hou ha say just b
4408	5888	6376	B000084EKD	A1Z54EM24Y40LL	c2	1	1	-1	1090972800	Needs improved	I ha had ye like o

Average W2V

In [84]:
```
i = 0
listOfSentences = []
for sentence in Data["ProcessedText"].values:
    subSentence = []
    for word in sentence.split():
        subSentence.append(word)

    listOfSentences.append(subSentence)
```

In [85]:
```
print(Data['ProcessedText'].values[0])
print("\n")
print(listOfSentences[0:2])
print("\n")
print(type(listOfSentences))
```

```
this was realli good idea and the final product outstand use the decal car window and everybodi ask where bought the de
cal made two thumb


[['this', 'was', 'realli', 'good', 'idea', 'and', 'the', 'final', 'product', 'outstand', 'use', 'the', 'decal', 'car',
'window', 'and', 'everybodi', 'ask', 'where', 'bought', 'the', 'decal', 'made', 'two', 'thumb'], ['just', 'receiv', 'sh
ipment', 'and', 'could', 'hard', 'wait', 'tri', 'this', 'product', 'love', 'which', 'what', 'call', 'them', 'instead',
'sticker', 'becaus', 'they', 'can', 'remov', 'easili', 'daughter', 'design', 'sign', 'print', 'revers', 'use', 'her',
'car', 'window', 'they', 'print', 'beauti', 'have', 'the', 'print', 'shop', 'program', 'go', 'have', 'lot', 'fun', 'wit
h', 'this', 'product', 'becaus', 'there', 'are', 'window', 'everywher', 'and', 'other', 'surfac', 'like', 'screen', 'an
d', 'comput', 'monitor']]


<class 'list'>
```

In [86]:
```
w2vModel = gensim.models.Word2Vec(listOfSentences, size=400, min_count=5, workers=4)
```

In [87]:
```
# compute average word2vec for each review.
sentenceAsW2V = []
for sentence in listOfSentences:
    sentenceVector = np.zeros(400)
    TotalWordsPerSentence = 0
    for word in sentence:
        try:
            vect = w2vModel.wv[word]
            sentenceVector += vect
            TotalWordsPerSentence += 1
        except:
            pass
    sentenceVector /= TotalWordsPerSentence
    sentenceAsW2V.append(sentenceVector)

print(type(sentenceAsW2V))
print(len(sentenceAsW2V))
print(len(sentenceAsW2V[0]))
```

```
<class 'list'>
5000
400
```

In [88]:
```
standardized_Avg_w2v = StandardScaler().fit_transform(sentenceAsW2V)
print(standardized_Avg_w2v.shape)
print(type(standardized_Avg_w2v))
```

```
(5000, 400)
<class 'numpy.ndarray'>
```

Task 1. Split train and test data in a ratio of 80:20.

In [95]:
```
train_AvgW2V, test_AvgW2V, train_labels_AvgW2V, test_labels_AvgW2V = train_test_split(standardized_Avg_w2v, FinalSortedPo
```

Task 2. Perform GridSearch Cross Validation and Random Search Cross Validation to find optimal Value of λ.

Grid Search

In [96]:
```
clf = LogisticRegression()

hyper_parameters = [{'C': [10**-3, 10**-2, 10**-1, 10**0, 2, 4, 6, 8, 10**1, 50, 10**2, 500, 10**3]}]
bestCV = GridSearchCV(clf, hyper_parameters, scoring = "accuracy", cv = 5)
bestCV.fit(train_AvgW2V, train_labels_AvgW2V)

print(bestCV.best_estimator_)
```

```
LogisticRegression(C=1000, 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 [97]:
```
best_parameter = bestCV.best_params_
best_parameter["C"]
```

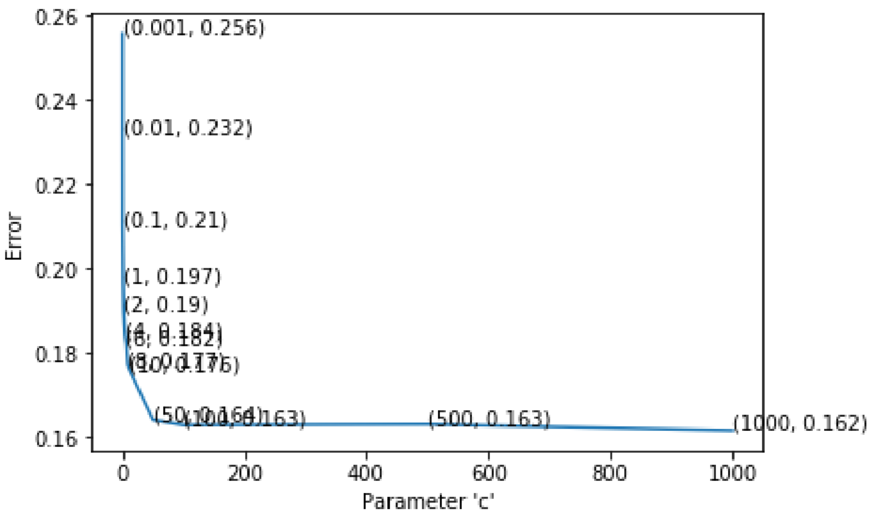
Out[97]:
```
1000
```

In [98]:
```
scoreData = bestCV.grid_scores_
```

In [99]:
```
scoreData
```

Out[99]:
```
[mean: 0.74425, std: 0.02091, params: {'C': 0.001},
 mean: 0.76825, std: 0.01774, params: {'C': 0.01},
 mean: 0.78975, std: 0.01772, params: {'C': 0.1},
 mean: 0.80325, std: 0.01606, params: {'C': 1},
 mean: 0.81025, std: 0.01086, params: {'C': 2},
 mean: 0.81575, std: 0.01549, params: {'C': 4},
 mean: 0.81775, std: 0.01754, params: {'C': 6},
 mean: 0.82275, std: 0.01981, params: {'C': 8},
 mean: 0.82350, std: 0.01954, params: {'C': 10},
 mean: 0.83575, std: 0.01950, params: {'C': 50},
 mean: 0.83700, std: 0.01928, params: {'C': 100},
 mean: 0.83675, std: 0.01737, params: {'C': 500},
 mean: 0.83825, std: 0.01635, params: {'C': 1000}]
```

In [100]:
```
error = []
parameter = []
for i in range(len(scoreData)):
    error.append(1 - scoreData[i][1])
    parameter.append(scoreData[i][0]["C"])

plot.plot(parameter, np.round(error, 4))
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error1 = []
for e in error:
    error1.append(np.round(e,3))

parameter1 = []
for p in parameter:
    parameter1.append(np.round(p,3))

for xy in zip(parameter1, error1):
    plot.annotate(xy,xy)

plot.show()
```



Random Search

In [101]:
```
n = list(np.random.normal(loc=1000, scale=300, size = 500))  #taking 500 numbers which are distributed normally with mean
                                                             #and std-dev = 300
```

In [102]:
```
clf = LogisticRegression()

hyper_parameters2 = {'C': n}

bestCV_random = RandomizedSearchCV(clf, hyper_parameters2, scoring = "accuracy", cv = 3)

bestCV_random.fit(train_AvgW2V, train_labels_AvgW2V)
print(bestCV_random.best_estimator_)
```

```
LogisticRegression(C=584.2338438746062, 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 [103]:
```
best_random_parameter = bestCV_random.best_params_
best_random_parameter["C"]
```

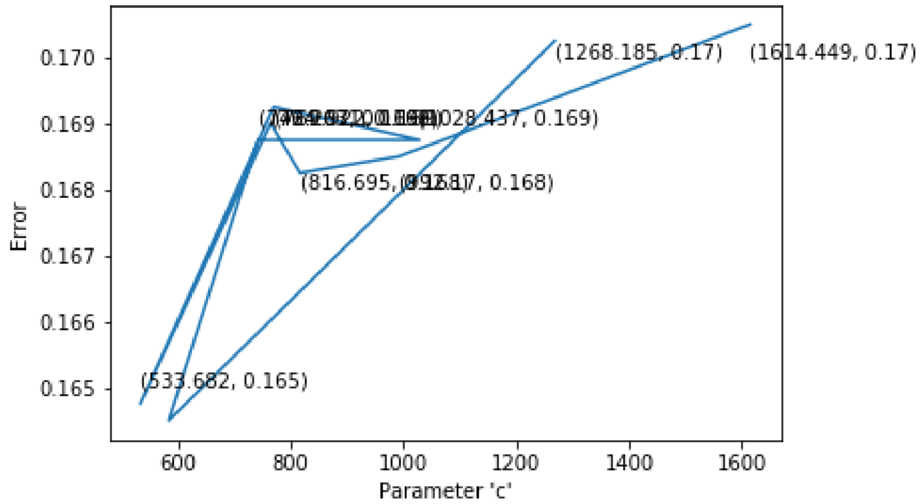
Out[103]:
```
584.2338438746062
```

In [104]:
```
scoreRandomData = bestCV_random.grid_scores_
```

In [105]:
```
scoreRandomData
```

Out[105]:
```
[mean: 0.82975, std: 0.01597, params: {'C': 1268.184913640975},
 mean: 0.83550, std: 0.01679, params: {'C': 584.2338438746062},
 mean: 0.83125, std: 0.01646, params: {'C': 742.206958137426},
 mean: 0.83125, std: 0.01585, params: {'C': 1028.436950767072},
 mean: 0.83075, std: 0.01592, params: {'C': 769.9219219260337},
 mean: 0.83525, std: 0.01575, params: {'C': 533.6818785326748},
 mean: 0.83100, std: 0.01599, params: {'C': 764.631018708247},
 mean: 0.83175, std: 0.01679, params: {'C': 816.6946926002321},
 mean: 0.83150, std: 0.01613, params: {'C': 992.1699084299191},
 mean: 0.82950, std: 0.01440, params: {'C': 1614.448735785675}]
```

In [106]:
```
error2 = []
parameter2 = []
for i in range(len(scoreRandomData)):
    error2.append(1 - scoreRandomData[i][1])
    parameter2.append(scoreRandomData[i][0]["C"])

plot.plot(parameter2, error2)
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error3 = []
for e in error2:
    error3.append(np.round(e,3))

parameter3 = []
for p in parameter2:
    parameter3.append(np.round(p,3))

for xy in zip(parameter3, error3):
        plot.annotate(xy,xy)

plot.show()
```



In [107]:
```
# We are taking our hyper-parameter λ as the average of best λ computed from gridSearchCV and RandomSearchCV
FinalHP = (best_parameter["C"] + best_random_parameter["C"]) / 2
FinalHP
```

Out[107]:
```
792.1169219373031
```

Task 3. Apply Logistic Regression using both L1 and L2 regularizations and report accuracy.

