

# \_TSNE\_FINAL

April 5, 2019

## 1 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 - 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).

[Q] How to determine if a review is positive or negative? [Ans] We could use the Score/Rating. A rating of 4 or 5 could be considered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is neutral and ignored. 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 will 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
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

```

C:\Users\MANISH\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows warnings.warn("detected Windows; aliasing chunkize to chunkize\_serial")

## 2 [1]. Reading Data

```

In [2]: # using the SQLite Table to read data.
con = sqlite3.connect('./amazon-fine-food-reviews/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 point.

```

```

# 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 5000 """)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000 """)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating
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 (5000, 10)

```

Out[2]:

```

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	d11 pa	
2	3	B000LQOCHO	ABXLMWJIXXAIN	Natalia Corres	"Natalia Corres"

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
0	1	1	1	1303862400	
1	0	0	0	1346976000	
2	1	1	1	1219017600	

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...

```

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

```

```

In [4]: print(display.shape)
display.head()

```

(80668, 7)

```
Out [4]:
```

	UserId	ProductId	ProfileName	Time	Score	\
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	
1	#oc-R11D9D7SHXIJB9	B005HG9ETO	Louis E. Emory "hoppy"	1342396800	5	
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	
3	#oc-R1105J5ZVQE25C	B005HG9ETO	Penguin Chick	1346889600	5	
4	#oc-R12KPBODL2B5ZD	B0070SBE1U	Christopher P. Presta	1348617600	1	

	Text	COUNT(*)
0	Overall its just OK when considering the price...	2
1	My wife has recurring extreme muscle spasms, u...	3
2	This coffee is horrible and unfortunately not ...	2
3	This will be the bottle that you grab from the...	3
4	I didnt like this coffee. Instead of telling y...	2

```
In [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out [5]:
```

	UserId	ProductId	ProfileName	Time	\
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	

	Score	Text	COUNT(*)
80638	5	I was recommended to try green tea extract to ...	5

```
In [6]: display['COUNT(*)'].sum()
```

```
Out [6]: 393063
```

### 3 Exploratory Data Analysis

#### 3.1 [2] 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 [7]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

```
Out [7]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	\
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	

	HelpfulnessDenominator	Score	Time \
0	2	5	1199577600
1	2	5	1199577600
2	2	5	1199577600
3	2	5	1199577600
4	2	5	1199577600

	Summary \
0	LOACKER QUADRATINI VANILLA WAFERS
1	LOACKER QUADRATINI VANILLA WAFERS
2	LOACKER QUADRATINI VANILLA WAFERS
3	LOACKER QUADRATINI VANILLA WAFERS
4	LOACKER QUADRATINI VANILLA WAFERS

	Text
0	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

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

```
In [9]: #Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first')
final.shape
```

```
Out[9]: (4986, 10)
```

```
In [10]: #Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[10]: 99.72
```

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

```
display.head()
```

```
Out[11]:      Id  ProductId      UserId      ProfileName \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

      HelpfulnessNumerator  HelpfulnessDenominator  Score      Time \
0                        3                        1      5  1224892800
1                        3                        2      4  1212883200

                        Summary \
0      Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                        Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

```
In [12]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [13]: #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()
```

```
(4986, 10)
```

```
Out[13]: 1      4178
0        808
Name: Score, dtype: int64
```

## 4 [3]. Text Preprocessing.

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 [14]: # 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.
=====
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The be
=====
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
=====
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
=====
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_1500 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
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)
```

Why is this \$[...] when the same product is available for \$[...] here? />The Victor M380 and M380S are the best value for the money. I recently tried this flavor/brand and was surprised at how delicious these chips are. The best I have ever had. I love to order my coffee on amazon. easy and shows up quickly. This k cup is great coffee. dca

```
In [17]: # 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 [18]: sent_1500 = decontracted(sent_1500)
        print(sent_1500)
        print("="*50)
```

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other was  
=====

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
        sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
        print(sent_0)
```

Why is this \$[...] when the same product is available for \$[...] here?<br /> /><br />The Victor

```
In [20]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
        sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
        print(sent_1500)
```

