[1]. Reading Data

Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

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

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

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

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

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is 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 is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
```

```python
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 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 Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

```
paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip in
stall paramiko` to suppress
```

In [2]:

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code
```

```
Enter your authorization code:
··········
Mounted at /content/drive
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('/content/drive/My Drive/Colab Notebooks/databas
e.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 50
0000 data points
# you can change the number to any other number based on your computing
 power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 100000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [0]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text

	UserId	ProductId	ProfileName	Time	Score	Text	(
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]:
```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

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

In [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[10]:
```
(87775, 10)
```

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]:
```
87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)
```

```
#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(87773, 10)
```

Out[14]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)
```

```
sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [16]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
```

```
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
```

```
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [19]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [20]:
```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
```

```
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:29<00:00, 2935.21it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'way hot blood took bite jig lol'
```

In [25]:
```
#here preprocessed_review is my X and final['Score'] is my Y
```

```
print(len(preprocessed_reviews))
print(len(final['Score']))
X=preprocessed_reviews
Y=final['Score']
#if both are of same lenght then proceed....
```

```
87773
87773
```

In [0]:
```
#here i am performing splittig operation as train test and cv...
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33) # this is random splitting
```

In [27]:
```
#checking the types of test and train X,y
print(type(X_train))
print(type(X_test))
print(type(X_cv))
print(type(y_train))
print(type(y_test))
print(type(y_cv))
#now i have xtrain ,xtest,tcv and ytrain,ytest ,ycv....
```

```
<class 'list'>
<class 'list'>
<class 'list'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[4] Featurization

[4.1] BAG OF WORDS

In [28]:

```
#BoW
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fitting on train data ,we cant perform fit on
 test or cv

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
#you can also check X_train_bow is of sparse matrix type or not
#below is code for that
print(type(X_train_bow))
#displaying number of unique words in each of splitted dataset
print("the number of unique words in train: ", X_train_bow.get_shape()[
1])
print("the number of unique words in cv: ", X_cv_bow.get_shape()[1])
print("the number of unique words in test: ", X_test_bow.get_shape()[1
])
```

```
After vectorizations
(39400, 37424) (39400,)
(19407, 37424) (19407,)
(28966, 37424) (28966,)
======================================================================
=============================
<class 'scipy.sparse.csr.csr_matrix'>
the number of unique words in train:  37424
the number of unique words in cv:  37424
the number of unique words in test:  37424
```

[4.3] TF-IDF

In [29]:
```
#below code for converting to tfidf
#i refered sample solution to write this code
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(X_train)
X_test_tf_idf = tf_idf_vect.transform(X_test)
X_cv_tf_idf = tf_idf_vect.transform(X_cv)
print("the type of count vectorizer ",type(X_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_shap
e())
print("the number of unique words including both unigrams and bigrams "
, X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble buy', 'able drink', 'able eat', 'able enjoy', 'able find', 'able fi
nish', 'able get', 'able give']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (39400, 23617)
the number of unique words including both unigrams and bigrams  23617
```