Implementing L2 Regularization

In [108]:
```
model = LogisticRegression(penalty="l2", C = FinalHP)

model.fit(train_AvgW2V, train_labels_AvgW2V)

prediction = model.predict(test_AvgW2V)

AccuracyScore = accuracy_score(test_labels_AvgW2V, prediction) * 100

print("Accuracy score on L2 regularization = "+str(AccuracyScore)+"%")
```

```
Accuracy score on L2 regularization = 83.6%
```

In [109]:
```
Precision = precision_score(test_labels_AvgW2V, prediction)
print("Precision Score on L2 regularization = "+str(Precision))
```

```
Precision Score on L2 regularization = 0.8366935483870968
```

In [110]:
```
Recall = recall_score(test_labels_AvgW2V, prediction)
print("Recall Score on L2 regularization = "+str(Recall))
```

```
Recall Score on L2 regularization = 0.8333333333333334
```

In [111]:
```
Confusion_Matrix = confusion_matrix(test_labels_AvgW2V, prediction)
print("Confusion Matrix on L2 regularization  \n"+str(Confusion_Matrix))
```

```
Confusion Matrix on L2 regularization
[[421  81]
 [ 83 415]]
```

In [116]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_AvgW2V, prediction).ravel()
tn, fp, fn, tp
```

Out[116]:
```
(421, 81, 83, 415)
```

Implementing L1 Regularization

In [112]:
```
model2 = LogisticRegression(penalty="l1", C = FinalHP)

model2.fit(train_AvgW2V, train_labels_AvgW2V)

prediction2 = model2.predict(test_AvgW2V)

AccuracyScore2 = accuracy_score(test_labels_AvgW2V, prediction2) * 100

print("Accuracy score on L1 regularization = "+str(AccuracyScore2)+"%")
```

```
Accuracy score on L1 regularization = 83.2%
```

In [113]:
```
Precision2 = precision_score(test_labels_AvgW2V, prediction2)
print("Precision Score on L1 regularization = "+str(Precision2))
```

```
Precision Score on L1 regularization = 0.8423236514522822
```

In [114]:
```
Recall2 = recall_score(test_labels_AvgW2V, prediction2)
print("Recall Score on L1 regularization = "+str(Recall2))
```

```
Recall Score on L1 regularization = 0.8152610441767069
```

In [115]:
```
Confusion_Matrix2 = confusion_matrix(test_labels_AvgW2V, prediction2)
print("Confusion Matrix on L1 regularization \n"+str(Confusion_Matrix2))
```

```
Confusion Matrix on L1 regularization
[[426  76]
 [ 92 406]]
```

In [117]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_AvgW2V, prediction2).ravel()
tn, fp, fn, tp
```

Out[117]:
```
(426, 76, 92, 406)
```

Task 4. Use L1 regularization for different values of λ and report error and sparsity for each value of λ.

In [118]:
```
model3 = LogisticRegression(penalty="l1", C = 10000)

model3.fit(train_AvgW2V, train_labels_AvgW2V)

accuracyScore_10000 = model3.score(test_AvgW2V, test_labels_AvgW2V)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.0001 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.0001 = "+str(error))
```

```
Number of non-zero for λ value of 0.0001 = 400
Error for λ value of 0.0001 = 0.16900000000000004
```

In [119]:
```
model3 = LogisticRegression(penalty="l1", C = 1000)

model3.fit(train_AvgW2V, train_labels_AvgW2V)

accuracyScore_1000 = model3.score(test_AvgW2V, test_labels_AvgW2V)

error = 1 - accuracyScore_1000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.001 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.001 = "+str(error))
```

```
Number of non-zero for λ value of 0.001 = 400
Error for λ value of 0.001 = 0.17100000000000004
```

In [120]:
```
model3 = LogisticRegression(penalty="l1", C = 10)

model3.fit(train_AvgW2V, train_labels_AvgW2V)

accuracyScore_10 = model3.score(test_AvgW2V, test_labels_AvgW2V)

error = 1 - accuracyScore_10

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.1 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.1 = "+str(error))
```

```
Number of non-zero for λ value of 0.1 = 230
Error for λ value of 0.1 = 0.18000000000000005
```

In [121]:
```
model3 = LogisticRegression(penalty="l1", C = 10**-3)

model3.fit(train_AvgW2V, train_labels_AvgW2V)

accuracyScore_0001 = model3.score(test_AvgW2V, test_labels_AvgW2V)

error = 1 - accuracyScore_0001

weightVector = model3.coef_

print("Number of non-zero for λ value of 1000 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 1000 = "+str(error))
```

```
Number of non-zero for λ value of 1000 = 0
Error for λ value of 1000 = 0.498
```

In [122]:
```
model3 = LogisticRegression(penalty="l1", C = 10**-4)

model3.fit(train_AvgW2V, train_labels_AvgW2V)

accuracyScore_00001 = model3.score(test_AvgW2V, test_labels_AvgW2V)

error = 1 - accuracyScore_00001

weightVector = model3.coef_

print("Number of non-zero for λ value of 10000 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 10000 = "+str(error))
```

```
Number of non-zero for λ value of 10000 = 0
Error for λ value of 10000 = 0.498
```

Task 5. Check for multi-collinearity of features and find top-10 most important features

In [140]:
```
# computing weight vector prior to adding noise in out data
model4 = LogisticRegression(penalty="l2", C = FinalHP)

model4.fit(train_AvgW2V, train_labels_AvgW2V)

weightVector1 = model4.coef_
```

In [142]:
```
noise = np.random.normal(loc=0.1, scale=0.1) # generating random positive number from normal distribution
noise_matrix = np.full((4000, 400), noise) # generating matrix of size 4000 * 400 where every number in matrix is 'n'
dataWithNoise_AvgW2V = noise_matrix + train_AvgW2V # adding noise to training data fro pertubation test

print(noise)
print(dataWithNoise_AvgW2V.shape)
```

```
0.08064478322530538
(4000, 400)
```

In [143]:
```
# computing weight vector after adding noise in out data
model5 = LogisticRegression(penalty="l2", C = FinalHP)

model5.fit(dataWithNoise_AvgW2V, train_labels_AvgW2V)

weightVector2 = model5.coef_
```

