

Amazon Fine Food Reviews Analysis

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.Id
- 2.ProductId - unique identifier for the product
- 3.UserId - unique 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).

1. Import required libraries

```
In [32]: 1 import warnings
          2 warnings.filterwarnings("ignore")
```

```
In [33]: 1 %matplotlib inline
2
3 import sqlite3
4 import pandas as pd
5 import numpy as np
6 import nltk
7 import string
8 import matplotlib.pyplot as plt
9 import seaborn as sns
10 from sklearn.feature_extraction.text import TfidfTransformer
11 from sklearn.feature_extraction.text import TfidfVectorizer
12
13 from sklearn.feature_extraction.text import CountVectorizer
14 from sklearn import metrics
15 from sklearn.model_selection import train_test_split
16 import re
17 # Tutorial about Python regular expressions: https://pymotw.com/2/re/
18 import string
19 from nltk.corpus import stopwords
20 from nltk.stem import SnowballStemmer
21 from nltk.stem.wordnet import WordNetLemmatizer
22 from gensim.models import Word2Vec
23 from gensim.models import KeyedVectors
24 import pickle
25
26 from tqdm import tqdm_notebook
27 from tqdm import tqdm
28 from bs4 import BeautifulSoup
29 import os
```

Function for object state :

- a. `saveToFile()`: to save the current state of object for future use using pickle.
- b. `openFromFile()`: to load the past state of object for further use.

```
In [25]: 1 #Functions to save objects for later use and retireve it
2 def savetofile(obj,filename):
3     pickle.dump(obj,open(filename+".pkl","wb"))
4 def openfromfile(filename):
5     temp = pickle.load(open(filename+".pkl","rb"))
6     return temp
7
```

2. Read the Dataset

- Create a Connection object that represents the database. Here the data will be stored in the 'database.sqlite' file.
- Read the Dataset table using connection object where the score column != 3
- Replace the score values with 'positive' and 'negative' label.(i.e Score 1 & 2 is labeled as negative and Score 4 & 5 is labeled as positive)
- Score with value 3 is neutral.

```

In [34]: 1 # using SQLite Table to read data.
2 con = sqlite3.connect('database.sqlite')
3
4 # filtering only positive and negative reviews i.e.
5 # not taking into consideration those reviews with Score=3
6 # SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
7 # you can change the number to any other number based on your computing power
8
9 # filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
10 # for tsne assignment you can take 5k data points
11
12 filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)
13
14 # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
15 def partition(x):
16     if x < 3:
17         return 0
18     return 1
19
20 #changing reviews with score less than 3 to be positive and vice-versa
21 actualScore = filtered_data['Score']
22 positiveNegative = actualScore.map(partition)
23 filtered_data['Score'] = positiveNegative
24 print("Number of data points in our data", filtered_data.shape)
25 filtered_data.head(3)

```

Number of data points in our data (525814, 10)

```

Out[34]:

```

		Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...

Type *Markdown* and LaTeX: α^2

```
In [35]: 1 display = pd.read_sql_query("""
2 SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
3 FROM Reviews
4 GROUP BY UserId
5 HAVING COUNT(*)>1
6 """, con)
```

```
In [36]: 1 print(display.shape)
2 display.head()
```

(80668, 7)

```
Out[36]:
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [37]: 1 display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[37]:
```

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

```
In [38]: 1 display['COUNT(*)'].sum()
```

```
Out[38]: 393063
```

4. Exploratory Data Analysis

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 [39]: 1 display= pd.read_sql_query("""
2 SELECT *
3 FROM Reviews
4 WHERE Score != 3 AND UserId="AR5J8UI46CURR"
5 ORDER BY ProductID
6 """, con)
7 display.head()
```

```
Out[39]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS FIND THEM EUROPEAN WAFERS
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS FIND THEM EUROPEAN WAFERS
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS FIND THEM EUROPEAN WAFERS
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS FIND THEM EUROPEAN WAFERS
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS FIND THEM EUROPEAN WAFERS

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

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 delete 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 [40]: 1 #Sorting data according to ProductId in ascending order
        2 sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_posit
```

```
In [41]: 1 #Deduplication of entries
        2 final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
        3 final.shape
```

Out[41]: (364173, 10)