[4.4] Avg W2V

In [30]:
```
#in average w2v the output is of list form and here we write same code
 of all train ,test and cv
#this code is for train data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
```

```
#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
  0%|          | 117/39400 [00:00<00:33, 1168.94it/s]
```

```
number of words that occured minimum 5 times  11967
sample words  ['quality', 'popcorn', 'excellent', 'beware', 'hot', 'lik
```

```
e', 'good', 'product', 'purchase', 'bought', 'keurig', 'times', 'singl
e', 'cup', 'coffee', 'needed', 'not', 'anticipated', 'husband', 'underw
helmed', 'assortment', 'coffees', 'came', 'brewer', 'found', 'double',
'black', 'diamond', 'extra', 'bold', 'used', 'afternoon', 'breaks', 'fe
el', 'investment', 'headed', 'yard', 'sale', 'shop', 'anytime', 'soon',
'lover', 'stronger', 'premium', 'may', 'exactly', 'looking', 'pot', 'si
mply', 'much']
```

```
100%|██████████| 39400/39400 [00:56<00:00, 695.43it/s]
```

```
(39400, 50)
[-0.57792841  0.24822368 -0.47830741  1.0136394  -0.2020394   0.4599398
9
 -1.02915884 -0.25225687 -0.69304183 -0.0513287   0.15109402 -0.5402475
4
 -0.57653417 -0.32070273 -0.54617336  0.09290533 -0.32168681  0.3667075
7
  0.17359612  0.01340312  0.42120196  1.50718632 -0.09189347 -0.6075458
1
 -0.23279045  0.96824186  1.15993677  0.55685158  1.5698531  -0.2032167
2
  0.52190766  0.21814158  1.13853605 -0.39497199  1.41759514 -1.0487864
1
  0.70962729 -0.72465516  0.42106878 -0.92444248 -0.39801354 -1.3219260
1
  1.19290768  0.2170359  -0.4161275  -0.21898747 -0.38539314  0.5206012
8
 -0.52643298 -0.42255851]
```

In [31]:
```
#this code is for test data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
```

```
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_test,min_count=5,size=50, workers=4
)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is store
d in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
  1%|          | 147/28966 [00:00<00:19, 1453.87it/s]
```

```
number of words that occured minimum 5 times  10405
sample words  ['think', 'nacho', 'doritos', 'favorite', 'snack', 'chi
p', 'category', 'mini', 'bags', 'perfect', 'not', 'risk', 'large', 'siz
e', 'bag', 'going', 'stale', 'also', 'great', 'lunches', 'snacking', 'a
nytime', 'delicious', 'cookies', 'course', 'rich', 'buttery', 'like',
'convenience', 'packaging', 'calories', 'per', 'cookie', 'keep', 'tryin
```

```
g', 'eat', 'one', 'time', 'good', 'usually', 'least', 'eating', 'whol
e', 'box', 'auto', 'deliver', 'high', 'quality', 'hand', 'guests']
```

```
100%|██████████| 28966/28966 [00:38<00:00, 758.48it/s]
```

```
(28966, 50)
[ 0.11269041 -0.07546746 -0.47505313  0.42356087 -0.03596501  0.6298474
2
 -0.62578878 -0.62502887  0.30887511  0.01481705 -0.81577246  0.0703703
8
 -1.01709111  0.19745413  0.68101989 -0.07767615 -0.19559941  0.5180739
2
  1.09597392  0.17568543  0.11004021  0.06099144 -0.02219731 -0.0919348
5
  0.18880497  0.6461248   0.71002186  0.4126379  -0.1828041  -0.5787147
  0.39352248  0.44266268  0.71028393  0.61044518  0.73341377 -0.2216822
9
 -0.27216675  0.53236457  0.22475001 -0.66990467  0.14470425 -0.4039793
3
  0.53711633 -0.11579637 -0.05758427 -0.04078682 -0.32371378 -0.1612029
5
 -0.40681199 -0.61908097]
```

In [32]:

```
#this code is for cv data:
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())


#training word2vect model
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_cv,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
#this is the actuall code to convert word2vect to avg w2v:
from tqdm import tqdm
import numpy as np
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored
 in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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_cv.append(sent_vec)
sent_vectors_cv= np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
  1%|          | 109/19407 [00:00<00:17, 1073.59it/s]
```

```
number of words that occured minimum 5 times  8583
sample words  ['skeptical', 'get', 'rid', 'tartar', 'year', 'old', 'gol
den', 'retriever', 'first', 'loves', 'treat', 'giving', 'weeks', 'proba
bly', 'every', 'day', 'noticed', 'significant', 'reduction', 'build',
'wow', 'excited', 'product', 'like', 'gave', 'kids', 'babies', 'wante
d', 'biscuit', 'broken', 'not', 'useful', 'baby', 'pieces', 'small', 'h
old', 'returned', 'another', 'spoke', 'representative', 'shipping', 'ne
w', 'asap', 'thank', 'prompt', 'challenge', 'terrier', 'figured', 'no',
'time']
```

```
100%|██████████| 19407/19407 [00:23<00:00, 835.56it/s]
```

```
(19407, 50)
[-0.75811055 -0.26502779 -0.01489491 -0.03985056 -0.591609    0.6184510
```

```
4
 -0.25893702 -0.47912341 -0.21117713 -0.16980615  0.11693896  0.5059156
8
 -0.81199563 -0.51764701  0.39599331  0.08734107 -0.54903274 -0.4681612
  0.71957166 -0.60674486  0.19079661 -0.32713279  0.64426243 -0.1443584
1
  0.16644702  0.16627813 -0.17714299  0.10605373 -0.16242047  0.1045033
4
 -0.16752101 -0.13078239 -0.20623839  0.63714965  0.666589   -0.4493365
6
 -0.45000468  0.44354152  0.32085629 -0.00771956  0.56130059  0.6945729
 -0.41378955  0.40212434 -0.14464418  0.34317935  0.04923523  0.1833318
 -0.72303815 -0.67807692]
```