Wow So far two two star reviews One obviously had no idea what they were ordering the other was

```
In [21]: # 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 reumoved in the 1st step

        stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'that',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # Combining all the above stundents
        from tqdm import tqdm
        preprocessed_reviews = []
        # tqdm is for printing the status bar
        for sentence in tqdm(final['Text'].values):
```

```

sentence = re.sub(r"http\S+", "", sentence)
sentence = BeautifulSoup(sentence, 'lxml').get_text()
sentence = decontracted(sentence)
sentence = re.sub("\S*\d\S*", "", sentence).strip()
sentence = re.sub('[^A-Za-z]+', ' ', sentence)
# https://gist.github.com/sebleier/554280
sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
preprocessed_reviews.append(sentence.strip())

```

100%|| 4986/4986 [00:13<00:00, 375.61it/s]

In [23]: preprocessed\_reviews[1500]

Out[23]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey'

### [3.2] Preprocess Summary

In [24]: *## Similarly you can do preprocessing for review summary also.*

## 5 [4] Featurization

### 5.1 [4.1] BAG OF WORDS

In [25]: *#BoW*

```

count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
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])

```

```

some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdomina
=====
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997

```

### 5.2 [4.2] Bi-Grams and 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))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/mod

```

```

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
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_counts.get_shape()[0])

```

```

the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer (4986, 3144)
the number of unique words including both unigrams and bigrams 3144

```

### 5.3 [4.3] TF-IDF

```

In [27]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names())
print('='*50)

```

```

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
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_shape()[0])

```

```

some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
=====

```

```

the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (4986, 3144)
the number of unique words including both unigrams and bigrams 3144

```

### 5.4 [4.4] Word2Vec

```

In [28]: # Train your own Word2Vec model using your own text corpus

```

```

i=0
list_of_sentence=[]
for sentence in preprocessed_reviews:
    list_of_sentence.append(sentence.split())

```

```

In [29]: # 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
# or change these variable according to your need

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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentence,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

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.b
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True,

[('excellent', 0.995392918586731), ('think', 0.9952558279037476), ('anything', 0.9951905608177)
=====
[('wow', 0.9995905160903931), ('awesome', 0.999480128288269), ('soda', 0.9993873834609985), ('

In [30]: w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

number of words that occurred minimum 5 times 3817
sample words ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby

```

## 5.5 [4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

### [4.4.1.1] Avg W2v

```

In [31]: # 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 tqdm(list_of_sentence): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
    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]))

```

100%| 4986/4986 [00:19<00:00, 251.49it/s]

4986

50

#### [4.4.1.2] TFIDF weighted W2v

```

In [32]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

In [33]: # 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 cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sentence): # 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 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 corpus
            # sent.count(word) = tf value 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

```

100%|| 4986/4986 [02:15<00:00, 37.52it/s]

## 6 [5] Applying TSNE

<li> you need to plot 4 tsne plots with each of these feature set

<ol>

<li>Review text, preprocessed one converted into vectors using (BOW)</li>

<li>Review text, preprocessed one converted into vectors using (TFIDF)</li>

<li>Review text, preprocessed one converted into vectors using (AVG W2v)</li>

<li>Review text, preprocessed one converted into vectors using (TFIDF W2v)</li>

</ol>

</li>

<li> <font color='blue'>Note 1: The TSNE accepts only dense matrices</font></li>

<li> <font color='blue'>Note 2: Consider only 5k to 6k data points </font></li>

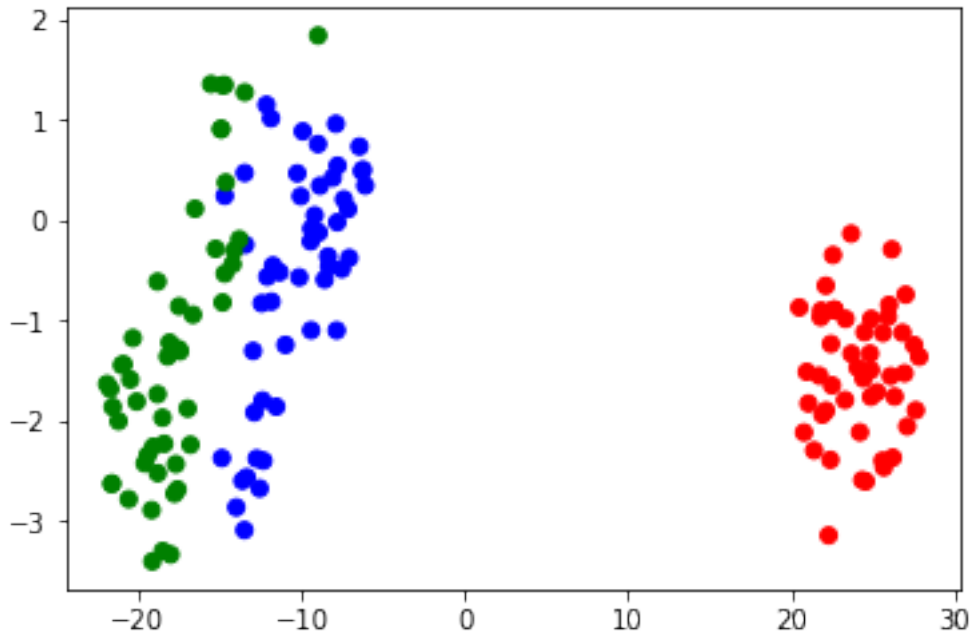
```
In [34]: import numpy as np
         from sklearn.manifold import TSNE
         from sklearn import datasets
         import pandas as pd
         import matplotlib.pyplot as plt