```
In [42]: 1 #Checking to see how much % of data still remains
        2 (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[42]: 69.25890143662969

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 calculations


```
In [43]: 1 display= pd.read_sql_query("""
2 SELECT *
3 FROM Reviews
4 WHERE Score != 3 AND Id=44737 OR Id=64422
5 ORDER BY ProductID
6 """, con)
7
8 display.head(2)
```

```
Out[43]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

- 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

```
In [44]: 1 final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [45]: 1 #Before starting the next phase of preprocessing Lets see the number of entries left
2 print(final.shape)
3
4 #How many positive and negative reviews are present in our dataset?
5 final['Score'].value_counts()
```

```
(364171, 10)
```

```
Out[45]: 1 307061
0 57110
Name: Score, dtype: int64
```

```
In [46]: 1 final_pos=final.loc[final['Score']==1]
2 final_neg=final.loc[final['Score']==0]
3 final_pos=final_pos.sample(40000)
4 final_neg=final_neg.sample(30000)
5 final=final_pos.append(final_neg, ignore_index=True)
6 final = final.sample(frac=1).reset_index(drop=True)
```

5. Preprocessing

[5.1]. Preprocessing Review Text and Summary

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 observed to be better than Porter Stemming)

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

In [47]:

```
1 # https://gist.github.com/sebleier/554280
2 # we are removing the words from the stop words list: 'no', 'nor', 'not'
3 # <br /><br /> ==> after the above steps, we are getting "br br"
4 # we are including them into stop words list
5 # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step
6
7 stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
8 "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
9 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
10 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
11 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
12 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
13 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
14 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furth
15 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mc
16 'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
17 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're'
18 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
19 "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
20 "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "wer
21 'won', "won't", 'wouldn', "wouldn't"])
```

In [48]:

```

1  # https://stackoverflow.com/a/47091490/4084039
2  import re
3
4  def decontracted(phrase):
5      # specific
6      phrase = re.sub(r"won't", "will not", phrase)
7      phrase = re.sub(r"can't", "can not", phrase)
8
9      # general
10     phrase = re.sub(r"n't", " not", phrase)
11     phrase = re.sub(r"\ 're", " are", phrase)
12     phrase = re.sub(r"\ 's", " is", phrase)
13     phrase = re.sub(r"\ 'd", " would", phrase)
14     phrase = re.sub(r"\ 'll", " will", phrase)
15     phrase = re.sub(r"\ 't", " not", phrase)
16     phrase = re.sub(r"\ 've", " have", phrase)
17     phrase = re.sub(r"\ 'm", " am", phrase)
18     return phrase

```

In [49]:

```

1  # Combining all the above students
2  from tqdm import tqdm
3  def createCleanedText(final, review_text, column_name):
4      sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer
5      preprocessed_reviews = []
6      # tqdm is for printing the status bar
7      for sentence in tqdm(review_text):
8          sentence = re.sub(r"http\S+", "", sentence) # \S=except space; + = 1 or more
9          sentence = BeautifulSoup(sentence, 'lxml').get_text() # remove links
10         sentence = decontracted(sentence) # expand short forms
11         sentence = re.sub(r"\S*\d\S*", "", sentence).strip() #remove words containing digits
12         sentence = re.sub(r'[^A-Za-z]+', ' ', sentence) # remove special char
13         # https://gist.github.com/sebleier/554280
14         sentence = ' '.join(sno.stem(e.lower()) for e in sentence.split() if e.lower() not in stopwords)
15         preprocessed_reviews.append(sentence.strip())
16         #adding a column of CleanedText which displays the data after pre-processing of the review
17         final[column_name]=preprocessed_reviews
18     return final
19

```

```
In [50]: 1 def createDB(final, filename):
2         if not os.path.isfile(filename+'.sqlite'):#('final.sqlite'):
3             '''createCleanedText(final['Text_Summary'].values,column_name='CleanedTextSumm')
4             createCleanedText(final['Text'].values,column_name='CleanedText')'''
5             conn = sqlite3.connect(filename+'.sqlite')#('final.sqlite')
6             c=conn.cursor()
7             conn.text_factory = str
8             final.to_sql('Reviews', conn, schema=None, if_exists='replace', \
9                           index=True, index_label=None, chunksize=None, dtype=None)
10            conn.close()
```