[4.4.1] TFIDF-W2V

In [33]:

```
#this is for train data
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())


# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# 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 ce
ll_val = tfidf
```

```
tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_train): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            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_train.append(sent_vec)
    row += 1
tfidf_sent_vectors_train= np.array(sent_vectors_train)
print(tfidf_sent_vectors_train.shape)
print(tfidf_sent_vectors_train[0])
```

```
100%|██████████| 39400/39400 [10:41<00:00, 61.40it/s]
```

```
(39400, 50)
[-0.57792841  0.24822368 -0.47830741  1.0136394  -0.2020394   0.4599398
9
 -1.02915884 -0.25225687 -0.69304183 -0.0513287   0.15109402 -0.5402475
4
 -0.57653417 -0.32070273 -0.54617336  0.09290533 -0.32168681  0.3667075
7
  0.17359612  0.01340312  0.42120196  1.50718632 -0.09189347 -0.6075458
1
 -0.23279045  0.96824186  1.15993677  0.55685158  1.5698531  -0.2032167
2
  0.52190766  0.21814158  1.13853605 -0.39497199  1.41759514 -1.0487864
1
```

```
1
  0.70962729 -0.72465516  0.42106878 -0.92444248 -0.39801354 -1.3219260
1
  1.19290768  0.2170359  -0.4161275  -0.21898747 -0.38539314  0.5206012
8
 -0.52643298 -0.42255851]
```

In [34]:
```
#this is for test data
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())


# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_test)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# 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 ce
ll_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sentance_test): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            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_test.append(sent_vec)
    row += 1
tfidf_sent_vectors_test= np.array(sent_vectors_test)
print(tfidf_sent_vectors_test.shape)
print(tfidf_sent_vectors_test[0])
```

```
100%|██████████| 28966/28966 [06:51<00:00, 69.69it/s]
```

```
(28966, 50)
[ 0.11269041 -0.07546746 -0.47505313  0.42356087 -0.03596501  0.6298474
2
 -0.62578878 -0.62502887  0.30887511  0.01481705 -0.81577246  0.0703703
8
 -1.01709111  0.19745413  0.68101989 -0.07767615 -0.19559941  0.5180739
2
  1.09597392  0.17568543  0.11004021  0.06099144 -0.02219731 -0.0919348
5
  0.18880497  0.6461248   0.71002186  0.4126379  -0.1828041  -0.5787147
  0.39352248  0.44266268  0.71028393  0.61044518  0.73341377 -0.2216822
9
 -0.27216675  0.53236457  0.22475001 -0.66990467  0.14470425 -0.4039793
3
  0.53711633 -0.11579637 -0.05758427 -0.04078682 -0.32371378 -0.1612029
5
 -0.40681199 -0.61908097]
```

In [35]:

```
#this is for cv data
i=0
list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
```

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_cv)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# 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 ce
ll_val = tfidf

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            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_cv.append(sent_vec)
    row += 1
tfidf_sent_vectors_cv= np.array(sent_vectors_cv)
```