         iris = datasets.load_iris()
         x = iris['data']
         y = iris['target']

         tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)

         X_embedding = tsne.fit_transform(x)
         # if x is a sparse matrix you need to pass it as X_embedding = tsne.fit_transform(x.toarray())

         for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
         for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x', 'Dimension_y', 'Score'])
         colors = {0: 'red', 1: 'blue', 2: 'green'}
         plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score'])
         plt.show()
```



## 6.1 [5.1] Applying TNSE on Text BOW vectors

```
In [35]: # please write all the code with proper documentation, and proper titles for each sub.
# when you plot any graph make sure you use
# a. Title, that describes your plot, this will be very helpful to the reader
# b. Legends if needed
# c. X-axis label
# d. Y-axis label
l=final['Score']
labels = l.head(5000)
print("the shape of sample data = ", final['Text'].shape)
```

the shape of sample data = (4986,)

```
In [40]: #Standardizing the data# Data-preprocessing: Standardizing the data

from sklearn.preprocessing import MaxAbsScaler
standardized_data = MaxAbsScaler().fit_transform(final_counts).todense()
print(standardized_data.shape)
```

(4986, 12997)

```
In [45]: # TSNE
import seaborn as sn
```

```

from sklearn.manifold import TSNE
from time import time

# Picking the top 2k points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:1000,:]
labels_1000 = labels[0:1000]

model = TSNE(n_components=2, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

