Vectorization of text using different techniques on Amazon Fine Food Reviews

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

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

Number of reviews : 568,454

Number of 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:

to covert text to vectors using :

1. Bag of Words
2. TF-IDF
3. W2V
4. Avg-W2V
5. TF-IDF W2v

1. Reading Data

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 easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [3]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

import 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



# using the 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
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)


# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#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
```

In [4]:

```
print(filtered_data.shape) #looking at the number of attributes and size of the data
filtered_data.head()
```

```
(525814, 10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

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 [4]:

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

Out[4]:

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

As can be seen above the same user has multiple reviews of the with the 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 [5]:

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

In [6]:

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

Out[6]:

```
(364173, 10)
```

In [7]:

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

Out[7]:

```
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 [8]:

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

display.head()
```

Out[8]:

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

In [9]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [10]:

```
#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[10]:

```
positive    307061
negative     57110
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 [13]:

```
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this poi
nt, I consider myself a connoisseur of children's books and this is one of t
he best. Santa Clause put this under the tree. Since then, we've read it per
petually and he loves it.<br /><br />First, this book taught him the months
of the year.<br /><br />Second, it's a pleasure to read. Well suited to 1.5
y/o old to 4+.<br /><br />Very few children's books are worth owning. Most s
hould be borrowed from the library. This book, however, deserves a permanent
spot on your shelf. Sendak's best.
```

In [17]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characte
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

```
{'theirs', 'can', 'again', 'do', 'while', 'doesn', 'into', 'her', 'other',
"aren't", 'before', 'ourselves', 'about', 'to', 'nor', 'too', "it's", 'sam
e', 'haven', 'most', 'been', 'any', 'has', "shan't", 'themselves', 's', 'a
t', 'did', 'this', "mustn't", "wasn't", 'than', 'above', 'won', 'over', 'ai
n', 'will', "you're", 'so', 'off', 'are', 'down', 'how', 'some', 'own', 'i',
'their', 'for', 'y', 'who', "wouldn't", 'which', 'very', 'didn', 'isn', 'bet
ween', 'under', 'from', 'of', 'by', "don't", 'just', 'she', "you'll", 'is',
'they', 'shouldn', 'its', 'during', "you've", 'ours', "isn't", 'our', "has
n't", 'why', 'through', 'on', 'further', 've', 'what', 'after', 'mightn', 'i
n', 'because', "you'd", 'he', 'yourself', 'those', 'needn', 'yourselves', 'd
oes', 'or', 'until', "haven't", 'aren', "weren't", 'were', 'don', 'them', 'y
our', 'm', 'hers', 'shan', 'that', 'against', 'few', "didn't", 'once', 'itse
lf', 'me', 're', 'him', 'whom', 'but', 'hasn', 'myself', 'when', 'wouldn',
'each', 'a', 'such', 'not', "won't", "couldn't", 'out', 'o', "hadn't", 'must
n', 't', 'ma', 'all', 'as', 'if', 'an', 'd', 'the', 'couldn', 'am', 'no', 'm
y', 'where', "needn't", "that'll", 'himself', 'yours', "she's", 'with', 'an
d', 'herself', 'doing', "doesn't", "shouldn't", 'up', 'we', 'be', "might
n't", 'weren', 'only', 'here', 'both', 'it', 'these', 'have', 'hadn', "shoul
d've", 'more', 'was', 'had', 'his', 'then', 'should', 'now', 'being', 'havin
g', 'll', 'you', 'below', 'wasn', 'there'}
************************************
tasti
```

In [20]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe po
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe ne
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("*****************************************************************")

    final_string.append(str1)
    i+=1
```

In [21]:

```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data a
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
```

In [23]:

```
final.head(3) #below the processed review can be seen in the CleanedText Column


# store final table into an SQLLite table for future.
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=No
```

4. Bag of Words (BoW)

In [24]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
final_counts = count_vect.fit_transform(final['CleanedText'].values)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 71624)
the number of unique words  71624
```

4.1. Bi-Grams and n-Grams.

Motivation

Now that we have our list of words describing positive and negative reviews lets analyse them.