In [144]:
```
differenceInWeights = weightVector1 - weightVector2
```

In [147]:
```
differenceInWeights.ravel()
```

Out[147]:
```
array([ 1.24805364e-01,  1.70932769e-01, -1.11871736e-02,  1.03912758e-01,
        7.93223365e-02,  4.18818609e-01,  1.36661723e-01,  4.56989891e-01,
       -3.92898222e-02,  1.98360903e-01, -9.05991926e-02,  1.49630554e-01,
        4.45303410e-01, -2.78539118e-01, -2.28059601e-01,  9.20603735e-02,
        5.06654285e-01,  2.77161689e-01,  3.67958794e-01, -2.30820079e-01,
        2.28275354e-01,  2.03324644e-01,  3.78095885e-01,  7.99650271e-01,
       -1.28208345e-01,  3.46592046e-01,  4.44813411e-01,  9.70867561e-02,
        2.07690249e-01,  2.95724366e-01, -3.50075373e-01,  1.00622917e+00,
        1.61358295e-01,  8.61132850e-02,  1.40308026e-01,  2.56285727e-01,
        4.15229139e-02,  6.16934756e-01,  5.63410977e-01,  1.85443428e-01,
       -1.09950435e-01,  1.06918044e+00,  3.31539367e-01,  8.17591979e-02,
        8.78324226e-01,  8.58061135e-02,  2.94267113e-01,  1.06538155e+00,
        1.20842287e+00,  2.76184956e-01,  4.25152321e-01,  4.43081384e-01,
       -5.81952172e-02,  3.58375538e-01,  5.85569475e-01,  2.97492574e-01,
        3.03267071e-01,  8.26227974e-02,  2.65602719e-01,  1.02464454e-02,
        4.21451227e-01,  2.00680818e-01,  4.96993374e-01,  1.03055776e-01,
       -7.40148166e-02,  2.13286517e-01,  2.46906956e-01,  1.88369034e-01,
        5.24534264e-01, -3.20966633e-01,  7.27964680e-01,  9.45298222e-01,
        2.29459372e-01,  4.16584121e-01,  6.61523893e-01,  4.99814342e-01,
        8.29048009e-01,  3.68537195e-01,  1.94110578e-01,  4.42285596e-01,
        2.56450048e-01, -3.48518964e-02,  1.89228769e-01,  5.46985398e-01,
        9.49375835e-01,  4.58690126e-01, -2.67423812e-02,  3.40964963e-01,
        5.54421597e-01,  4.84868998e-01,  2.40709530e-02,  3.58396693e-01,
        1.49007996e-01, -1.83513260e-01,  1.24827919e-01,  4.79384202e-01,
        2.00794188e-01,  3.06368654e-01, -1.18141115e-01,  3.72210156e-01,
        2.51906344e-01,  3.51717187e-01,  5.41065366e-01, -7.35527221e-02,
       -1.58340831e-01,  4.87058281e-01,  8.95438049e-01,  1.03854550e-01,
        1.00439286e-01,  2.93393722e-01, -5.07122478e-01,  5.04931285e-01,
        6.30164023e-01,  3.94235764e-02,  1.05508795e+00,  1.03311885e+00,
        6.68149670e-01,  3.74619866e-01,  3.12626179e-01,  3.05340279e-01,
       -2.56234661e-01, -2.65207201e-01, -1.82099370e-01,  2.57596511e-01,
        2.16413057e-03,  3.28279146e-01, -7.92442011e-02,  3.25958290e-01,
        2.39980832e-01, -1.49201676e-01,  2.42135338e-01,  4.47915127e-02,
       -3.49030350e-02,  1.17165109e-01,  5.61850684e-01, -3.28313918e-01,
       -7.43693536e-02,  7.85999682e-01,  3.55374369e-01,  5.55577781e-02,
        3.79739592e-01,  2.76772539e-01,  2.94876076e-01,  2.36463420e-01,
        4.87127108e-01,  1.97415952e-01, -1.62836862e-03, -1.24705697e-01,
        1.14246184e-02,  2.08988574e-01,  1.24318222e-01,  1.93498681e-01,
        1.00648334e-01,  3.20811791e-01,  3.32418195e-01,  7.67962046e-01,
```

```
         2.82235083e-01,  7.16188788e-01,  6.35167830e-01,  2.22631369e-01,
        -6.29899991e-02,  7.31413939e-01,  6.59577503e-01,  1.78584225e-01,
         1.91347480e-01,  1.78972716e-01, -2.03139938e-02,  4.50327053e-01,
         9.04072322e-02,  2.20986228e-01,  3.50507749e-01, -2.69136334e-01,
         4.04063262e-02, -1.60356872e-01,  1.83957421e-01,  3.77319095e-01,
         5.38471832e-01,  8.89826232e-02,  5.02519959e-01,  6.07566272e-01,
         6.50689063e-01,  3.67674446e-01,  2.96181834e-01,  4.98253668e-01,
         7.74590255e-01, -4.14652772e-02,  4.51390911e-01, -1.09632072e-01,
         5.37925406e-02,  3.91369876e-01,  5.13204841e-02,  8.62799391e-02,
         1.39188084e-01, -8.02947543e-02,  1.07248938e-01,  5.26179986e-01,
         2.08525487e-01,  8.00284616e-01,  1.51087920e-01,  5.64281566e-01,
        -2.64850973e-01,  3.24668607e-01,  4.43954915e-01,  4.26527130e-01,
         2.39677775e-01,  3.59030805e-01,  6.41259226e-01,  1.26441581e-02,
         6.09254650e-01,  6.55294745e-01,  7.14076365e-01,  5.53791730e-01,
         5.47820777e-01,  4.40104004e-02,  1.61113477e-01,  6.85916387e-01,
         6.47760074e-01, -1.33268531e-02,  4.50506669e-01, -3.99707516e-02,
         5.46508010e-01, -3.89394944e-02,  9.49167316e-01,  9.92742012e-01,
         5.77666693e-02,  6.42723662e-02,  1.31996462e-01,  2.29660671e-01,
         5.18451997e-01,  2.87799759e-01,  7.10948107e-01, -3.01324531e-03,
         2.48263845e-01, -1.15256396e-01,  4.74086275e-01, -9.96701291e-02,
         2.40030283e-01,  3.26822019e-01,  3.09829494e-01,  3.42286955e-01,
        -7.94562284e-02,  1.22285833e-01,  2.22504277e-01,  1.96888498e-01,
         4.42124069e-02,  2.08337127e-01,  8.37378645e-02,  5.76794718e-02,
         6.86462117e-01,  3.97984954e-01,  2.15607237e-01,  1.33977515e-01,
         3.67862358e-01,  3.23858732e-01,  1.79866645e-01,  2.50038776e-01,
         7.39319661e-01,  3.06798441e-01,  4.06416306e-01,  7.07730483e-02,
        -4.20182817e-04,  5.07370981e-01,  2.69692759e-01,  3.61643504e-01,
         1.01138441e-01,  3.10386477e-02,  5.05332470e-01,  3.63870479e-01,
         3.24108477e-01,  4.35835206e-01,  1.71558620e-01,  2.80191649e-01,
         1.41587752e-01,  2.74615010e-01,  5.59725645e-01, -9.91373448e-02,
         3.37538566e-01,  3.06475001e-01,  4.10064222e-01,  3.52169404e-01,
         4.95157461e-01,  4.77048757e-01,  4.52725310e-01,  3.07093521e-01,
         3.80153702e-01,  1.76876748e-01,  1.42553019e-01,  1.06393530e-01,
         2.25627098e-01,  4.02454357e-01,  5.44444671e-01, -3.19736744e-01,
         7.53514463e-01,  6.34355877e-01,  1.23178748e-01,  5.89458801e-01,
         4.48672103e-01,  1.91204387e-01,  6.42749555e-01,  4.36002899e-03,
         2.67609836e-01,  1.00810757e+00,  1.32856538e+00,  1.39763759e-01,
        -4.91160727e-02,  1.77639282e-01,  1.64558945e-01,  1.54511543e-01,
         4.04451858e-01,  2.70437412e-01,  6.14438121e-01,  2.54836260e-01,
         1.26968403e-01,  7.09930554e-01,  1.57577183e-01,  4.22352021e-01,
        -7.02333946e-03,  4.85025832e-02,  7.70285887e-01,  1.70478325e-04,
         2.17975130e-01,  4.67651921e-01,  3.16562708e-01,  8.90707878e-02,
```

```
        1.15799794e-01,  1.60488416e-01,  1.18044912e-01, -3.02497989e-01,
       -1.18970001e-01,  5.18692957e-01, -3.38028071e-01,  2.21294104e-01,
        2.07167986e-01,  3.02691200e-01, -1.56786088e-01,  9.22424338e-01,
        3.13854099e-01,  6.55860242e-03,  5.08907949e-01,  5.55281142e-01,
        3.04803265e-01,  4.34123837e-01,  4.79045548e-01,  3.41044499e-01,
        5.83144990e-02,  8.64363875e-02,  2.49431756e-01,  7.29173082e-01,
        8.10117595e-01,  3.76805438e-01,  4.88921826e-01,  3.21518333e-01,
        5.56678525e-01,  4.72075033e-01,  3.46593438e-01, -1.54011655e-01,
        5.69432685e-01,  2.42449288e-01, -3.37470324e-02,  3.47602309e-01,
        5.50275167e-01,  8.07253974e-02,  3.15475905e-01,  6.11767181e-02,
        3.34535894e-02, -7.33594160e-02, -1.12946983e-01,  3.03065416e-01,
       -2.88330692e-02,  5.89248325e-03,  2.54087495e-01,  6.90436342e-01,
        8.16437816e-01,  6.13717577e-01,  6.63999597e-01, -1.63747183e-02,
        2.68326393e-01,  3.37025185e-01,  2.23616180e-01,  2.15017996e-01,
        3.08547628e-01, -1.06558366e-01,  5.63531097e-01,  1.98735600e-01,
        4.09682318e-01,  1.05233156e-01,  6.67351564e-01,  9.82870751e-01,
        3.06501693e-01,  3.73773431e-01,  4.32641497e-01,  1.74075573e-01,
       -6.41460824e-02,  4.05255089e-01,  3.65375878e-01,  2.79768806e-01,
        4.68523915e-01,  3.57768142e-01,  1.63550609e-01,  4.15214299e-02])
```

Here, as you can see that the differnce in the weight vectors prior and after adding noise is high. It means that features are collinear, hence we cannot use weight vectors as feature importance. Since word to vec generates vectors for words which are dependent on each other owing to their similarity of preserving semetic meaning between similar words. Therefore, features in word 2 vec are collinear

TFIDF-W2V

In [148]:
```
tfidf_vect = TfidfVectorizer(ngram_range = (1,2))
tfidf = tfidf_vect.fit_transform(Data["ProcessedText"].values)
```

In [149]:
```
w2v_Model = gensim.models.Word2Vec(listOfSentences, size=300, min_count=5, workers=4)
```

In [150]:
```
print(tfidf.shape)
print(type(tfidf))
```

```
(5000, 141225)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [151]:
```
# TF-IDF weighted Word2Vec
tfidf_features = tfidf_vect.get_feature_names()

tfidf_w2v = []
reviews = 0

for sentence in listOfSentences:
    sentenceVector = np.zeros(300)
    weightTfidfSum = 0
    for word in sentence:
        try:
            W2V_Vector = w2v_Model.wv[word]
            tfidfVector = tfidf[reviews, tfidf_features.index(word)]
            sentenceVector += (W2V_Vector * tfidfVector)
            weightTfidfSum += tfidfVector
        except:
            pass
    sentenceVector /= weightTfidfSum
    tfidf_w2v.append(sentenceVector)
    reviews += 1
```

In [152]:
```
standardized_tfidf_w2v = StandardScaler().fit_transform(tfidf_w2v)
print(standardized_tfidf_w2v.shape)
print(type(standardized_tfidf_w2v))
```

