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 - 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).

1. Import required libraries

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

In [3]:
```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.model_selection import train_test_split
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm_notebook
from tqdm import tqdm
from bs4 import BeautifulSoup
import os
```

2. Read the Dataset

```
a. Create a Connection object that represents the database. Here the data will be stored in the 'databas
e.sqlite' file.
b. Read the Dataset table using connection object where the score column != 3
c. 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)
d. Score with value 3 is neutral.
```

In [4]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.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 500000 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 Score != 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 """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<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 (525814, 10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I sev c
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	P a labe Pe

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	Tl con th ar

Type *Markdown* and LaTeX: α^2

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

In [6]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[6]:

	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 [7]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:

	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 [8]:
```
display['COUNT(*)'].sum()
```

Out[8]: 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 [9]:

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA 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 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 [10]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort',
```

In [11]:

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

Out[11]:

```
(364173, 10)
```

In [12]:

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

Out[12]:

```
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 calcualtions

In [13]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head(2)
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	sp h
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	a fi

- 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 [14]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [15]:
```
#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()
```

```
(364171, 10)
```

Out[15]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

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 obsereved to be better than Porter Stemming)

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

In [16]:

```
# 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 [17]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <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', 'ours', 'ourselves', 'you', "you're", "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'tho
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', '
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'f
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mus
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'were
            'won', "won't", 'wouldn', "wouldn't"])
```

In [19]:

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

```

In [20]:

```
if not os.path.isfile('final.sqlite'):
    #createCleanedText(final['Text_Summary'].values,column_name='CleanedTextSumm')
    createCleanedText(final['Text'].values,column_name='CleanedText')
    conn = sqlite3.connect('final.sqlite')
    c=conn.cursor()
    conn.text_factory = str
    final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()
```

```
100%|██████████| 364171/364171 [06:19<00:00, 958.48it/s]
```

In [21]:

```
if os.path.isfile('final.sqlite'):
    conn = sqlite3.connect('final.sqlite')
    final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, conn)
    conn.close()
else:
    print("Please the above cell")
```

In [22]:
```
print(final.head(3))
final.shape
```

```
    index      Id   ProductId          UserId          ProfileName  \
0  138706  150524  0006641040   ACITT7DI6IDDL        shari zychinski   
1  138688  150506  0006641040  A2IW4PEEKO2R0U                Tracy   
2  138689  150507  0006641040  A1S4A3IQ2MU7V4  sally sue "sally sue"   

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     0                       0      1   939340800   
1                     1                       1      1  1194739200   
2                     1                       1      1  1191456000   

                                    Summary  \
0                    EVERY book is educational   
1  Love the book, miss the hard cover version   
2               chicken soup with rice months   

                                                Text  \
0  this witty little book makes my son laugh at l...   
1  I grew up reading these Sendak books, and watc...   
2  This is a fun way for children to learn their ...   

                                         CleanedText  
0  witti littl book make son laugh loud recit car...  
1  grew read sendak book watch realli rosi movi i...  
2  fun way children learn month year learn poem t...  
```

Out[22]:
```
(364171, 12)
```

6. Randomly smpled data points

In [27]:
```
#randomly sample 8000 points (positive and negative)
df=final.sample(n=8000)
#sorted dataFrame by time
'''
df['Time']=pd.to_datetime(final['Time'],unit='s')
df=df.sort_values(by="Time")
df.head(20)
'''
df=df.sort_values(by=['Time'])
#df.head(5)
```

In [28]:
```
#TEXT COLUMN
X=np.array(df['CleanedText'])
#SCORE COLUMN
y=np.array(df['Score'])
```

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 [30]:
```
#bi-gram
def bowVector(X,max_features=None):
    count_vect = CountVectorizer(ngram_range=(1,2),min_df=5,max_features=max_features)
    X_bigram = count_vect.fit_transform(X)
    print("the type of count vectorizer: ",type(X_bigram))
    print("the shape of out text BOW vectorizer: ",X_bigram.get_shape())
    print("the number of unique words including both unigrams and bigrams: ", X_bigram.get_shape()[1])

