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\sujpanda\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarni
ng: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
# 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 ne
gative 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 (525814, 10)
```

Out[2]:

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

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

Out[3]:

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

In [4]:
```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inp
lace=False, kind='quicksort', na_position='last')
```

In [5]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[5]:
```
(364173, 10)
```

In [6]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[6]:
```
69.25890143662969
```

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

display.head()
```

Out[7]:

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

In [8]:
```
    #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()
```

```
(364173, 10)
```

Out[8]:
```
1    307063
0     57110
Name: Score, dtype: int64
```

Text preprocessing

In [9]:

```
# 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 poin
t, I consider myself a connoisseur of children's books and this is one of the
best. Santa Clause put this under the tree. Since then, we've read it perpetu
ally and he loves it.<br /><br />First, this book taught him the months of th
e year.<br /><br />Second, it's a pleasure to read. Well suited to 1.5 y/o ol
d to 4+.<br /><br />Very few children's books are worth owning. Most should b
e borrowed from the library. This book, however, deserves a permanent spot on
your shelf. Sendak's best.
```

In [10]:

```
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 spe
cial characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

```
{'don', 'if', 'aren', "isn't", 'itself', "hadn't", 'over', 'at', 'did', 'd',
"shan't", 'them', 'but', 'further', 'doing', 'down', 'herself', "aren't", 'mi
ghtn', 'as', 'from', 'ain', 's', 'have', 'between', "you'd", 'wasn', 'for',
'they', 'won', 'be', 'having', 'each', 'when', 'my', 'isn', 'very', 'now', 'a
nd', 'he', 'it', 'is', 'yours', 'this', "don't", 'can', 'after', 'same', 'suc
h', 'your', 'hasn', 'was', 'shouldn', 'ours', 'off', 'hers', 'both', "were
n't", 're', 'most', "mustn't", 'by', 'below', 'more', 'being', 'will', "you'v
e", 'until', 'in', 'than', 'who', 'the', 'me', 'been', 'wouldn', 't', 'how',
'other', 'an', "wouldn't", "should've", 'were', 'here', 'while', 'm', 'up',
'haven', 'a', 'once', "you're", 'so', 'does', 'doesn', 'too', "couldn't", 'we
ren', 'we', 'all', 'am', 'just', 'needn', 'yourself', 'himself', "shouldn't",
'few', 'shan', 'then', 'again', 'through', "didn't", 'her', 'no', "needn't",
'some', 'ma', 'i', 'into', 'should', 'ourselves', 'which', 'above', 'there',
'about', 'before', 've', "wasn't", 'she', 'to', 'o', "it's", "she's", 'what',
'with', 'mustn', 'that', 'or', 'during', 'these', 'him', "haven't", 'on', 'ha
dn', 'theirs', 'only', 'his', 'under', 'because', 'out', 'not', 'll', 'own',
'against', 'yourselves', "hasn't", 'didn', 'their', "mightn't", 'our', 'its',
'are', 'whom', 'has', 'of', "you'll", 'couldn', 'do', 'nor', 'themselves', 'w
hy', 'you', "that'll", "doesn't", 'any', 'y', 'had', 'where', "won't", 'thos
e', 'myself'}
************************************
tasti
```

In [11]:

```
#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.
if not os.path.isfile('final.sqlite'):
    final_string=[]
    all_positive_words=[] # store words from +ve reviews here
    all_negative_words=[] # store words from -ve reviews here.
    for i, sent in enumerate(tqdm(final['Text'].values)):
        filtered_sentence=[]
        #print(sent);
        sent=cleanhtml(sent) # remove HTMl tags
        for w in sent.split():
            # we have used cleanpunc(w).split(), one more split function here
 because consider w="abc.def", cleanpunc(w) will return "abc def"
            # if we dont use .split() function then we will be considring "abc
 def" as a single word, but if you use .split() function we will get "abc", "d
ef"
            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] == 1:
                            all_positive_words.append(s) #list of all words us
ed to describe positive reviews
                        if(final['Score'].values)[i] == 0:
                            all_negative_words.append(s) #list of all words us
ed to describe negative reviews reviews
        str1 = b" ".join(filtered_sentence) #final string of cleaned words
        #print("***********************************************************
*********")
        final_string.append(str1)