```
(5000, 300)
<class 'numpy.ndarray'>
```

Task 1. Split train and test data in a ratio of 80:20.

In [154]:
```
train_tfidf_w2v, test_tfidf_w2v, train_labels_tfidf_w2v, test_labels_tfidf_w2v = train_test_split(standardized_tfidf_w2v,
```

Task 2. Perform GridSearch Cross Validation and Random Search Cross Validation to find optimal Value of λ.

Grid Search

In [158]:
```
clf = LogisticRegression()

hyper_parameters = [{'C': [10**-3, 10**-2, 10**-1, 10**0, 2, 4, 6, 8, 10**1, 50, 10**2, 500, 10**3]}]
bestCV = GridSearchCV(clf, hyper_parameters, scoring = "accuracy", cv = 5)
bestCV.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

print(bestCV.best_estimator_)
```

```
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)
```

In [159]:
```
best_parameter = bestCV.best_params_
best_parameter["C"]
```

Out[159]:
```
100
```

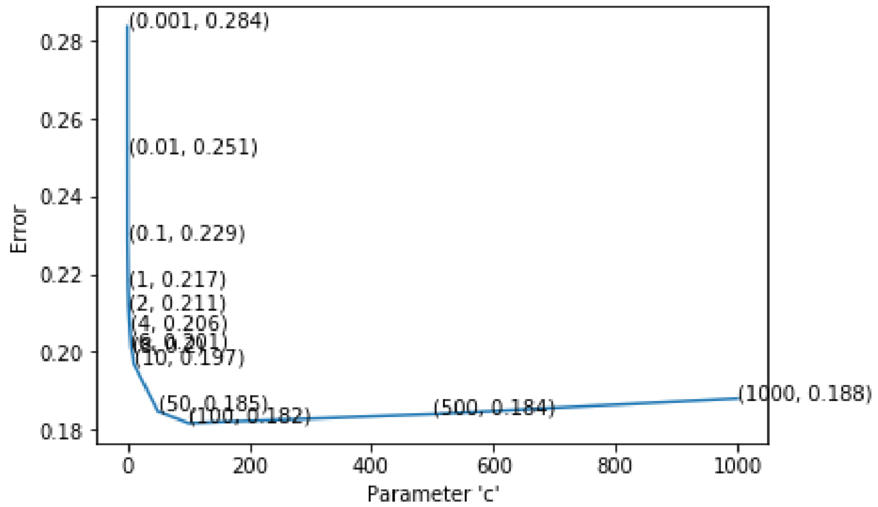
In [160]:
```
scoreData = bestCV.grid_scores_
```

In [161]:
```
scoreData
```

Out[161]:
```
[mean: 0.71625, std: 0.00749, params: {'C': 0.001},
 mean: 0.74875, std: 0.01448, params: {'C': 0.01},
 mean: 0.77125, std: 0.02277, params: {'C': 0.1},
 mean: 0.78325, std: 0.02076, params: {'C': 1},
 mean: 0.78925, std: 0.01894, params: {'C': 2},
 mean: 0.79425, std: 0.02060, params: {'C': 4},
 mean: 0.79900, std: 0.02077, params: {'C': 6},
 mean: 0.80050, std: 0.02104, params: {'C': 8},
 mean: 0.80300, std: 0.01889, params: {'C': 10},
 mean: 0.81525, std: 0.01951, params: {'C': 50},
 mean: 0.81825, std: 0.01816, params: {'C': 100},
 mean: 0.81600, std: 0.02139, params: {'C': 500},
 mean: 0.81200, std: 0.02279, params: {'C': 1000}]
```

In [162]:

```
error = []
parameter = []
for i in range(len(scoreData)):
    error.append(1 - scoreData[i][1])
    parameter.append(scoreData[i][0]["C"])

plot.plot(parameter, np.round(error, 4))
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error1 = []
for e in error:
    error1.append(np.round(e,3))

parameter1 = []
for p in parameter:
    parameter1.append(np.round(p,3))

for xy in zip(parameter1, error1):
    plot.annotate(xy,xy)

plot.show()
```



Random Search

In [168]:
```
n = list(np.random.normal(loc=100, scale=40, size = 500))  #taking 500 numbers which are distributed normally with mean =
                                                             #and std-dev = 40
```

In [170]:
```
clf = LogisticRegression()

hyper_parameters2 = {'C': n}

bestCV_random = RandomizedSearchCV(clf, hyper_parameters2, scoring = "accuracy", cv = 3)

bestCV_random.fit(train_tfidf_w2v, train_labels_tfidf_w2v)
print(bestCV_random.best_estimator_)
```

```
LogisticRegression(C=107.56924891512601, 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 [171]:
```
best_random_parameter = bestCV_random.best_params_
best_random_parameter["C"]
```

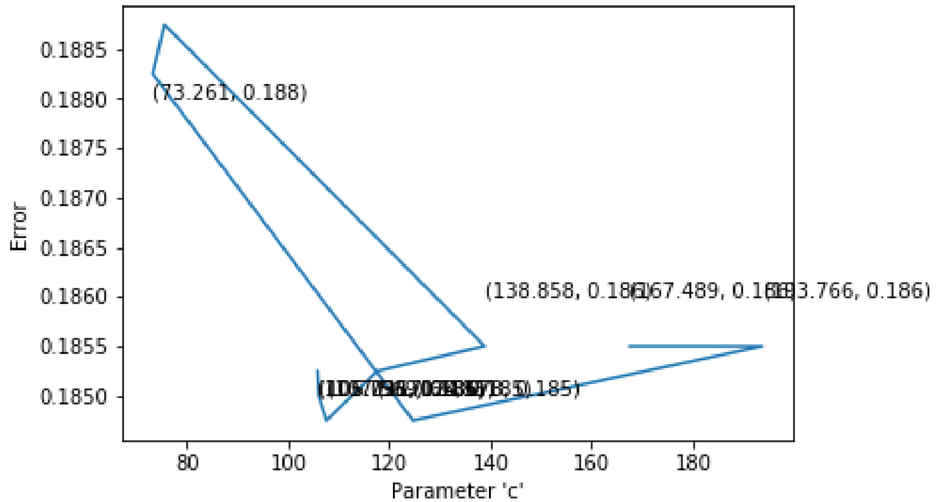
Out[171]:
```
107.56924891512601
```

In [172]:
```
scoreRandomData = bestCV_random.grid_scores_
```

In [173]:
```
scoreRandomData
```

Out[173]:
```
[mean: 0.81475, std: 0.01267, params: {'C': 105.79602841658702},
 mean: 0.81500, std: 0.01234, params: {'C': 106.03462308100598},
 mean: 0.81525, std: 0.01239, params: {'C': 107.56924891512601},
 mean: 0.81475, std: 0.01352, params: {'C': 117.66038469974445},
 mean: 0.81450, std: 0.01239, params: {'C': 138.8575850139698},
 mean: 0.81125, std: 0.01322, params: {'C': 75.54013058206478},
 mean: 0.81175, std: 0.01303, params: {'C': 73.26089309251445},
 mean: 0.81525, std: 0.01384, params: {'C': 124.6782235005891},
 mean: 0.81450, std: 0.01218, params: {'C': 193.7661208459057},
 mean: 0.81450, std: 0.01299, params: {'C': 167.489215456119}]
```

In [174]:
```
error2 = []
parameter2 = []
for i in range(len(scoreRandomData)):
    error2.append(1 - scoreRandomData[i][1])
    parameter2.append(scoreRandomData[i][0]["C"])

plot.plot(parameter2, error2)
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error3 = []
for e in error2:
    error3.append(np.round(e,3))

parameter3 = []
for p in parameter2:
    parameter3.append(np.round(p,3))

for xy in zip(parameter3, error3):
        plot.annotate(xy,xy)

plot.show()
```



In [175]:
```
# We are taking our hyper-parameter λ as the average of best λ computed from gridSearchCV and RandomSearchCV
FinalHP = (best_parameter["C"] + best_random_parameter["C"]) / 2
FinalHP
```

Out[175]:
```
103.78462445756301
```

Task 3. Apply Logistic Regression using both L1 and L2 regularizations and report accuracy.

Implementing L2 Regularization

In [176]:
```
model = LogisticRegression(penalty="l2", C = FinalHP)

model.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

prediction = model.predict(test_tfidf_w2v)

AccuracyScore = accuracy_score(test_labels_tfidf_w2v, prediction) * 100

print("Accuracy score on L2 regularization = "+str(AccuracyScore)+"%")
```

```
Accuracy score on L2 regularization = 80.7%
```

In [179]:
```
Precision = precision_score(test_labels_tfidf_w2v, prediction)
print("Precision Score on L2 regularization = "+str(Precision))
```

```
Precision Score on L2 regularization = 0.8157349896480331
```

In [188]:
```
Recall = recall_score(test_labels_tfidf_w2v, prediction)
print("Recall Score on L2 regularization = "+str(Recall))
```

```
Recall Score on L2 regularization = 0.7911646586345381
```

In [189]:
```
Confusion_Matrix = confusion_matrix(test_labels_tfidf_w2v, prediction)
print("Confusion Matrix on L2 regularization \n"+str(Confusion_Matrix))
```

```
Confusion Matrix on L2 regularization
[[413  89]
 [104 394]]
```

In [187]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_tfidf_w2v, prediction).ravel()
tn, fp, fn, tp
```

Out[187]:
```
(413, 89, 104, 394)
```

Implementing L1 Regularization

In [191]:
```
model2 = LogisticRegression(penalty="l1", C = FinalHP)

model2.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

prediction2 = model2.predict(test_tfidf_w2v)

AccuracyScore2 = accuracy_score(test_labels_tfidf_w2v, prediction2) * 100

print("Accuracy score on L1 regularization = "+str(AccuracyScore2)+"%")
```

```
Accuracy score on L1 regularization = 80.7%
```

In [192]:
```
Precision2 = precision_score(test_labels_tfidf_w2v, prediction2)
print("Precision Score on L1 regularization = "+str(Precision2))
```

```
Precision Score on L1 regularization = 0.8144329896907216
```

In [193]:
```
Recall2 = recall_score(test_labels_tfidf_w2v, prediction2)
print("Precision Score on L1 regularization = "+str(Precision2))
```

```
Precision Score on L1 regularization = 0.8144329896907216
```

In [194]:
```
Confusion_Matrix2 = confusion_matrix(test_labels_tfidf_w2v, prediction2)
print("Confusion Matrix on L1 regularization \n"+str(Confusion_Matrix2))
```

```
Confusion Matrix on L1 regularization
[[412  90]
 [103 395]]
```

In [195]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_tfidf_w2v, prediction2).ravel()
tn, fp, fn, tp
```

Out[195]:
```
(412, 90, 103, 395)
```

Task 4. Use L1 regularization for different values of λ and report error and sparsity for each value of λ.