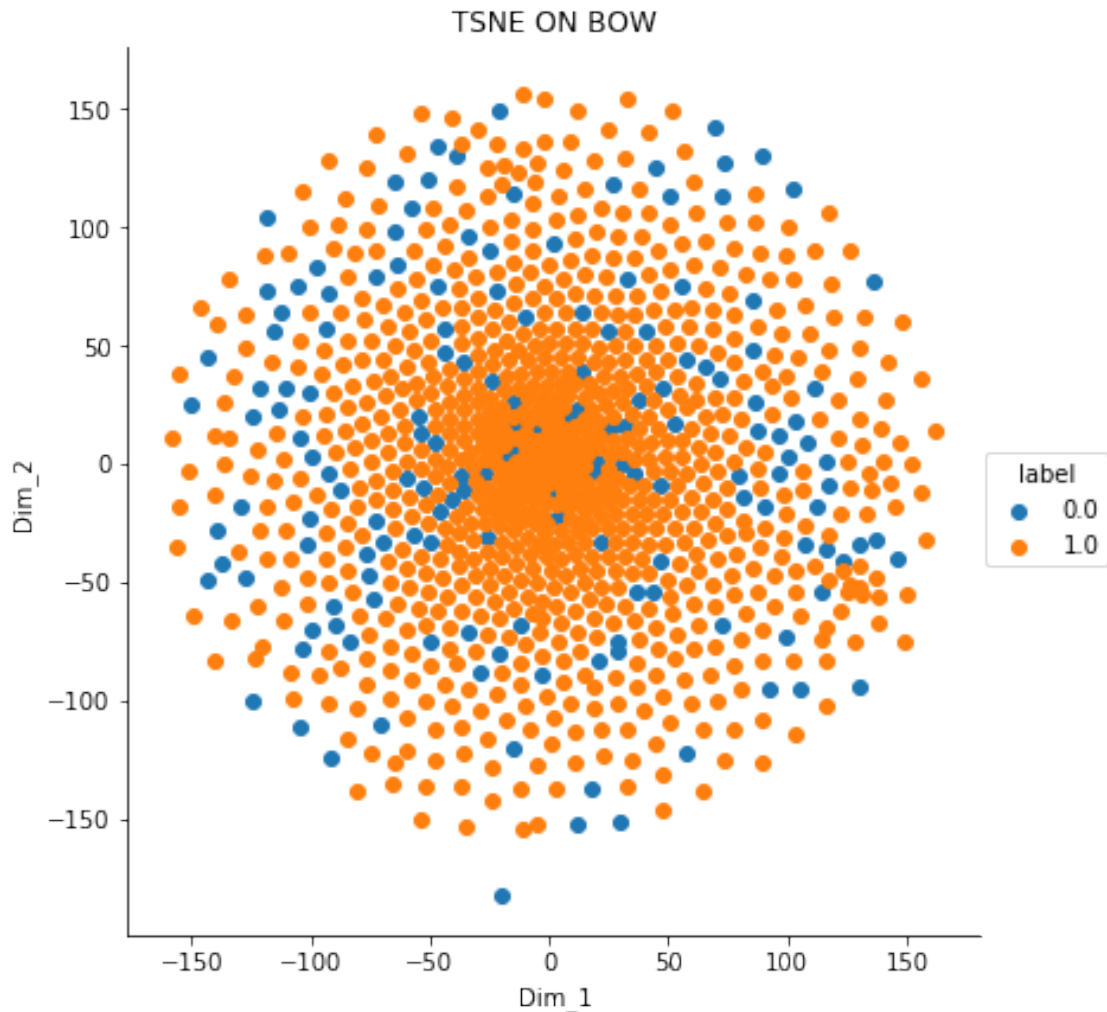
tsne_data = model.fit_transform(data_1000)

# creating a new data frame which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title("TSNE ON BOW")
plt.show()

```





## 6.2 [5.1] Applying TNSE on Text TFIDF vectors

```
In [42]: # please write all the code with proper documentation, and proper titles for each sub.
# when you plot any graph make sure you use
# a. Title, that describes your plot, this will be very helpful to the reader
# b. Legends if needed
# c. X-axis label
# d. Y-axis label

#Standardizing the data# Data-preprocessing: Standardizing the data

from sklearn.preprocessing import MaxAbsScaler
standardized_data = MaxAbsScaler().fit_transform(final_tf_idf).todense()
print(standardized_data.shape)
```

(4986, 3144)

```
In [48]: # TSNE
import seaborn as sn
from sklearn.manifold import TSNE
from time import time