    #############---- storing the data into .sqlite file ------###############
#########
    final['CleanedText']=final_string #adding a column of CleanedText which di
splays the data after pre-processing of the review
    final['CleanedText']=final['CleanedText'].str.decode("utf-8")
        # 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=None, chunksize=None, dtype=None)
    conn.close()


    with open('positive_words.pkl', 'wb') as f:
        pickle.dump(all_positive_words, f)
    with open('negitive_words.pkl', 'wb') as f:
        pickle.dump(all_negative_words, f)
```

In [12]:

```
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")
```

Bag of Words (BoW)

In [13]:

```
#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
```

In [18]:

```
final_10k_rows = final_counts[:3000]
print(final_10k_rows.shape[0] )
```

```
3000
```

TSNE for BOW

In [19]:

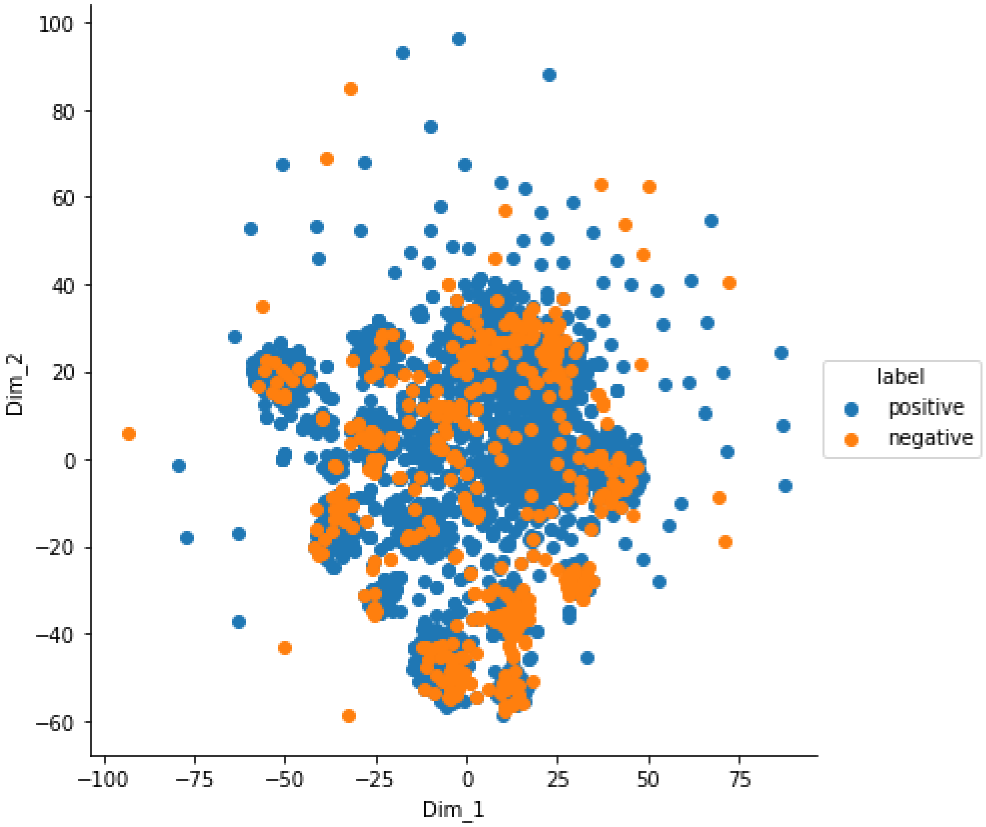
```
from sklearn.manifold import TSNE
```

In [20]:

```
X_tsne = TSNE(n_components=2, random_state=0).fit_transform(final_10k_rows.toa
rray())
print(X_tsne[:,1])
```