In [196]:
```
model3 = LogisticRegression(penalty="l1", C = 10000)

model3.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

accuracyScore_10000 = model3.score(test_tfidf_w2v, test_labels_tfidf_w2v)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.0001 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.0001 = "+str(error))
```

```
Number of non-zero for λ value of 0.0001 = 300
Error for λ value of 0.0001 = 0.18799999999999994
```

In [197]:
```
model3 = LogisticRegression(penalty="l1", C = 100)

model3.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

accuracyScore_10000 = model3.score(test_tfidf_w2v, test_labels_tfidf_w2v)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.01 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.01 = "+str(error))
```

```
Number of non-zero for λ value of 0.0001 = 292
Error for λ value of 0.01 = 0.19099999999999995
```

In [198]:
```
model3 = LogisticRegression(penalty="l1", C = 1)

model3.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

accuracyScore_10000 = model3.score(test_tfidf_w2v, test_labels_tfidf_w2v)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 1 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 1 = "+str(error))
```

```
Number of non-zero for λ value of 0.0001 = 85
Error for λ value of 1 = 0.21799999999999997
```

In [199]:
```
model3 = LogisticRegression(penalty="l1", C = 10**-3)

model3.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

accuracyScore_10000 = model3.score(test_tfidf_w2v, test_labels_tfidf_w2v)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 1000 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 1000 = "+str(error))
```

```
Number of non-zero for λ value of 0.0001 = 0
Error for λ value of 1000 = 0.498
```

Task 5. Check for multi-collinearity of features and find top-10 most important features

In [200]:
```
# computing weight vector prior to adding noise in out data
model4 = LogisticRegression(penalty="l2", C = FinalHP)

model4.fit(train_tfidf_w2v, train_labels_tfidf_w2v)

weightVector1 = model4.coef_
```

In [202]:
```
noise = np.random.normal(loc=0.1, scale=0.1) # generating random positive number from normal distribution
noise_matrix = np.full((4000, 300), noise) # generating matrix of size 4000 * 300 where every number in matrix is 'noise'
dataWithNoise_tfidf_AvgW2V = noise_matrix + train_tfidf_w2v # adding noise to training data fro pertubation test

print(noise)
print(dataWithNoise_tfidf_AvgW2V.shape)
```

```
0.02300070878465929
(4000, 300)
```

In [203]:
```
# computing weight vector after adding noise in out data
model5 = LogisticRegression(penalty="l2", C = FinalHP)

model5.fit(dataWithNoise_tfidf_AvgW2V, train_labels_tfidf_w2v)

weightVector2 = model5.coef_
```

In [204]:
```
differenceWeights = weightVector1 - weightVector2
differenceWeights.ravel()
```

Out[204]:
```
array([-1.29288568e-02,  7.05277795e-03, -1.42343271e-03,  7.95322385e-03,
       -1.16889928e-02,  5.90871933e-03, -6.26136071e-03, -9.31689173e-03,
        1.33540254e-03, -5.96326251e-03,  6.90715865e-03, -3.68980884e-03,
        4.37822680e-02,  1.71258714e-02,  1.63797423e-02,  9.53680248e-03,
        2.97579528e-02,  9.21537063e-03,  8.74318400e-04,  2.27536772e-02,
        1.24565633e-02,  6.22320722e-03,  8.98027940e-03,  3.39631451e-02,
       -8.69922080e-03, -9.38652612e-03,  7.88057618e-03, -1.80001347e-02,
        9.52894852e-03,  1.67691780e-03, -2.23035889e-02,  4.44436800e-02,
        1.04282626e-02,  1.67814503e-02,  2.65462036e-02,  2.49449828e-02,
        7.46989681e-03,  1.46980704e-02,  3.44653723e-02,  1.17297600e-02,
       -3.59700228e-03,  1.89764839e-02, -7.86729323e-03, -4.28927479e-03,
        1.27985915e-02,  1.54100148e-02,  1.46615161e-02,  2.50155434e-02,
        4.39080479e-02, -6.89995646e-03,  7.89254100e-04,  1.89435706e-02,
        1.37793028e-02,  2.69557685e-02,  2.01593807e-02,  3.30413386e-02,
        2.20252252e-03,  5.78021432e-05,  2.00581838e-02,  2.94766449e-02,
        9.59486874e-03,  1.98451434e-02,  2.41118813e-02,  2.69028426e-02,
        1.75097725e-02,  1.48005297e-02,  1.76063979e-02,  2.52302641e-02,
        8.60402641e-03, -3.72362313e-02,  2.76334107e-02,  2.53467129e-02,
        2.79811449e-02,  1.52795109e-02,  2.31202846e-02,  3.43281479e-02,
        3.57026673e-02,  1.89714313e-02,  6.16142014e-03,  2.56034684e-03,
        4.27175817e-02,  4.28640429e-03,  3.48918707e-02, -2.21128580e-03,
        7.79310586e-03, -8.09082661e-03,  1.03958875e-03,  1.14192952e-02,
       -1.74228144e-03, -6.72781671e-03,  2.42709449e-03,  3.34218822e-02,
        4.58764657e-02, -4.47550726e-03,  1.22148027e-02,  4.08199037e-02,
       -1.09994975e-02, -5.44970413e-03,  1.76397208e-02,  3.18352971e-02,
        3.68297402e-03,  5.50001594e-03,  1.25791441e-02,  8.66721677e-03,
        6.41165524e-04,  3.52191406e-03,  1.15536786e-02, -4.42487278e-03,
       -2.44744001e-03,  8.83058879e-03,  2.04647997e-02,  2.77297122e-02,
        2.10541433e-02,  1.47236151e-02,  2.05876896e-02,  3.45681104e-02,
        2.45070403e-02,  3.49042063e-02,  2.74687251e-02, -7.62727467e-03,
       -1.50394138e-02,  5.09583659e-03,  1.50386606e-03,  2.54651849e-02,
       -9.14386708e-04,  2.04366963e-02, -3.74593228e-03,  2.37375830e-02,
        7.00176793e-03, -7.94990824e-03,  9.30414135e-03,  1.25059876e-02,
        1.82125249e-02,  1.40370479e-03,  3.32188113e-02,  7.41891723e-03,
        7.90949350e-03,  1.69666236e-02,  7.19428450e-03,  1.10829523e-02,
        4.30086144e-03, -4.26823843e-03,  2.36707200e-02, -3.54514201e-03,
       -7.36973871e-04, -1.47893098e-03,  2.67653271e-02, -4.82599562e-04,
        2.07336855e-03,  7.20284339e-03,  2.69721139e-02,  1.15745577e-02,
```

```
 2.64788389e-02,  1.96452302e-03, -9.75064695e-04,  2.39540152e-02,
 3.00248281e-02,  4.35006866e-03,  2.86924479e-05,  1.69789897e-02,
-3.18299133e-03,  4.37864797e-02,  1.91727608e-02,  3.11715949e-03,
 7.50710607e-03,  2.69576940e-03,  1.32698988e-02,  4.71260076e-03,
 1.29500018e-02, -1.08379866e-02,  1.49851571e-02,  1.41275614e-03,
-2.22084554e-03, -1.89911698e-02,  4.03563058e-02,  4.40112532e-02,
 2.89244558e-02,  6.79338767e-04,  1.22537447e-02,  1.90237065e-02,
 3.29933441e-02,  7.09342272e-04, -7.18983067e-03,  1.39917504e-02,
 2.26315691e-02, -2.35455597e-02,  1.42681828e-02,  2.38999577e-03,
-2.62687828e-03,  2.86301355e-02,  4.39345616e-03,  1.78175563e-02,
-2.38391680e-02, -2.22888031e-02, -9.03474122e-03,  1.69955888e-02,
-8.63809845e-03,  1.20604640e-02,  1.65870950e-02,  2.26401444e-02,
 5.48320031e-03,  1.05104456e-02,  1.45774871e-02, -5.22714896e-03,
 3.66872573e-02,  3.29730924e-02,  3.55197654e-02, -7.82809862e-04,
 2.37766482e-02,  3.47782804e-02,  2.63046155e-02,  2.64243203e-03,
 3.91315329e-02,  6.79380420e-03,  8.13766531e-03,  2.67886226e-02,
 3.44584033e-02, -1.45107355e-03,  1.58441048e-02,  1.34140813e-02,
 2.48469657e-02,  1.39581086e-03,  3.96359961e-02,  6.33467028e-02,
-2.35697745e-03,  1.12049581e-02,  4.23534138e-03,  1.18595201e-03,
 3.49489466e-02, -3.85972210e-03,  2.19280836e-02, -1.01796036e-02,
 3.42379426e-03,  6.33040974e-03,  3.33812137e-03, -1.53450058e-02,
-1.50509670e-02,  5.34299687e-03, -1.90178296e-02,  2.98581962e-02,
 8.66443158e-03,  2.26125595e-02,  3.32108769e-02,  8.68890498e-03,
 5.21294282e-02,  9.44983688e-04,  6.32327061e-03,  7.01957867e-03,
 5.29718837e-03,  4.30849944e-02, -2.88021517e-03,  3.95739857e-03,
 1.54274143e-02,  2.29668525e-02,  2.34616817e-02,  3.50075985e-02,
 2.08920100e-02,  8.10276761e-03,  2.34019137e-02,  2.99366472e-02,
 1.86114426e-02,  3.44886199e-02,  1.99063303e-02, -1.88186501e-02,
-8.07823638e-03,  8.39836538e-03,  1.92440120e-02,  2.29446310e-02,
 1.78121544e-02,  1.40858703e-02,  6.81572286e-04,  1.93565151e-02,
 1.86555689e-02,  1.08613608e-02,  2.56613710e-02, -7.10438671e-03,
-4.56741674e-03, -1.82289501e-03, -4.38715820e-03,  1.73233933e-02,
-1.77129238e-02,  3.09385956e-02,  7.32300567e-03,  1.61863662e-02,
 1.62363847e-02, -1.26858917e-02, -3.46642531e-03,  1.45849349e-02,
 6.42539473e-03,  1.65615186e-02,  1.08547807e-02, -8.76500504e-03,
 1.61640498e-02, -4.48148748e-03, -1.71698476e-02,  3.84135142e-02,
-1.43176288e-02,  1.02955205e-02,  1.60826714e-02,  3.94640406e-02])
```