```
print(tfidf_sent_vectors_cv.shape)
print(tfidf_sent_vectors_cv[0])
```

```
100%|██████████| 19407/19407 [04:00<00:00, 80.56it/s]
```

```
(19407, 50)
[-0.75811055 -0.26502779 -0.01489491 -0.03985056 -0.591609    0.6184510
4
 -0.25893702 -0.47912341 -0.21117713 -0.16980615  0.11693896  0.5059156
8
 -0.81199563 -0.51764701  0.39599331  0.08734107 -0.54903274 -0.4681612
  0.71957166 -0.60674486  0.19079661 -0.32713279  0.64426243 -0.1443584
1
  0.16644702  0.16627813 -0.17714299  0.10605373 -0.16242047  0.1045033
4
 -0.16752101 -0.13078239 -0.20623839  0.63714965  0.666589   -0.4493365
6
 -0.45000468  0.44354152  0.32085629 -0.00771956  0.56130059  0.6945729
 -0.41378955  0.40212434 -0.14464418  0.34317935  0.04923523  0.1833318
 -0.72303815 -0.67807692]
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

    - Find the best hyper parameter which will give the maximum AUC value

- Find the best hyper paramter using k-fold cross validation or simple cross validation data
- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Pertubation Test**

- Get the weights W after fit your model with the data X i.e Train data.
- Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
- Fit the model again on data X' and get the weights W'
- Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
- Now find the % change between W and W' (| (W-W') / (W) |)*100)
- Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
- Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
- Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

- Calculate sparsity on weight vector obtained after using L1 regularization

NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

- Get top 10 important features for both positive and negative classes separately.

6. **Feature engineering**

    - To increase the performance of your model, you can also experiment with with feature engineering like :
        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.

7. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the [confusion matrix] with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps.]

8. **[Conclusion]**

    - [You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link]

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this [link.]

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [36]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve, GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(X_train_bow, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9503523123025837
```

[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [37]:
```
import numpy as np

clf = LogisticRegression(C=1, penalty='l1');
clf.fit(X_train_bow, y_train);
w = clf.coef_
total=37542
print(w.shape)
z=total-np.count_nonzero(w)
print("number of zeros:",z)
print("sparsity in percentage:",(z/total)*100)
```

```
(1, 37424)
number of zeros: 34046
sparsity in percentage: 90.68776303872995
```

[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [38]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(X_train_bow, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
```

```
          tol=0.0001, verbose=0, warm_start=False)
0.9502288097841493
```

[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

In [0]:
```
X_train_bow.dtype=np.float64
```

In [0]:
```
X_train_bow.data+=0.001
```

In [41]:
```
# import copy
# X_train_bow_copy=copy.deepcopy(X_train_bow)
# e = np.random.normal(0,0.1)
# print(e)
# np.add(X_train_bow_copy.data,e,out=X_train_bow_copy.data,casting='uns
afe')
pertubated_model=LogisticRegression(C=1,penalty='l2')
pertubated_model.fit(X_train_bow,y_train)
```

Out[41]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [42]:
```
w_after_pertubated=pertubated_model.coef_
print(w_after_pertubated.shape)
print(w.shape)
print(w_after_pertubated[0,0:5])
```

```
(1, 37424)
(1, 37424)
[ 0.00046165  0.00048714  0.00016237 -0.00083695  0.00016237]
```

In [0]:
```
w=w+10**-6
w_after_pertubated=w_after_pertubated+10**-6
```

In [44]:
```
print(w[:5])
print(w_after_pertubated[:5])
```

```
[[1.e-06 1.e-06 1.e-06 ... 1.e-06 1.e-06 1.e-06]]
[[0.00046265 0.00048814 0.00016337 ... 0.00016343 0.00016337 0.0001633
3]]
```

In [0]:
```
a=(abs((w-w_after_pertubated)/(w)))*100
```

In [46]:
```
print(type(a))
print(a.shape)
print(len(vectorizer.get_feature_names()))
```

```
<class 'numpy.ndarray'>
(1, 37424)
37424
```