    return count_vect, X_bigram
```

In [31]:
```
# BoW vector with all features
%time count_vect, X_bigram= bowVector(X,max_features=None)
# BoW vector with feature engineering
#%time count_vect_fe,X_train_bigram_fe,X_test_bigram_fe=bowVector(X_train_fe,X_test_fe,max_features=None)
#tfidf vector with 500 feature and without summ. include
#%time  count_vect_500, X_train_bigram_500, X_test_bigram_500=bowVector(X_train,X_test,max_features=500)
#tfidf vector with 500 feature and without summ. include
#%time  count_vect_fe500, X_train_bigram_fe500, X_test_bigram_fe500=bowVector(X_train_fe,X_test_fe,max_featu
```

```
the type of count vectorizer:  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer:  (8000, 11070)
the number of unique words including both unigrams and bigrams:  11070
CPU times: user 1.91 s, sys: 16 ms, total: 1.92 s
Wall time: 1.92 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) = (Number of times term t appears in a document) / (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(Total number of documents / Number of documents with term t 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 [32]:
```
def tfidfVector(X, max_features=None):
    tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=5,max_features=max_features)
    X_tfidf = tf_idf_vect.fit_transform(X)
    print("the type of count vectorizer: ",type(X))
    print("the shape of out text TFIDF vectorizer: ",X_tfidf.get_shape())
    print("the number of unique words including both unigrams and bigrams: ", X_tfidf.get_shape()[1])
    return tf_idf_vect, X_tfidf
```

In [33]:
```
# Tfidf vector with all features which we use for brute force implementation
%time tf_idf_vect, X_tfidf=tfidfVector(X,max_features=None)
# Tfidf vector with feature engineering
#%time tf_idf_vect_fe, X_train_tfidf_fe, X_test_tfidf_fe=tfidfVector(X_train_fe,X_test_fe,max_features=None)
#tfidf vector with 500 feature and without summ. include
#%time tf_idf_vect_500, X_train_tfidf_500, X_test_tfidf_500=tfidfVector(X_train,X_test,max_features=500)
#tfidf vector with 500 feature and without summ. include
#%time tf_idf_vect_fe500, X_train_tfidf_fe500, X_test_tfidf_fe500=tfidfVector(X_train_fe,X_test_fe,max_featu
```

```
the type of count vectorizer:  <class 'numpy.ndarray'>
the shape of out text TFIDF vectorizer:  (8000, 11070)
the number of unique words including both unigrams and bigrams:  11070
CPU times: user 1.92 s, sys: 8 ms, total: 1.92 s
Wall time: 1.92 s
```

[7.3] Word2Vec

In [34]:
```
# Train your own Word2Vec model using your own text corpus
def preSETUPW2V(X):
    i=0
    list_of_sent=[]
    for sent in X:
        list_of_sent.append(sent.split())

    return list_of_sent
```

In [35]:
```
list_of_sent=preSETUPW2V(X)
#list_of_sent_fe,list_of_sent_test_fe=preSETUPW2V(X_train_fe,X_test_fe)
```

In [36]:

```
size_of_w2v=100
def w2vMODEL(list_of_sent):
    # Using Google News Word2Vectors
    is_your_ram_gt_16g=False
    want_to_use_google_w2v = False
    want_to_train_w2v = True
    if want_to_train_w2v:
        #min_count = 5 considers only words that occured atleast 5 times
        w2v_model=Word2Vec(list_of_sent,min_count=5,size=size_of_w2v, workers=4)

    elif want_to_use_google_w2v and is_your_ram_gt_16g:
        if os.path.isfile('GoogleNews-vectors-negative300.bin'):
            w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
        else:
            print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w
    return w2v_model
```

In [37]:

```
w2v_model=w2vMODEL(list_of_sent)
#w2v_model_fe=w2vMODEL(list_of_sent_fe,list_of_sent_test_fe)
w2v_words = list(w2v_model.wv.vocab)
#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 [38]:

```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2v(w2v_model,vocab,list_of_sent,size):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent): # for each review/sentence
        sent_vec = np.zeros(size) # as word vectors are of zero length 50, you might need to change this to
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in vocab:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print('dimension:',len(sent_vectors[0]))
    return sent_vectors
```

In [39]:

```
# Parallelizing using Pool.apply()

import multiprocessing as mp

# Step 1: Init multiprocessing.Pool()
pool = mp.Pool(mp.cpu_count())
# Step 2: `pool.apply` the `howmany_within_range()`
%time avg_sent_vectors = pool.apply(avg_w2v, args=(w2v_model,w2v_words,list_of_sent,size_of_w2v))
# Step 3: Don't forget to close
pool.close()
```

```
100%|██████████| 8000/8000 [00:14<00:00, 538.06it/s]

8000
dimension: 100
CPU times: user 260 ms, sys: 132 ms, total: 392 ms
Wall time: 15.5 s
```

Out[39]:

```
'pool = mp.Pool(mp.cpu_count())\n%time avg_sent_vectors_test = pool.apply(avg_w2v, args=(w2v_model,w2v_words,li
st_of_sent_test,size_of_w2v))\npool.close()\n\npool = mp.Pool(mp.cpu_count())\n%time avg_sent_vectors_fe = poo
l.apply(avg_w2v, args=(w2v_model_fe,w2v_words_fe,list_of_sent_fe,size_of_w2v))\npool.close()\n\npool = mp.Pool
(mp.cpu_count())\n%time avg_sent_vectors_test_fe = pool.apply(avg_w2v, args=(w2v_model_fe,w2v_words_fe,list_of_
sent_test_fe,size_of_w2v))\npool.close()'
```

[7.4.1.2] TFIDF weighted W2v

In [40]:

```
def tfidf_w2v_(w2v_model,vocab,tf_idf_vect,list_of_sent,size):
    # TF-IDF weighted Word2Vec for Train
    dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))
    tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    row=0;
    for sent in tqdm(list_of_sent): # for each review/sentence
        sent_vec = np.zeros(size) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in vocab 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.append(sent_vec)
        row += 1
    return tfidf_sent_vectors
```

In [41]:
```
# Parallelizing using Pool.apply()

import multiprocessing as mp

# Step 1: Init multiprocessing.Pool()
pool = mp.Pool(mp.cpu_count())
# Step 2: `pool.apply` the `howmany_within_range()`
%time tfidf_sent_vectors = pool.apply(tfidf_w2v_, args=(w2v_model,w2v_words,tf_idf_vect,list_of_sent,size_of
# Step 3: Don't forget to close
pool.close()
```

```
100%|██████████| 8000/8000 [00:58<00:00, 137.65it/s]

CPU times: user 708 ms, sys: 372 ms, total: 1.08 s
Wall time: 59.1 s
```

Out[41]:
```
'pool = mp.Pool(mp.cpu_count())\n%time tfidf_sent_vectors_test = pool.apply(tfidf_w2v_, args=(w2v_model,w2v_words,tf_idf_vect,list_of_sent_test,size_of_w2v))\npool.close()\n\npool = mp.Pool(mp.cpu_count())\n%time tfidf_sent_vectors_fe = pool.apply(tfidf_w2v_, args=(w2v_model_fe,w2v_words_fe,tf_idf_vect_fe,list_of_sent_fe,size_of_w2v))\npool.close()\n\npool = mp.Pool(mp.cpu_count())\n%time tfidf_sent_vectors_test_fe = pool.apply(tfidf_w2v_, args=(w2v_model_fe,w2v_words_fe,tf_idf_vect_fe,list_of_sent_test_fe,size_of_w2v))\npool.close()'
```

9. 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 [43]:

```
#Functions to save objects for later use and retireve it
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".pkl","wb"))
def openfromfile(filename):
    temp = pickle.load(open(filename+".pkl","rb"))
    return temp

savetofile(count_vect,'count_vect')
savetofile(X_bigram,'X_bigram')

savetofile(tf_idf_vect,'tf_idf_vect')
savetofile(X_tfidf,'X_tfidf')

savetofile(avg_sent_vectors,'avg_sent_vectors')

savetofile(tfidf_sent_vectors,'tfidf_sent_vectors')

savetofile(X,'X')
savetofile(y,'y')
```

In [ ]:

```

```