```
In [69]: 1 def sampled_train_test_data(final, no_samples, filename, algo=None):
2         if algo=='kd':
3             final=final.sample(no_samples)
4             final=createCleanedText(final, final['Text'].values, column_name='CleanedText')
5             final=final.sort_values(by=['Time'])
6             #TEXT COLUMN
7             x=np.array(final['CleanedText'])
8             #SCORE COLUMN
9             y=np.array(final['Score'])
10            # split the data set into train and test
11            X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, shuffle=False)
12            print('X_train.shape=',X_train.shape,'y_train.shape=',y_train.shape)
13            print('X_test.shape=',X_test.shape,'y_test.shape=',y_test.shape)
14            #createDB(final, filename)
15            return X_train, X_test, y_train, y_test
```

```
In [70]: 1 X_train, X_test, y_train, y_test = sampled_train_test_data(final, 70000, 'final_brute')
2 X_train_kd, X_test_kd, y_train_kd, y_test_kd = sampled_train_test_data(final, 20000, 'final_kd', 'kd')
3
4 savetofile(X_train, 'X_train')
5 savetofile(X_test, 'X_test')
6
7 savetofile(y_train, 'y_train')
8 savetofile(y_test, 'y_test')
9
10 savetofile(X_train_kd, 'X_train_kd')
11 savetofile(X_test_kd, 'X_test_kd')
12
13 savetofile(y_train_kd, 'y_train_kd')
14 savetofile(y_test_kd, 'y_test_kd')
```

```
100%|██████████| 70000/70000 [01:12<00:00, 962.48it/s]
0%|          | 92/20000 [00:00<00:21, 912.61it/s]
```

```
X_train.shape= (49000,) y_train.shape= (49000,)
X_test.shape= (21000,) y_test.shape= (21000,)
```

```
100%|██████████| 20000/20000 [00:21<00:00, 944.03it/s]
```

```
X_train.shape= (14000,) y_train.shape= (14000,)
X_test.shape= (6000,) y_test.shape= (6000,)
```

```
In [21]: 1 if os.path.isfile('final.sqlite'):
2     conn = sqlite3.connect('final.sqlite')
3     final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, conn)
4     conn.close()
5 else:
6     print("Please the above cell")
```

In [22]:

```
1 print(final.head(3))
2 final.shape
```

```

      index      Id  ProductId      UserId  ProfileName  \
0  138695  150513  0006641040  ASH0DZQQF6AIZ    tessarat
1  138707  150525  0006641040  A2QID6VCFTY51R         Rick
2  138708  150526  0006641040  A3E9QZFE9KXH8J  R. Mitchell

      HelpfulnessNumerator  HelpfulnessDenominator  Score      Time  \
0                0                0          1  1325721600
1                1                2          1  1025481600
2               11               18          0  1129507200

                                Summary  \
0                                A classic
1  In December it will be, my snowman's anniversa...
2                                awesome book poor size

                                Text  \
0  I remembered this book from my childhood and g...
1  My daughter loves all the "Really Rosie" books...
2  This is one of the best children's books ever ...

                                Text_Summary  \
0  I remembered this book from my childhood and g...
1  My daughter loves all the "Really Rosie" books...
2  This is one of the best children's books ever ...

                                CleanedTextSumm  \
0  rememb book childhood got kid good rememb kid ...
1  daughter love realli rosi book introduc realli...
2  one best children book ever written mini versi...

                                CleanedText
0  rememb book childhood got kid good rememb kid ...
1  daughter love realli rosi book introduc realli...
2  one best children book ever written mini versi...
```

Out[22]: (160176, 14)

7. Featurization

[7.1] BAG OF WORDS

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- 1.A vocabulary of known words.
- 2.A measure of the presence of known words.

In [71]:

```
1  #bi-gram
2  def bowVector(X_train,X_test,max_features=None):
3      count_vect = CountVectorizer(ngram_range=(1,2),min_df=10,max_features=max_features)
4      X_train_bigram = count_vect.fit_transform(X_train)
5      print("the type of count vectorizer: ",type(X_train_bigram))
6      print("the shape of out text BOW vectorizer: ",X_train_bigram.get_shape())
7      print("the number of unique words including both unigrams and bigrams: ", X_train_bigram.get_shape()[1])
8
9      #processing of test data(convert test data into numerical vectors)
10     X_test_bigram = count_vect.transform(X_test)
11     print("the shape of out text BOW vectorizer: ",X_test_bigram.get_shape())
12     return count_vect, X_train_bigram,X_test_bigram
```


In [72]:

```

1  # BoW vector with all features
2  %time count_vect, X_train_bigram, X_test_bigram= bowVector(X_train,X_test,max_features=None)
3  savetofile(count_vect,'count_vect')
4  savetofile(X_train_bigram,'X_train_bigram')
5  savetofile(X_test_bigram,'X_test_bigram')
6
7  # BoW vector with all features
8  %time count_vect_kd, X_train_bigram_kd, X_test_bigram_kd= bowVector(X_train_kd,X_test_kd,max_features=500)
9  savetofile(count_vect_kd,'count_vect_kd')
10 savetofile(X_train_bigram_kd,'X_train_bigram_kd')
11 savetofile(X_test_bigram_kd,'X_test_bigram_kd')

```

```

the type of count vectorizer: <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer: (49000, 30357)
the number of unique words including both unigrams and bigrams: 30357
the shape of out text BOW vectorizer: (21000, 30357)
CPU times: user 12.5 s, sys: 336 ms, total: 12.9 s
Wall time: 12.8 s
the type of count vectorizer: <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer: (14000, 500)
the number of unique words including both unigrams and bigrams: 500
the shape of out text BOW vectorizer: (6000, 500)
CPU times: user 3.94 s, sys: 68 ms, total: 4 s
Wall time: 3.99 s

```

[7.2] TF-IDF

Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

- 1.TF: Term Frequency, which measures how frequently a term occurs in a document.

$$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document}).$$
- 2.IDF: Inverse Document Frequency, is a scoring of how rare the word is across documents.

$$IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it}).$$
- 3.The scores are a weighting where not all words are equally as important or interesting.

The scores have the effect of highlighting words that are distinct (contain useful information) in a given document. The idf of a rare term is high, whereas the idf of a frequent term is likely to be low.

```
In [73]: 1 def tfidfVector(X_train,X_test, max_features=None):
2         tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=max_features)
3         X_train_tfidf = tf_idf_vect.fit_transform(X_train)
4         print("the type of count vectorizer: ",type(X_train_tfidf))
5         print("the shape of out text TFIDF vectorizer: ",X_train_tfidf.get_shape())
6         print("the number of unique words including both unigrams and bigrams: ", X_train_tfidf.get_shape()[1])
7
8         #processing of test data(convert test data into numerical vectors)
9         X_test_tfidf = tf_idf_vect.transform(X_test)
10        print("the shape of out text BOW vectorizer: ",X_test_tfidf.get_shape())
11        return tf_idf_vect, X_train_tfidf, X_test_tfidf
```

```
In [74]: 1 %%time
2         # Tfidf vector with all features which we use for brute force implementation
3         tf_idf_vect, X_train_tfidf, X_test_tfidf=tfidfVector(X_train,X_test,max_features=None)
4         savetofile(tf_idf_vect,'tf_idf_vect')
5         savetofile(X_train_tfidf,'X_train_tfidf')
6         savetofile(X_test_tfidf,'X_test_tfidf')
7
8         tf_idf_vect_kd, X_train_tfidf_kd, X_test_tfidf_kd=tfidfVector(X_train_kd,X_test_kd,max_features=500)
9         savetofile(tf_idf_vect_kd,'tf_idf_vect_kd')
10        savetofile(X_train_tfidf_kd,'X_train_tfidf_kd')
11        savetofile(X_test_tfidf_kd,'X_test_tfidf_kd')
```

```
the type of count vectorizer: <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer: (49000, 30357)
the number of unique words including both unigrams and bigrams: 30357
the shape of out text BOW vectorizer: (21000, 30357)
the type of count vectorizer: <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer: (14000, 500)
the number of unique words including both unigrams and bigrams: 500
the shape of out text BOW vectorizer: (6000, 500)
CPU times: user 17.5 s, sys: 696 ms, total: 18.2 s
Wall time: 18.2 s
```

[7.3] Word2Vec