In [47]:
```
a=a.T
print(a.shape)
print()
```

```
(37424, 1)
```

In [48]:
```
b=np.argsort(a.T)
b
```

Out[48]:
```
array([[11326, 25613, 19806, ..., 28984, 23429, 26615]])
```

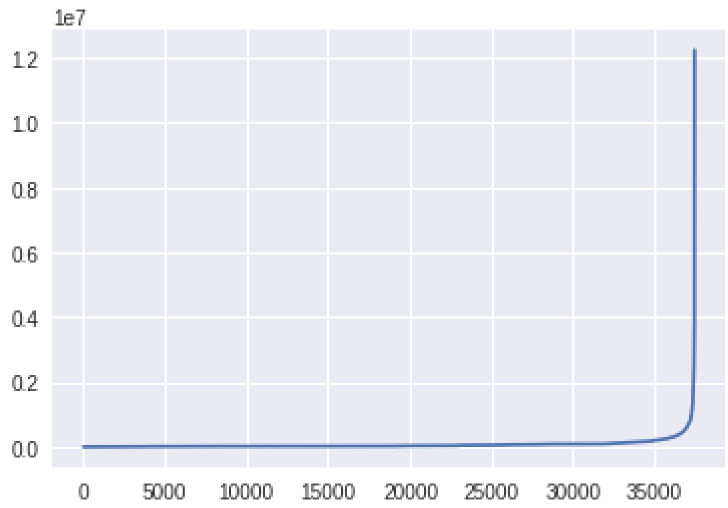
In [0]:
```
c = sorted(a)
```

In [50]:
```
for i in range(0,101,10):
  print(np.percentile(c, i))
```

```
0.43872946083630215
2532.2707240839713
16230.005991281168
16236.794447225426
```

```
16243.207248752085
18730.299944190447
34964.394787322846
66099.81514061615
83691.32166888344
143655.10805263463
12243853.01721977
```

In [51]:
```
%matplotlib inline
plt.plot(list(range(len(vectorizer.get_feature_names()))), c, label='si
mple')
plt.show()
```



now we got the thresold value i.e 37400

total number of features are 37424 lets print 24 features which are multicollinear

In [70]:
```
d=list(range(len(vectorizer.get_feature_names())))
print(len(d))
```

```
37424
```

In [0]:
```
e=[]
for i in d:
  if(i>37400):
    e.append(i)
```

In [75]:
```
e
```

Out[75]:
```
[37401,
 37402,
 37403,
 37404,
 37405,
 37406,
 37407,
 37408,
 37409,
 37410,
 37411,
 37412,
 37413,
 37414,
 37415,
 37416,
 37417,
 37418,
 37419,
 37420,
 37421,
 37422,
 37423]
```

In [76]:
```
#these are the multi collinear features
print(np.take(vectorizer.get_feature_names(),e))
```

```
['zoomies' 'zooming' 'zoos' 'zotz' 'zours' 'zp' 'zreport' 'zsweet'
 'zucchini' 'zuchinni' 'zucini' 'zucs' 'zuk' 'zuke' 'zukes' 'zulu' 'zum
a'
 'zumba' 'zwieback' 'zx' 'zymox' 'zz' 'zzzzzzzzzz']
```

[5.1.3] Feature Importance on BOW, SET 1

[5.1.3.1] Top 10 important features of positive class from SET 1

In [56]:
```
w=clf.coef_
w=np.sort(w)
#print(w[0,-10:])
print(np.take(vectorizer.get_feature_names(),[3.29392218,2.98567319,2.78238019,2.73909904,2.63102463,2.61490743,2.58708351,2.58223035,2.5661904,2.52260501]))
```

```
['aaaaaaarrrrrggghhh' 'aaaa' 'aaaa' 'aaaa' 'aaaa' 'aaaa' 'aaaa' 'aaaa'
 'aaaa' 'aaaa']
```