We begin analysis by getting the frequency distribution of the words as shown below

In [44]:

```
freq_dist_positive=nltk.FreqDist(all_positive_words)
freq_dist_negative=nltk.FreqDist(all_negative_words)
print("Most Common Positive Words : ",freq_dist_positive.most_common(20))
print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive Words :  [(b'like', 139429), (b'tast', 129047), (b'goo
d', 112766), (b'flavor', 109624), (b'love', 107357), (b'use', 103888), (b'gr
eat', 103870), (b'one', 96726), (b'product', 91033), (b'tri', 86791), (b'te
a', 83888), (b'coffe', 78814), (b'make', 75107), (b'get', 72125), (b'food',
64802), (b'would', 55568), (b'time', 55264), (b'buy', 54198), (b'realli', 52
715), (b'eat', 52004)]
Most Common Negative Words :  [(b'tast', 34585), (b'like', 32330), (b'produc
t', 28218), (b'one', 20569), (b'flavor', 19575), (b'would', 17972), (b'tri',
17753), (b'use', 15302), (b'good', 15041), (b'coffe', 14716), (b'get', 1378
6), (b'buy', 13752), (b'order', 12871), (b'food', 12754), (b'dont', 11877),
(b'tea', 11665), (b'even', 11085), (b'box', 10844), (b'amazon', 10073), (b'm
ake', 9840)]
```

Observation:- From the above it can be seen that the most common positive and the negative words overlap for eg. 'like' could be used as 'not like' etc.
So, it is a good idea to consider pairs of consequent words (bi-grams) or q sequnce of n consecutive words (n-grams)

In [26]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
final_bigram_counts = count_vect.fit_transform(final['CleanedText'].values)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_count
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 2923725)
the number of unique words including both unigrams and bigrams  2923725
```

5. TF-IDF

In [33]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final['CleanedText'].values)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_s
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (364171, 2923725)
the number of unique words including both unigrams and bigrams  2923725
```

In [32]:

```
features = tf_idf_vect.get_feature_names()
print("some sample features(unique words in the corpus)",features[100000:100010])
```

```
some sample features(unique words in the corpus) ['antler fair', 'antler fal
l', 'antler fantast', 'antler far', 'antler fault', 'antler favorit', 'antle
r feel', 'antler french', 'antler get', 'antler girl']
```

In [34]:

```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature name
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(final_tf_idf[1,:].toarray()[0],features,25)
```

In [35]:

```
top_tfidf
```

Out[35]:

	feature	tfidf
0	page open	0.192673
1	read sendak	0.192673
2	movi incorpor	0.192673
3	paperback seem	0.192673
4	version paperback	0.192673
5	flimsi take	0.192673
6	incorpor love	0.192673
7	rosi movi	0.192673
8	keep page	0.192673
9	grew read	0.192673
10	realli rosi	0.186715
11	sendak book	0.186715
12	cover version	0.186715
13	miss hard	0.182488
14	kind flimsi	0.176530
15	hard cover	0.174265
16	watch realli	0.170572
17	book watch	0.170572
18	sendak	0.166345
19	howev miss	0.165169
20	paperback	0.162117
21	hand keep	0.153894
22	two hand	0.150202
23	rosi	0.148291
24	seem kind	0.147253

6. Word2Vec

In [39]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell


model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True
print("the vector representation of word 'computer'",model.wv['computer'])
print("the similarity between the words 'woman' and 'man'",model.wv.similarity('woman', 'ma
print("the most similar words to the word 'woman'",model.wv.most_similar('woman'))
# this will raise an error
# model.wv.most_similar('tasti')  # "tasti" is the stemmed word for tasty, tastful
```