Here, as you can see that the differnce in the weight vectors prior and after adding noise is high. It means that features are collinear, hence we cannot use weight vectors as feature importance. Since word to vec generates vectors for words which are dependent on each other owing to their similarity of preserving sementic meaning between similar words. Therefore, features in word 2 vec are collinear

GloVe

In [205]:
```
def cleanhtml(sentence): #function to clean htmltags
    cleanr = re.compile("<.*?>")
    cleantext = re.sub(cleanr, " ", sentence)
    return cleantext

def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [206]:
```
#removing HTML tags and punctuation from our text

i = 0
final_string = []
s = ""
for sentence in data["Text"].values:
    filteredSentence = []
    EachReviewText = ""
    sentenceHTMLCleaned = cleanhtml(sentence)
    for eachWord in sentenceHTMLCleaned.split():
        for sentencePunctCleaned in cleanpunc(eachWord).split():
            if((sentencePunctCleaned.isalpha()) & (len(sentencePunctCleaned)>2)):
                sentenceLower = sentencePunctCleaned.lower()
                filteredSentence.append(sentenceLower)

    EachReviewText = ' '.join(filteredSentence)
    final_string.append(EachReviewText)
```

In [207]:
```
data["ProcessedText2"] = final_string
```

In [208]:
```
data.head()
```

Out[208]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I b seve V ca
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	-1	1346976000	Not as Advertised	Pr a label J S Pea
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Thi confe tha aro
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	taff
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	1	1342051200	Nice Taffy	I wil fo or tl

In [209]:
```
allPositiveReviews2 = data[(data["Score"] == 1)]
```

In [210]:
```
allPositiveReviews2.shape
```
Out[210]:
```
(307061, 13)
```

In [211]:
```
positiveReviews2_500 = allPositiveReviews2[:500]
```

In [212]:
```
positiveReviews2_500.shape
```
Out[212]:
```
(500, 13)
```

In [213]:
```
positiveReviews2_500.head()
```

Out[213]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I bo seve V ca
2	2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Thi confe tha arou
3	4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	(taffy I T v
4	5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	1	1342051200	Nice Taffy	I wil for orc th
5	6	7	B006K2ZZ7K	A1SP2KVKFXXRU1	David C. Sullivan	0	0	1	1340150400	Great! Just as good as the expensive brands!	salty taffy fl and v

In [214]:
```
allNegativeReviews2 = data[(data["Score"] == -1)]
```

In [215]:
```
allNegativeReviews2.shape
```
Out[215]:
```
(57110, 13)
```

In [216]:
```
negativeReviews2_500 = allNegativeReviews2[:500]
```

In [217]:
```
negativeReviews2_500.shape
```
Out[217]:
```
(500, 13)
```

In [218]:
```
negativeReviews2_500.head()
```

Out[218]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	T
1	1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	-1	1346976000	Not as Advertised	Prod arri labe Jun Sal Peanu
11	12	13	B0009XLVG0	A327PCT23YH90	LT	1	1	-1	1339545600	My Cats Are Not Fans of the New Food	My c ha be hap eat Felic Pla
15	16	17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	0	-1	1348099200	poor taste	I l eat th and tl are gc
25	26	27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	1	-1	1332633600	Nasty No flavor	watc - cand just re No fla . J pla
45	47	51	B001EO5QW8	A108P30XVUFKXY	Roberto A	0	7	-1	1203379200	Don't like it	T oatm is good. mus so

In [219]:
```
frames2_1000 = [positiveReviews2_500, negativeReviews2_500]
```

In [220]:
```
FinalPositiveNegative2 = pd.concat(frames2_1000)
```

In [221]:
```
FinalPositiveNegative2.shape
```

Out[221]:
```
(1000, 13)
```

In [222]:
```
#Sorting FinalDataframe by "Time"
FinalSortedPositiveNegative2_1000 = FinalPositiveNegative2.sort_values('Time', axis=0, ascending=True, inplace=False)
```

In [223]:
```
FinalSortedPositiveNegativeScore2_1000 = FinalSortedPositiveNegative2_1000["Score"]
```

In [224]:
```
FinalSortedPositiveNegative2_1000.shape
```

Out[224]:
```
(1000, 13)
```

In [225]:
```
FinalSortedPositiveNegativeScore2_1000.shape
```

Out[225]:
```
(1000,)
```

In [226]:
```
Data2 = FinalSortedPositiveNegative2_1000
```

In [227]:
```
Data2_Labels = FinalSortedPositiveNegativeScore2_1000
```

In [228]:
```
print(Data2.shape)
print(Data2_Labels.shape)
```

```
(1000, 13)
(1000,)
```

In [229]:
```
Data2.head()
```

Out[229]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
9	10	11	B0001PB9FE	A3HDKO7OW0QNK4	Canadian Fan	1	1	1	1107820800	Th S
1653	2106	2296	B0001VWE02	AQM74O8Z4FMS0	Sunshine	0	0	-1	1127606400	Belo
2558	3667	3984	B0005ZHWXI	A26HFSVLAGULIM	Heather L. Parisi "Robert and Heather Parisi"	0	1	-1	1131235200	BLUI
1341	1779	1935	B000F4EU52	A2PNOU7NXB1JE4	Peggy "pab920"	14	17	-1	1153008000	In
2307	3362	3661	B000FDKQC4	A1PNP10DP0M7V1	D. Chamberlain "dchamberlain072002"	7	8	-1	1156377600	

In [230]:
```
i = 0
listOfSentences2 = []
for sentence in Data2["ProcessedText2"].values:
    subSentence = []
    for word in sentence.split():
        subSentence.append(word)

    listOfSentences2.append(subSentence)
```

In [231]:
```
print(Data2['ProcessedText2'].values[0])
print("\n")
print(listOfSentences2[0:2])
print("\n")
print(type(listOfSentences2))
print("\n")
print(len(listOfSentences2))
```

```
dont know its the cactus the tequila just the unique combination ingredients but the flavour this hot sauce makes one k
ind picked bottle once trip were and brought back home with and were totally blown away when realized that simply could
nt find anywhere our city were bummed now because the magic the internet have case the sauce and are ecstatic because y
ou love hot sauce mean really love hot sauce but dont want sauce that tastelessly burns your throat grab bottle tequila
picante gourmet inclan just realize that once you taste you will never want use any other sauce thank you for the perso
nal incredible service


[['dont', 'know', 'its', 'the', 'cactus', 'the', 'tequila', 'just', 'the', 'unique', 'combination', 'ingredients', 'bu
t', 'the', 'flavour', 'this', 'hot', 'sauce', 'makes', 'one', 'kind', 'picked', 'bottle', 'once', 'trip', 'were', 'an
d', 'brought', 'back', 'home', 'with', 'and', 'were', 'totally', 'blown', 'away', 'when', 'realized', 'that', 'simply',
'couldnt', 'find', 'anywhere', 'our', 'city', 'were', 'bummed', 'now', 'because', 'the', 'magic', 'the', 'internet', 'h
ave', 'case', 'the', 'sauce', 'and', 'are', 'ecstatic', 'because', 'you', 'love', 'hot', 'sauce', 'mean', 'really', 'lo
ve', 'hot', 'sauce', 'but', 'dont', 'want', 'sauce', 'that', 'tastelessly', 'burns', 'your', 'throat', 'grab', 'bottl
e', 'tequila', 'picante', 'gourmet', 'inclan', 'just', 'realize', 'that', 'once', 'you', 'taste', 'you', 'will', 'neve
r', 'want', 'use', 'any', 'other', 'sauce', 'thank', 'you', 'for', 'the', 'personal', 'incredible', 'service'], ['too',
'much', 'the', 'white', 'pith', 'this', 'orange', 'peel', 'making', 'the', 'product', 'overly', 'bitter', 'and', 'dilut
ing', 'the', 'real', 'good', 'taste', 'the', 'orange', 'zest']]


<class 'list'>


1000
```

In [11]:
```
#Loading pre-trained GloVe vectors
words = pd.read_table("glove.6B.100d.txt", sep=" ", index_col=0, header=None, quoting=csv.QUOTE_NONE)

# Here, We have downloaded pre-trained Glove vectors. You just have to type "Glove word vectors" on google then click on
# "https://nlp.stanford.edu/projects/glove/" link. Then you can download pre-trained word-vectors. Zip file will be
# downloaded, you just have to extract it then load the txt file from extracted folder into ipython notebook using pandas
# just like we have done above.
```

In [233]:
```
def check(word):
    if (words.index == word).any():
        return 1
    else:
        return 0
```

In [234]:
```
# compute average GloVe for each review.
sentenceAsGlove = []
for sentence in listOfSentences2:
    sentenceVector = np.zeros(100)
    TotalWordsPerSentence = 0
    for word in sentence:
        if check(word) == 1:
            vect = words.loc[word]
            sentenceVector += vect
            TotalWordsPerSentence += 1

    sentenceVector /= TotalWordsPerSentence
    sentenceAsGlove.append(sentenceVector)

print(type(sentenceAsGlove))
print(len(sentenceAsGlove))
print(len(sentenceAsGlove[0]))
```