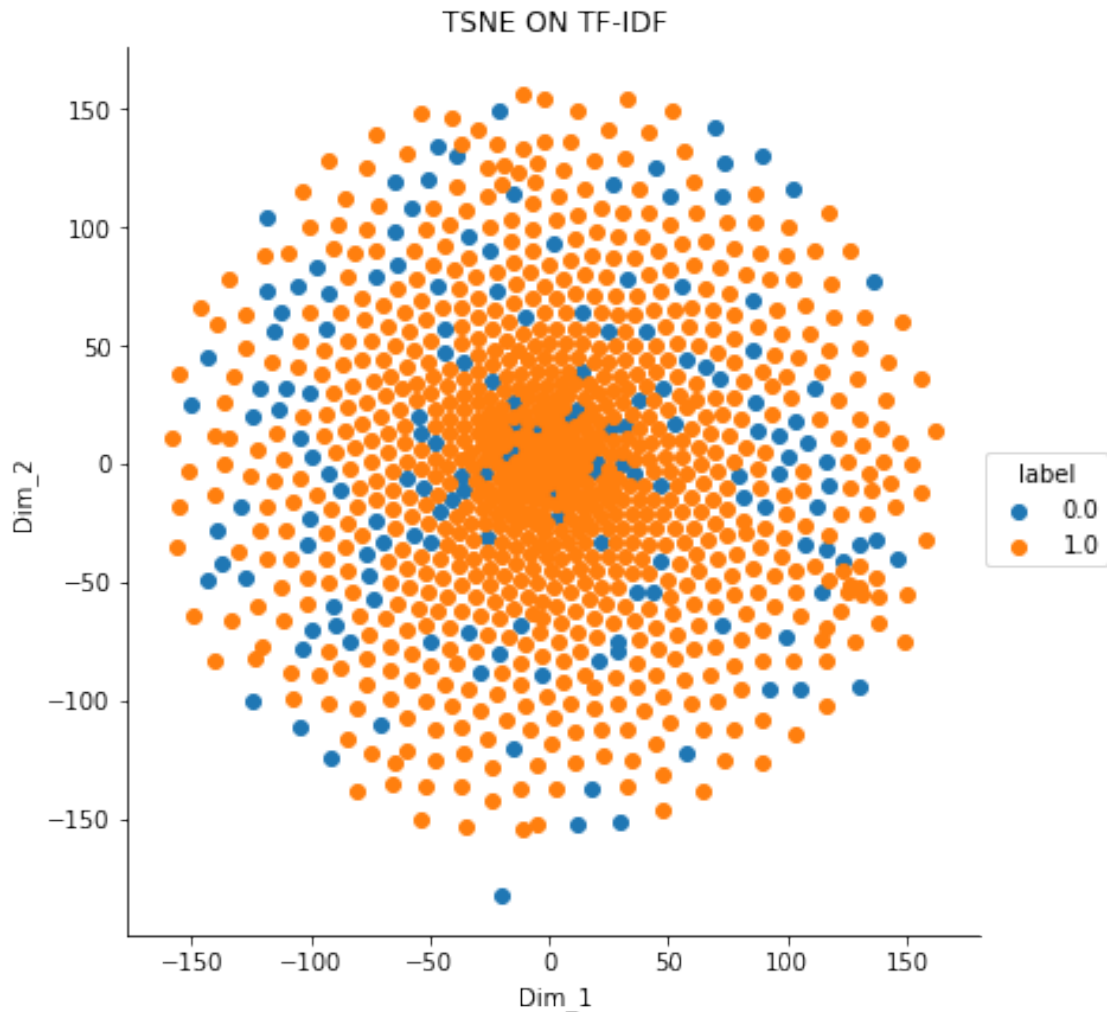
# Picking the top 1k points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:1000,:]
labels_1000 = labels[0:1000]

model = TSNE(n_components=2, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_1000)

# creating a new data frame which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title("TSNE ON TF-IDF")
plt.show()
```



### 6.3 [5.3] Applying TSNE on Text Avg W2V vectors

```
In [ ]: # please write all the code with proper documentation, and proper titles for each subs
        # when you plot any graph make sure you use
            # a. Title, that describes your plot, this will be very helpful to the reader
            # b. Legends if needed
            # c. X-axis label
            # d. Y-axis label

from sklearn.preprocessing import MaxAbsScaler
standardized_data = MaxAbsScaler().fit_transform(sent_vectors).todense()
print(standardized_data.shape)