```
the vector representation of word 'computer' [ 1.07421875e-01 -2.01171875e
-01  1.23046875e-01  2.11914062e-01
 -9.13085938e-02  2.16796875e-01 -1.31835938e-01  8.30078125e-02
  2.02148438e-01  4.78515625e-02  3.66210938e-02 -2.45361328e-02
  2.39257812e-02 -1.60156250e-01 -2.61230469e-02  9.71679688e-02
 -6.34765625e-02  1.84570312e-01  1.70898438e-01 -1.63085938e-01
 -1.09375000e-01  1.49414062e-01 -4.65393066e-04  9.61914062e-02
  1.68945312e-01  2.60925293e-03  8.93554688e-02  6.49414062e-02
  3.56445312e-02 -6.93359375e-02 -1.46484375e-01 -1.21093750e-01
 -2.27539062e-01  2.45361328e-02 -1.24511719e-01 -3.18359375e-01
 -2.20703125e-01  1.30859375e-01  3.66210938e-02 -3.63769531e-02
 -1.13281250e-01  1.95312500e-01  9.76562500e-02  1.26953125e-01
  6.59179688e-02  6.93359375e-02  1.02539062e-02  1.75781250e-01
 -1.68945312e-01  1.21307373e-03 -2.98828125e-01 -1.15234375e-01
  5.66406250e-02 -1.77734375e-01 -2.08984375e-01  1.76757812e-01
  2.38037109e-02 -2.57812500e-01 -4.46777344e-02  1.88476562e-01
  5.51757812e-02  5.02929688e-02 -1.06933594e-01  1.89453125e-01
 -1.16210938e-01  8.49609375e-02 -1.71875000e-01  2.45117188e-01
 -1.73828125e-01 -8.30078125e-03  4.56542969e-02 -1.61132812e-02
```

In [50]:

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

In [51]:

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

```
witti littl book make son laugh loud recit car drive along alway sing refrai
n hes learn whale india droop love new word book introduc silli classic book
will bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car',
'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whale', 'indi
a', 'droop', 'love', 'new', 'word', 'book', 'introduc', 'silli', 'classic',
'book', 'will', 'bet', 'son', 'still', 'abl', 'recit', 'memori', 'colleg']
```

In [53]:

```
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [63]:

```
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])
```

```
number of words that occured minimum 5 times  21938
sample words  ['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 're
cit', 'car', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn',
'whale', 'india', 'droop', 'love', 'new', 'word', 'introduc', 'silli', 'clas
sic', 'will', 'bet', 'still', 'abl', 'memori', 'colleg', 'grew', 'read', 'se
ndak', 'watch', 'realli', 'rosi', 'movi', 'incorpor', 'howev', 'miss', 'har
d', 'cover', 'version', 'paperback', 'seem', 'kind', 'flimsi', 'take', 'tw
o']
```

In [56]:

```
w2v_model.wv.most_similar('tasti')
```

Out[56]:

```
[('delici', 0.811565637588501),
 ('tastey', 0.7988753318786621),
 ('yummi', 0.770585834980011),
 ('satisfi', 0.6894760131835938),
 ('good', 0.6716829538345337),
 ('nice', 0.6640280485153198),
 ('hearti', 0.6638340950012207),
 ('nutriti', 0.661227822303772),
 ('crunchi', 0.656670331954956),
 ('versatil', 0.6266734004020691)]
```

In [57]:

```
w2v_model.wv.most_similar('like')
```

Out[57]:

```
[('weird', 0.7463775277137756),
 ('dislik', 0.7033219933509827),
 ('okay', 0.6805294752120972),
 ('prefer', 0.671764612197876),
 ('gross', 0.670783519744873),
 ('appeal', 0.6521075963973999),
 ('resembl', 0.641674280166626),
 ('funki', 0.6357078552246094),
 ('yucki', 0.6338918209075928),
 ('hate', 0.6323825716972351)]
```

In [59]:

```
count_vect_feat = count_vect.get_feature_names() # list of words in the BoW
print(count_vect_feat[count_vect_feat.index('like')])
```

```
like
```

7. Avg W2V, TFIDF-W2V

In [64]:

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

```
364171
50
```

In [ ]:

```
# TF-IDF weighted Word2Vec
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 list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            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
```