```
In [75]: 1 # Train your own Word2Vec model using your own text corpus
2 def preSETUPW2V(X_train,X_test):
3     i=0
4     list_of_sent=[]
5     for sent in X_train:
6         list_of_sent.append(sent.split())
7
8     list_of_sent_test=[]
9     for sent in X_test:
10        list_of_sent_test.append(sent.split())
11    return list_of_sent,list_of_sent_test
```

```
In [76]: 1 list_of_sent,list_of_sent_test=preSETUPW2V(X_train,X_test)
2 list_of_sent_fe,list_of_sent_test_fe=preSETUPW2V(X_train_kd,X_test_kd)
```

```
In [77]: 1 size_of_w2v=100
2 def w2vMODEL(list_of_sent,list_of_sent_test):
3     # Using Google News Word2Vectors
4     is_your_ram_gt_16g=False
5     want_to_use_google_w2v = False
6     want_to_train_w2v = True
7     if want_to_train_w2v:
8         #min_count = 5 considers only words that occurred at least 5 times
9         w2v_model=Word2Vec(list_of_sent,min_count=5,size=size_of_w2v, workers=4)
10
11    elif want_to_use_google_w2v and is_your_ram_gt_16g:
12        if os.path.isfile('GoogleNews-vectors-negative300.bin'):
13            w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
14        else:
15            print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")
16    return w2v_model
```

```
In [78]: 1 w2v_model=w2vMODEL(list_of_sent,list_of_sent_test)
2 w2v_model_fe=w2vMODEL(list_of_sent_fe,list_of_sent_test_fe)
3 w2v_words = list(w2v_model.wv.vocab)
4 w2v_words_fe = list(w2v_model_fe.wv.vocab)
```

[7.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[7.4.1.1] Avg W2v

In [79]:

```
1  # average Word2Vec
2  # compute average word2vec for each review.
3  def avg_w2v(w2v_model,vocab,list_of_sent,size):
4      sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
5      for sent in tqdm(list_of_sent): # for each review/sentence
6          sent_vec = np.zeros(size) # as word vectors are of zero length 50, you might need to change this to 300 if y
7          cnt_words = 0; # num of words with a valid vector in the sentence/review
8          for word in sent: # for each word in a review/sentence
9              if word in vocab:
10                 vec = w2v_model.wv[word]
11                 sent_vec += vec
12                 cnt_words += 1
13             if cnt_words != 0:
14                 sent_vec /= cnt_words
15             sent_vectors.append(sent_vec)
16     print(len(sent_vectors))
17     print('dimension:',len(sent_vectors[0]))
18     return sent_vectors
```

In [80]:

```

1  # Parallelizing using Pool.apply()
2
3  import multiprocessing as mp
4
5  # Step 1: Init multiprocessing.Pool()
6  pool = mp.Pool(mp.cpu_count())
7  # Step 2: `pool.apply` the `howmany_within_range()`
8  %time avg_sent_vectors = pool.apply(avg_w2v, args=(w2v_model,w2v_words,list_of_sent,size_of_w2v))
9  # Step 3: Don't forget to close
10 pool.close()
11
12 pool = mp.Pool(mp.cpu_count())
13 %time avg_sent_vectors_test = pool.apply(avg_w2v, args=(w2v_model,w2v_words,list_of_sent_test,size_of_w2v))
14 pool.close()
15
16 pool = mp.Pool(mp.cpu_count())
17 %time avg_sent_vectors_kd = pool.apply(avg_w2v, args=(w2v_model_fe,w2v_words_fe,list_of_sent_fe,size_of_w2v))
18 pool.close()
19
20 pool = mp.Pool(mp.cpu_count())
21 %time avg_sent_vectors_test_kd = pool.apply(avg_w2v, args=(w2v_model_fe,w2v_words_fe,list_of_sent_test_fe,size_of_w2v))
22 pool.close()

```

100%|██████████| 49000/49000 [05:17<00:00, 154.28it/s]

49000

dimension: 100

CPU times: user 4.26 s, sys: 2.53 s, total: 6.79 s

Wall time: 5min 21s

100%|██████████| 21000/21000 [02:11<00:00, 160.20it/s]

21000

dimension: 100

CPU times: user 1.8 s, sys: 1.04 s, total: 2.85 s

Wall time: 2min 12s

100%|██████████| 14000/14000 [00:39<00:00, 354.21it/s]

14000

dimension: 100

CPU times: user 688 ms, sys: 492 ms, total: 1.18 s

Wall time: 40.4 s

100%|██████████| 6000/6000 [00:12<00:00, 464.34it/s]

6000

dimension: 100

CPU times: user 296 ms, sys: 148 ms, total: 444 ms

Wall time: 13.3 s

In [81]:

```
1 savetofile(avg_sent_vectors, 'avg_sent_vectors')
2 savetofile(avg_sent_vectors_test, 'avg_sent_vectors_test')
3 savetofile(avg_sent_vectors_kd, 'avg_sent_vectors_kd')
4 savetofile(avg_sent_vectors_test_kd, 'avg_sent_vectors_test_kd')
```

[7.4.1.2] TFIDF weighted W2v

```
In [82]: 1 def tfidf_w2v(w2v_model,vocab,tf_idf_vect,list_of_sent,size):
2         # TF-IDF weighted Word2Vec for Train
3         dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))
4         tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
5         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf
6         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
7         row=0;
8         for sent in tqdm(list_of_sent): # for each review/sentence
9             sent_vec = np.zeros(size) # as word vectors are of zero length
10            weight_sum =0; # num of words with a valid vector in the sentence/review
11            for word in sent: # for each word in a review/sentence
12                if word in vocab and word in tfidf_feat:
13                    vec = w2v_model.wv[word]
14                    # tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
15                    # to reduce the computation we are
16                    # dictionary[word] = idf value of word in whole corpus
17                    # sent.count(word) = tf value of word in this review
18                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
19                    sent_vec += (vec * tf_idf)
20                    weight_sum += tf_idf
21            if weight_sum != 0:
22                sent_vec /= weight_sum
23            tfidf_sent_vectors.append(sent_vec)
24            row += 1
25        return tfidf_sent_vectors
```

```

In [83]: 1 # Parallelizing using Pool.apply()
2
3 import multiprocessing as mp
4
5 # Step 1: Init multiprocessing.Pool()
6 pool = mp.Pool(mp.cpu_count())
7 # Step 2: `pool.apply` the `howmany_within_range()`
8 %time tfidf_sent_vectors = pool.apply(tfidf_w2v_, args=(w2v_model,w2v_words,tf_idf_vect,list_of_sent,size_of_w2v))
9 # Step 3: Don't forget to close
10 pool.close()
11
12 pool = mp.Pool(mp.cpu_count())
13 %time tfidf_sent_vectors_test = pool.apply(tfidf_w2v_, args=(w2v_model,w2v_words,tf_idf_vect,list_of_sent_test,size_
14 pool.close()
15
16 pool = mp.Pool(mp.cpu_count())
17 %time tfidf_sent_vectors_kd = pool.apply(tfidf_w2v_, args=(w2v_model_fe,w2v_words_fe,tf_idf_vect_kd,list_of_sent_fe,
18 pool.close()
19
20 pool = mp.Pool(mp.cpu_count())
21 %time tfidf_sent_vectors_test_kd = pool.apply(tfidf_w2v_, args=(w2v_model_fe,w2v_words_fe,tf_idf_vect_kd,list_of_ser
22 pool.close()

```

100%|██████████| 49000/49000 [18:01<00:00, 45.32it/s]

CPU times: user 11.9 s, sys: 6.07 s, total: 17.9 s
Wall time: 18min 6s

100%|██████████| 21000/21000 [07:44<00:00, 45.21it/s]

CPU times: user 5.1 s, sys: 2.76 s, total: 7.86 s
Wall time: 7min 47s

100%|██████████| 14000/14000 [00:37<00:00, 370.10it/s]

CPU times: user 1.04 s, sys: 436 ms, total: 1.48 s
Wall time: 39 s

100%|██████████| 6000/6000 [00:14<00:00, 418.74it/s]

CPU times: user 420 ms, sys: 292 ms, total: 712 ms
Wall time: 15.1 s

In [84]:

```
1 savetofile(tfidf_sent_vectors, 'tfidf_sent_vectors')
2 savetofile(tfidf_sent_vectors_test, 'tfidf_sent_vectors_test')
3
4 savetofile(tfidf_sent_vectors_kd, 'tfidf_sent_vectors_kd')
5 savetofile(tfidf_sent_vectors_test_kd, 'tfidf_sent_vectors_test_kd')
```