In [47]: # TSNE
import seaborn as sn
from sklearn.manifold import TSNE
```

```

from time import time

# Picking the top 1k points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:1000,:]
labels_1000 = labels[0:1000]

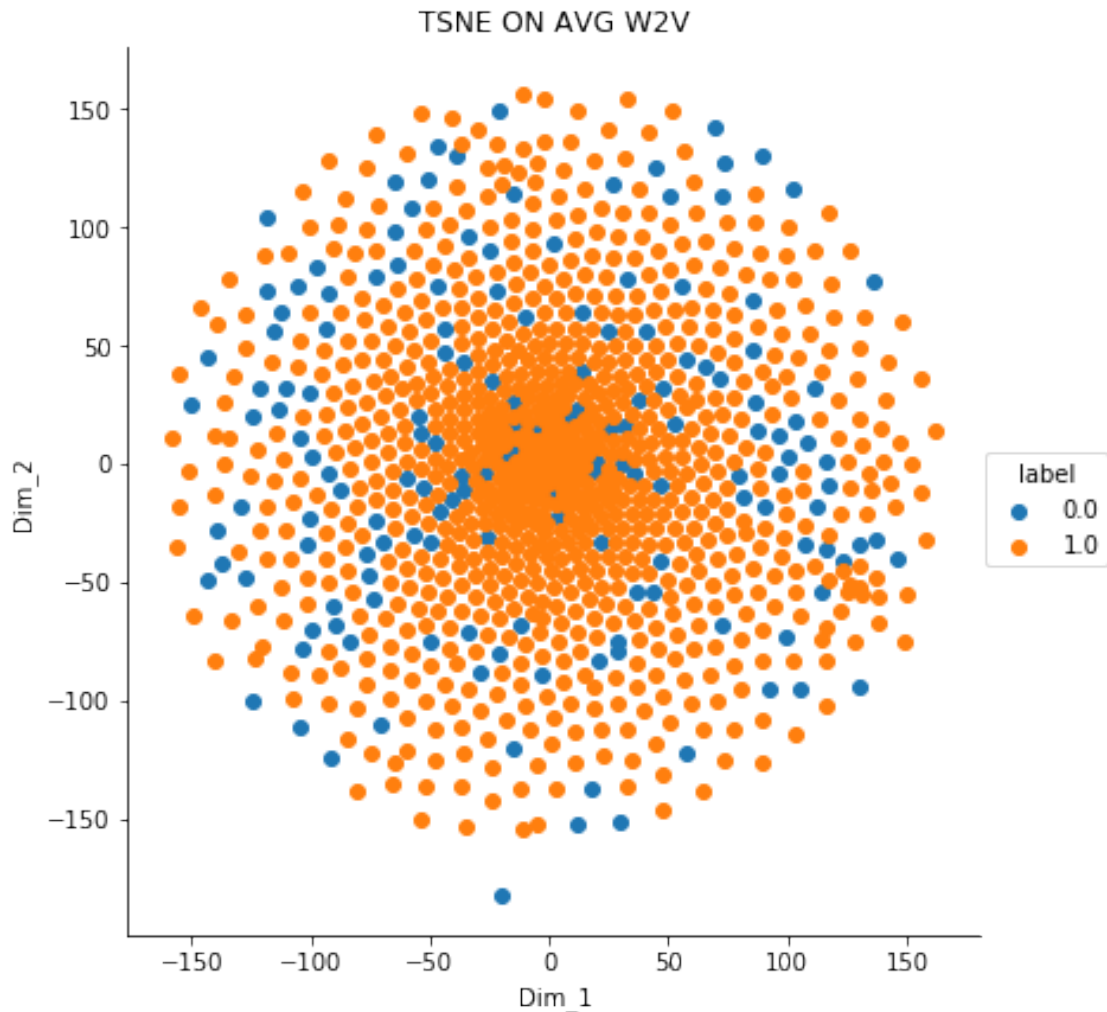
model = TSNE(n_components=2, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_1000)

# creating a new data frame which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title("TSNE ON AVG W2V")
plt.show()

```



## 6.4 [5.4] Applying TNSE on Text TFIDF weighted W2V vectors

```
In [ ]: # please write all the code with proper documentation, and proper titles for each subs
        # when you plot any graph make sure you use
            # a. Title, that describes your plot, this will be very helpful to the reader
            # b. Legends if needed
            # c. X-axis label
            # d. Y-axis label
```

```
from sklearn.preprocessing import MaxAbsScaler
standardized_data = MaxAbsScaler().fit_transform(tfidf_sent_vectors).toarray()
print(standardized_data.shape)
```

```
In [52]: # TSNE
```

```

import seaborn as sn
from sklearn.manifold import TSNE
from time import time

# Picking the top 1k points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:1000,:]
labels_1000 = labels[0:1000]

model = TSNE(n_components=2, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_1000)

# creating a new data frame which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.title("TSNE ON TFIDF-W2V")
plt.show()

```