```
<class 'list'>
1000
100
```

In [235]:
```
standardized_Avg_Glove = StandardScaler().fit_transform(sentenceAsGlove)
print(standardized_Avg_Glove.shape)
print(type(standardized_Avg_Glove))
```

```
(1000, 100)
<class 'numpy.ndarray'>
```

Task 1. Split train and test data in a ratio of 80:20.

In [236]:
```
train_glove, test_glove, train_labels_glove, test_labels_glove = train_test_split(standardized_Avg_Glove, Data2_Labels, t
```

In [241]:
```
train_glove.shape, test_glove.shape, train_labels_glove.shape, test_labels_glove.shape
```

Out[241]:
```
((800, 100), (200, 100), (800,), (200,))
```

Task 2. Perform GridSearch Cross Validation and Random Search Cross Validation to find optimal Value of λ.

Grid Search

In [242]:
```
clf = LogisticRegression()

hyper_parameters = [{'C': [10**-3, 10**-2, 10**-1, 10**0, 2, 4, 6, 8, 10**1, 50, 10**2, 500, 10**3]}]
bestCV = GridSearchCV(clf, hyper_parameters, scoring = "accuracy", cv = 5)
bestCV.fit(train_glove, train_labels_glove)

print(bestCV.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='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [243]:
```
best_parameter = bestCV.best_params_
best_parameter["C"]
```

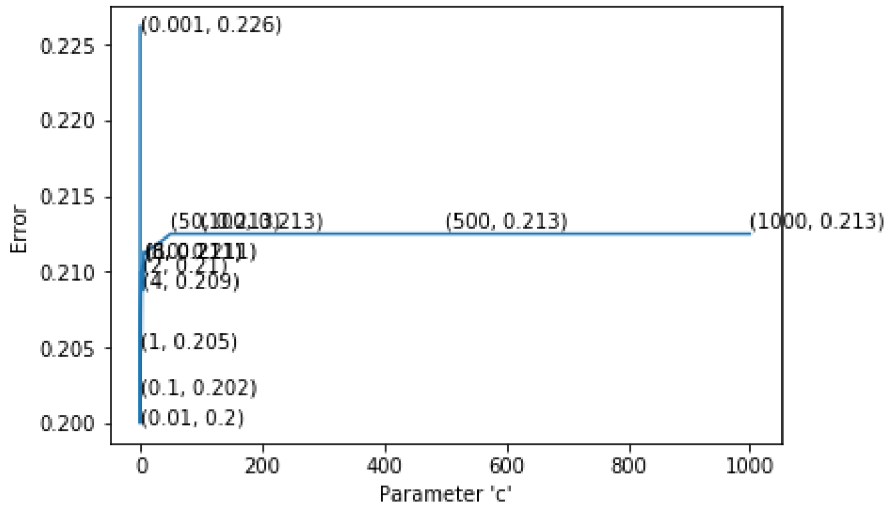
Out[243]:
```
0.01
```

In [244]:
```
scoreData = bestCV.grid_scores_
```

In [245]:
```
scoreData
```

Out[245]:
```
[mean: 0.77375, std: 0.03941, params: {'C': 0.001},
 mean: 0.80000, std: 0.03750, params: {'C': 0.01},
 mean: 0.79750, std: 0.05223, params: {'C': 0.1},
 mean: 0.79500, std: 0.04750, params: {'C': 1},
 mean: 0.79000, std: 0.04947, params: {'C': 2},
 mean: 0.79125, std: 0.05268, params: {'C': 4},
 mean: 0.78875, std: 0.05397, params: {'C': 6},
 mean: 0.78875, std: 0.05397, params: {'C': 8},
 mean: 0.78875, std: 0.05397, params: {'C': 10},
 mean: 0.78750, std: 0.05078, params: {'C': 50},
 mean: 0.78750, std: 0.05078, params: {'C': 100},
 mean: 0.78750, std: 0.05078, params: {'C': 500},
 mean: 0.78750, std: 0.05078, params: {'C': 1000}]
```

In [246]:
```
error = []
parameter = []
for i in range(len(scoreData)):
    error.append(1 - scoreData[i][1])
    parameter.append(scoreData[i][0]["C"])

plot.plot(parameter, np.round(error, 4))
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error1 = []
for e in error:
    error1.append(np.round(e,3))

parameter1 = []
for p in parameter:
    parameter1.append(np.round(p,3))

for xy in zip(parameter1, error1):
    plot.annotate(xy,xy)

plot.show()
```



Random Search

In [258]:
```
n = list(np.random.normal(loc=0.01, scale=0.001, size = 500))  #taking 500 numbers which are distributed normally with me
                                                                #and std-dev = 0.001
```

In [260]:
```
clf = LogisticRegression()

hyper_parameters2 = {'C': n}

bestCV_random = RandomizedSearchCV(clf, hyper_parameters2, scoring = "accuracy", cv = 3)

bestCV_random.fit(train_glove, train_labels_glove)
print(bestCV_random.best_estimator_)
```

```
LogisticRegression(C=0.010864778999588785, 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 [261]:
```
best_random_parameter = bestCV_random.best_params_
best_random_parameter["C"]
```

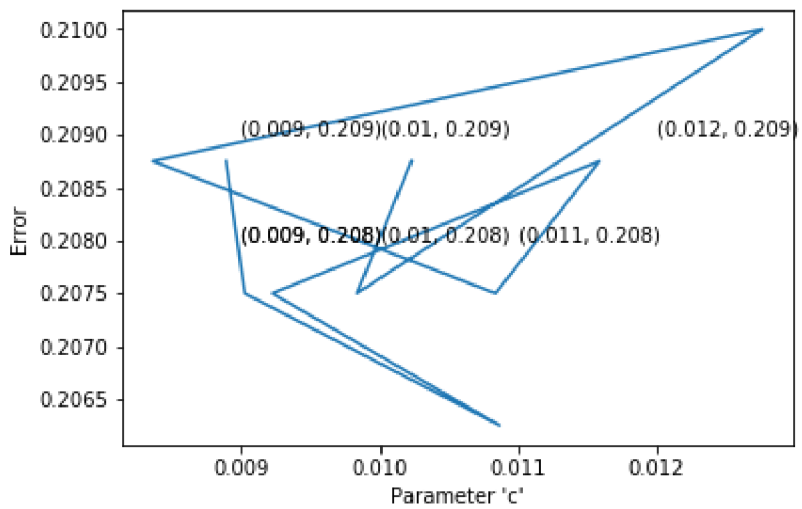
Out[261]:
```
0.010864778999588785
```

In [262]:
```
scoreRandomData = bestCV_random.grid_scores_
```

In [263]:
```
scoreRandomData
```

Out[263]:
```
[mean: 0.79125, std: 0.02455, params: {'C': 0.008893654934541298},
 mean: 0.79250, std: 0.02546, params: {'C': 0.009028580842770980},
 mean: 0.79375, std: 0.02647, params: {'C': 0.010864778999588785},
 mean: 0.79250, std: 0.02546, params: {'C': 0.009228367537127741},
 mean: 0.79125, std: 0.02471, params: {'C': 0.011585912590822901},
 mean: 0.79250, std: 0.02580, params: {'C': 0.010835089892548291},
 mean: 0.79125, std: 0.02455, params: {'C': 0.008366099150893012},
 mean: 0.79000, std: 0.02406, params: {'C': 0.012758200295857164},
 mean: 0.79250, std: 0.02546, params: {'C': 0.009837696631520804},
 mean: 0.79125, std: 0.02471, params: {'C': 0.010230751131592289}]
```

In [264]:
```
error2 = []
parameter2 = []
for i in range(len(scoreRandomData)):
    error2.append(1 - scoreRandomData[i][1])
    parameter2.append(scoreRandomData[i][0]["C"])

plot.plot(parameter2, error2)
plot.xlabel("Parameter 'c'")
plot.ylabel("Error")

error3 = []
for e in error2:
    error3.append(np.round(e,3))

parameter3 = []
for p in parameter2:
    parameter3.append(np.round(p,3))

for xy in zip(parameter3, error3):
        plot.annotate(xy,xy)

plot.show()
```



In [265]:
```
# We are taking our hyper-parameter λ as the average of best λ computed from gridSearchCV and RandomSearchCV
FinalHP = (best_parameter["C"] + best_random_parameter["C"]) / 2
FinalHP
```

Out[265]:
```
0.010432389499794394
```

Task 3. Apply Logistic Regression using both L1 and L2 regularizations and report accuracy.

Implementing L2 Regularization

In [266]:
```
model = LogisticRegression(penalty="l2", C = FinalHP)

model.fit(train_glove, train_labels_glove)

prediction = model.predict(test_glove)

AccuracyScore = accuracy_score(test_labels_glove, prediction) * 100

print("Accuracy score on L2 regularization = "+str(AccuracyScore)+"%")
```

```
Accuracy score on L2 regularization = 79.0%
```

In [267]:
```
Precision = precision_score(test_labels_glove, prediction)
print("Precision Score on L2 regularization = "+str(Precision))
```

```
Precision Score on L2 regularization = 0.8461538461538461
```

In [268]:
```
Recall = recall_score(test_labels_glove, prediction)
print("Recall Score on L2 regularization = "+str(Recall))
```

```
Recall Score on L2 regularization = 0.7333333333333333
```

In [269]:
```
ConfusionMatrix = confusion_matrix(test_labels_glove, prediction)
print("Confusion Matrix on L2 regularization \n= "+str(ConfusionMatrix))
```

```
Confusion Matrix on L2 regularization
= [[81 14]
 [28 77]]
```

In [270]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_glove, prediction).ravel()
tn, fp, fn, tp
```

Out[270]:
```
(81, 14, 28, 77)
```

Implementing L1 Regularization

In [273]:
```
model2 = LogisticRegression(penalty="l1", C = FinalHP)

model2.fit(train_glove, train_labels_glove)

prediction2 = model2.predict(test_glove)

AccuracyScore2 = accuracy_score(test_labels_glove, prediction2) * 100

print("Accuracy score on L1 regularization = "+str(AccuracyScore2)+"%")
```

```
Accuracy score on L1 regularization = 67.5%
```

In [274]:
```
Precision = precision_score(test_labels_glove, prediction2)
print("Precision Score on L1 regularization = "+str(Precision))
```

```
Precision Score on L1 regularization = 0.7222222222222222
```

In [275]:
```
Recall = recall_score(test_labels_glove, prediction2)
print("Recall Score on L1 regularization = "+str(Recall))
```

```
Recall Score on L1 regularization = 0.6190476190476191
```

In [276]:
```
ConfusionMatrix = confusion_matrix(test_labels_glove, prediction2)
print("Confusion Matrix on L1 regularization \n= "+str(ConfusionMatrix))
```

```
Confusion Matrix on L1 regularization
= [[70 25]
 [40 65]]
```

In [277]:
```
tn, fp, fn, tp = confusion_matrix(test_labels_glove, prediction2).ravel()
tn, fp, fn, tp
```

Out[277]:
```
(70, 25, 40, 65)
```

Task 4. Use L1 regularization for different values of λ and report error and sparsity for each value of λ.