[5.1.3.2] Top 10 important features of negative class from SET 1

In [57]:
```
w=np.sort(w)
#print(w[0,0:10])
print(np.take(vectorizer.get_feature_names(),[-3.82713768,-3.79596475,-3.78992683,-3.57587919,-3.51745129,-3.41489179,-3.13466695,-3.02036463,-2.96546889,-2.94899787]))
```

```
['zymox' 'zymox' 'zymox' 'zymox' 'zymox' 'zymox' 'zymox' 'zymox' 'zz'
'zz']
```

[5.2] Logistic Regression on TFIDF, SET 2

[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [58]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(X_train_tf_idf, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9543015059275872
```

[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [59]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(X_train_tf_idf, y_train)
```

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

```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=
True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9541646331543842
```

[5.2.3] Feature Importance on TFIDF, SET 2

[5.2.3.1] Top 10 important features of positive class from SET 2

In [60]:

```
clf = LogisticRegression(C=1, penalty='l1');
clf.fit(X_train_tf_idf, y_train);
w = clf.coef_
w=np.sort(w)
#print(w[0,-10:])
print(np.take(vectorizer.get_feature_names(),[18.11710507,17.7122167,1
5.84570687,14.86040435,13.97212241,11.88554852,10.82781873,10.82120392,
10.72653269,10.63936001]))
```

```
['abates' 'abandoned' 'aback' 'ab' 'aasanfood' 'aap' 'aamazon' 'aamazo
n'
 'aamazon' 'aamazon']
```

[5.2.3.2] Top 10 important features of negative class from SET 2

In [61]:

```
w=np.sort(w)
#print(w[0,0:10])
print(np.take(vectorizer.get_feature_names(),[-14.83761484,-14.42001353
,-14.23921853,-14.22187933,-12.47863029,-11.79524799,-11.58817646,-11.3
451241,-11.24989881,-10.90499686]))
```

```
['zuchinni' 'zuchinni' 'zuchinni' 'zuchinni' 'zucs' 'zuk' 'zuk' 'zuk'
```

```
'zuk' 'zuke']
```

[5.3] Logistic Regression on AVG W2V, SET 3

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

In [62]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(sent_vectors_train, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9188367230564531
```

[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [63]:

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

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(sent_vectors_train, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9190083273128257
```

[5.4] Logistic Regression on TFIDF W2V, SET 4

[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

In [64]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
```

```
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(tfidf_sent_vectors_train, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
0.9188367230564531
```

[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [77]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


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


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters
, scoring = 'f1', cv=5)
model.fit(tfidf_sent_vectors_train, y_train)

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

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
```

```
             tol=0.0001, verbose=0, warm_start=False)
0.9190083273128257
```

[6] Conclusions

In [79]:

```python
# creating
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Regularizer", "score"]
x.add_row(["BoW", "L1", 0.9503])
x.add_row(["tfidf", "L1", 0.9543])
x.add_row(["avg w2v", "L1", 0.9188])
x.add_row(["tfidfw2v", "L1", 0.9188])
print(x)
print("-----------------------------------------------------------------------
------------------")
y = PrettyTable()
y.field_names = ["Vectorizer", "Regularizer", "score"]
y.add_row(["BoW", "L2",0.9502])
y.add_row(["tfidf", "L2",0.9541])
y.add_row(["avg w2v", "L2",0.9190])
y.add_row(["tfidf", "L2",0.9190])
print(y)
```

```
+------------+-------------+--------+
| Vectorizer | Regularizer | score  |
+------------+-------------+--------+
|    BoW     |      L1     | 0.9503 |
|   tfidf    |      L1     | 0.9543 |
|  avg w2v   |      L1     | 0.9188 |
|  tfidfw2v  |      L1     | 0.9188 |
+------------+-------------+--------+
-----------------------------------------------------------------------
-----------
+------------+-------------+--------+
| Vectorizer | Regularizer | score  |
+------------+-------------+--------+
```

```
|     BoW      |      L2      | 0.9502 |
|    tfidf     |      L2      | 0.9541 |
|   avg w2v    |      L2      | 0.919  |
|    tfidf     |      L2      | 0.919  |
+--------------+--------------+--------+
```