In [278]:
```
model3 = LogisticRegression(penalty="l1", C = 1000)

model3.fit(train_glove, train_labels_glove)

accuracyScore_10000 = model3.score(test_glove, test_labels_glove)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.001 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.001 = "+str(error))
```

```
Number of non-zero for λ value of 0.001 = 100
Error for λ value of 0.001 = 0.22999999999999998
```

In [279]:

```
model3 = LogisticRegression(penalty="l1", C = 100)

model3.fit(train_glove, train_labels_glove)

accuracyScore_10000 = model3.score(test_glove, test_labels_glove)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 0.01 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 0.01 = "+str(error))
```

```
Number of non-zero for λ value of 0.01 = 100
Error for λ value of 0.01 = 0.22999999999999998
```

In [280]:

```
model3 = LogisticRegression(penalty="l1", C = 1)

model3.fit(train_glove, train_labels_glove)

accuracyScore_10000 = model3.score(test_glove, test_labels_glove)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 1 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 1 = "+str(error))
```

```
Number of non-zero for λ value of 1 = 92
Error for λ value of 1 = 0.21999999999999997
```

In [281]:
```
model3 = LogisticRegression(penalty="l1", C = 10**-2)

model3.fit(train_glove, train_labels_glove)

accuracyScore_10000 = model3.score(test_glove, test_labels_glove)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 100 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 100 = "+str(error))
```

```
Number of non-zero for λ value of 100 = 2
Error for λ value of 100 = 0.32499999999999996
```

In [282]:
```
model3 = LogisticRegression(penalty="l1", C = 10**-4)

model3.fit(train_glove, train_labels_glove)

accuracyScore_10000 = model3.score(test_glove, test_labels_glove)

error = 1 - accuracyScore_10000

weightVector = model3.coef_

print("Number of non-zero for λ value of 10000 = "+str(np.count_nonzero(weightVector)))

print("Error for λ value of 10000 = "+str(error))
```

```
Number of non-zero for λ value of 10000 = 0
Error for λ value of 10000 = 0.525
```

Task 5. Check for multi-collinearity of features and find top-10 most important features

In [283]:
```
# computing weight vector prior to adding noise in out data
model4 = LogisticRegression(penalty="l2", C = FinalHP)

model4.fit(train_glove, train_labels_glove)

weightVector1 = model4.coef_
```

In [284]:
```
noise = np.random.normal(loc=0.1, scale=0.1) # generating random positive number from normal distribution
noise_matrix = np.full((800, 100), noise) # generating matrix of size 4000 * 300 where every number in matrix is 'n'
dataWithNoise_glove = noise_matrix + train_glove # adding noise to training data fro pertubation test

print(noise)
print(dataWithNoise_glove.shape)
```

```
0.04490121809728769
(800, 100)
```

In [285]:
```
# computing weight vector after adding noise in out data
model5 = LogisticRegression(penalty="l2", C = FinalHP)

model5.fit(dataWithNoise_glove, train_labels_glove)

weightVector2 = model5.coef_
```

In [286]:
```
differenceWeights = weightVector1 - weightVector2
differenceWeights.ravel()
```

Out[286]:
```
array([ 4.23497834e-04,  2.89410219e-04,  3.25740221e-04,  4.29228648e-04,
        1.29199851e-04,  4.13238064e-04,  1.04042606e-04,  4.40570008e-04,
        3.74213476e-04, -2.34256914e-04,  2.38465597e-04,  3.42048111e-04,
        1.81592466e-04, -8.70773117e-06,  4.95984444e-04,  2.35376161e-04,
       -2.25085153e-04, -4.97572930e-05,  4.97980468e-04,  4.66802311e-04,
        5.24820587e-04, -2.03303600e-05,  3.48937061e-04,  3.46031406e-04,
       -2.16840855e-04,  3.03657060e-04,  2.83432580e-04,  1.67697936e-04,
        3.11069402e-04,  1.19079433e-04,  5.04643291e-04,  5.63305232e-04,
        3.32103226e-04, -4.26387743e-05,  2.65642812e-04,  3.00432122e-04,
        4.05883323e-04,  4.93371338e-04,  5.99841619e-04,  5.31515245e-04,
        2.05598990e-04,  3.87127227e-04,  8.49285958e-05,  1.93436503e-04,
        3.16120104e-04,  4.90627566e-04,  5.89163058e-04,  3.25872726e-04,
        2.03056121e-04,  1.03232189e-04,  5.53790632e-04, -5.77459757e-05,
        1.91736610e-04,  2.74947914e-04,  2.42638104e-04,  4.17039003e-04,
        5.65809512e-04,  3.88974375e-04,  6.20380261e-04,  3.05395078e-04,
        3.01788343e-04,  5.03711077e-04,  3.00680830e-04,  2.88001629e-04,
        3.89584425e-04, -7.67251800e-05,  5.02824435e-04,  6.89213136e-04,
        1.21004587e-04,  3.30275712e-04,  3.90439474e-04,  1.00528139e-04,
       -4.83776787e-06,  4.49744316e-04, -1.88515578e-05, -1.32591454e-05,
        6.47118872e-05,  2.40364963e-04,  3.42028491e-04,  1.45423493e-04,
        4.97924174e-04,  1.18295961e-05,  1.79866121e-04,  1.73004848e-04,
        1.16974214e-04,  5.24405118e-04,  7.28710373e-05,  5.28986271e-04,
        5.74490274e-04,  3.02961785e-04,  3.22756243e-04,  9.54011847e-05,
       -1.15529615e-04,  7.08858125e-05,  2.57587742e-04,  4.07674328e-04,
        4.20030843e-04,  4.20636958e-04,  3.17014906e-04,  5.69358505e-04])
```

Here, as you can see that the differnce in the weight vectors prior and after adding noise is high. It means that features are collinear, hence we cannot use weight vectors as feature importance. Since glove generates vectors for words which are dependent on each other owing to their similarity of preserving sementic meaning between similar words. Therefore, features in glove are collinear similar to w2v

Summary

1) Average W2V

1.1) Best Value of hyper-parameter(C) from Grid Search: 1000

1.2) Best Value of hyper-parameter(C) from Random Search: 584.2338

1.3) Accuracy of Logistic Regression on L2 Regularization: 83.6%

1.4) Accuracy of Logistic Regression on L1 Regularization: 83.2%

1.5) L1 regularization for different values of λ and report error and sparsity for each value of λ

1.5.1) Number of non-zero for λ value of 0.0001= 400

Error for λ value of 0.0001= 0.169

1.5.2) Number of non-zero for λ value of 0.001 = 400

Error for λ value of 0.001 = 0.171

1.5.3) Number of non-zero for λ value of 0.1 = 230

Error for λ value of 0.1 = 0.1800

1.5.4) Number of non-zero for λ value of 1000 = 0

Error for λ value of 1000 = 0.498

1.5.5) Number of non-zero for λ value of 10000 = 0

Error for λ value of 10000 = 0.498

2) TFIDF-W2V

2.1) Best Value of hyper-parameter(C) from Grid Search: 100

2.2) Best Value of hyper-parameter(C) from Random Search: 107.5692

2.3) Accuracy of Logistic Regression on L2 Regularization: 80.7%

2.4) Accuracy of Logistic Regression on L1 Regularization: 80.7%

2.5) L1 regularization for different values of λ and report error and sparsity for each value of λ

2.5.1) Number of non-zero for λ value of 0.001 = 300

Error for λ value of 0.001 = 0.18799

2.5.2) Number of non-zero for λ value of 0.01 = 292

Error for λ value of 0.01 = 0.19099

2.5.3) Number of non-zero for λ value of 1 = 85

Error for λ value of 1 = 0.21799

2.5.4) Number of non-zero for λ value of 1000 = 0

Error for λ value of 1000 = 0.498

3) GLoVe(Pre-trained)

3.1) Best Value of hyper-parameter(C) from Grid Search: 0.01

3.2) Best Value of hyper-parameter(C) from Random Search: 0.0108

3.3) Accuracy of Logistic Regression on L2 Regularization: 79.0%

3.4) Accuracy of Logistic Regression on L1 Regularization: 67.5%

3.5) L1 regularization for different values of λ and report error and sparsity for each value of λ

3.5.1) Number of non-zero for λ value of 0.001 = 100

Error for λ value of = 0.2299

3.5.2) Number of non-zero for λ value of = 100

Error for λ value of 0.01 = 0.2299

3.5.3) Number of non-zero for λ value of 1 = 92

Error for λ value of 1 = 0.21999

3.5.4) Number of non-zero for λ value of 100 = 2

Error for λ value of 100 = 0.324

3.5.5) Number of non-zero for λ value of 10000 = 0

Error for λ value of 10000 = 0.525