```
[  2.9658208  10.565228  -11.739106  ... -20.989239  -22.180576
 -24.916716 ]
```

In [22]:
```
import seaborn as sn
labels_3000 = final['Score'][:3000]
tsne_data = np.vstack((X_tsne.T, labels_3000)).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.show()
```



TF IDF

In [23]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final[:3000]['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_shape()[1])
```

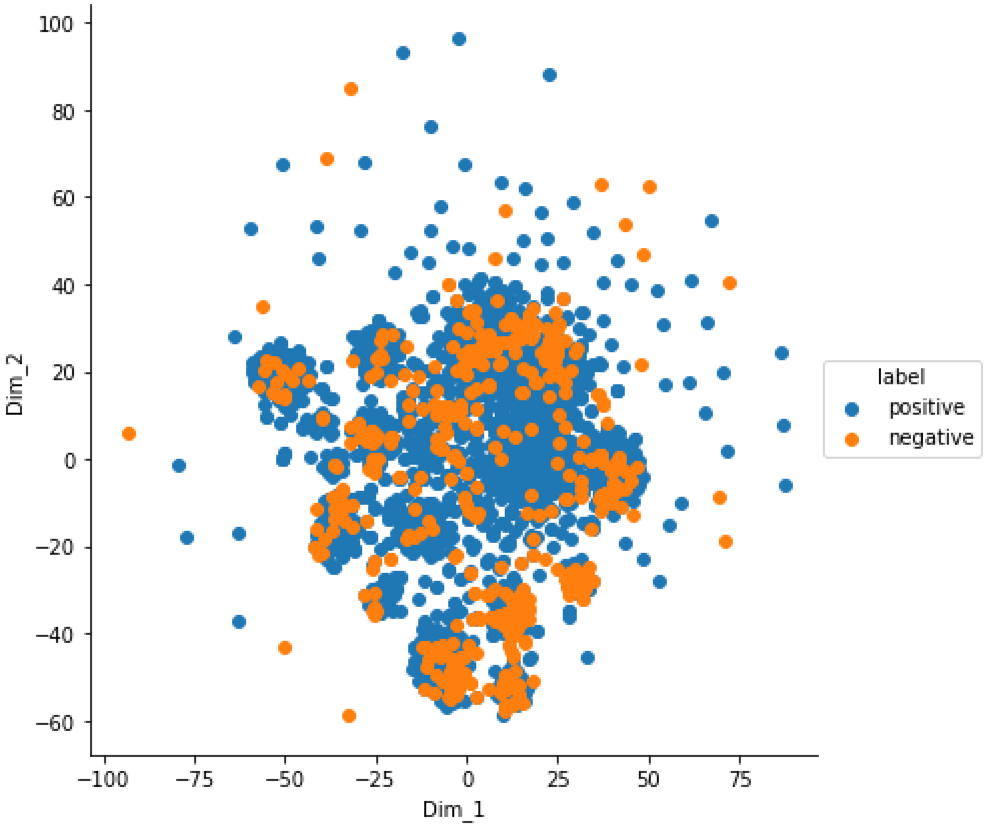
```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (3000, 102001)
the number of unique words including both unigrams and bigrams  102001
```

TSNE for TF IDF

In [24]:

```
X_tsne = TSNE(n_components=2, random_state=0).fit_transform(final_tf_idf.toarr
ay())
print(X_tsne[:,1])
labels_100= final['Score'][:3000]
tsne_data = np.vstack((X_tsne.T, labels_100)).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.show()
```

```
[ 8.594813  8.605626  9.278947 ... 24.001314 22.606585 23.230476]
```



Word to vec and avg word to vec

In [25]:

```
i=0
list_of_sent=[]
for sent in final[:3000]['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [26]:

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

```
witti littl book make son laugh loud recit car drive along alway sing refrain
hes learn whale india droop love new word book introduc silli classic book wi
ll bet son still abl recit memori colleg
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car', 'd
rive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whale', 'india',
'droop', 'love', 'new', 'word', 'book', 'introduc', 'silli', 'classic', 'boo
k', 'will', 'bet', 'son', 'still', 'abl', 'recit', 'memori', 'colleg']
```

In [27]:

```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [28]:

```
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  2762
sample words  ['littl', 'book', 'make', 'son', 'laugh', 'loud', 'car', 'driv
e', 'along', 'alway', 'sing', 'hes', 'learn', 'love', 'new', 'word', 'introdu
c', 'silli', 'classic', 'will', 'still', 'abl', 'memori', 'colleg', 'grew',
'read', 'sendak', 'watch', 'realli', 'movi', 'howev', 'miss', 'hard', 'cove
r', 'version', 'seem', 'kind', 'flimsi', 'take', 'two', 'hand', 'keep', 'pag
e', 'open', 'fun', 'way', 'children', 'month', 'year', 'poem']
```

In [29]:

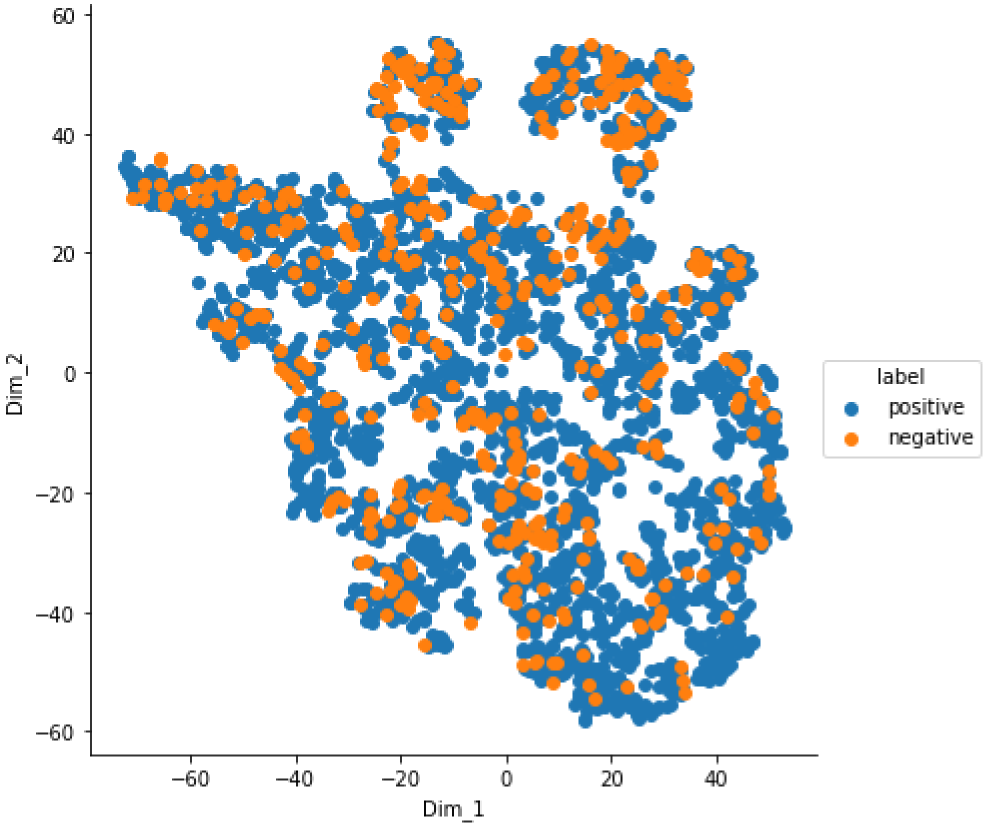
```
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in 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]))
```

```
100%|██████████| 3000/3000 [00:03<00:00, 873.09it/s]

3000
50
```

In [31]:

```
X_tsne = TSNE(n_components=2, random_state=0).fit_transform(sent_vectors)
print(X_tsne[:,1])
labels_100= final['Score'][:3000]
tsne_data = np.vstack((X_tsne.T, labels_100)).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.show()
```

```
[31.89499    11.162529   33.95997    ... 15.882422   21.818539
  0.84983313]
```



In [32]:

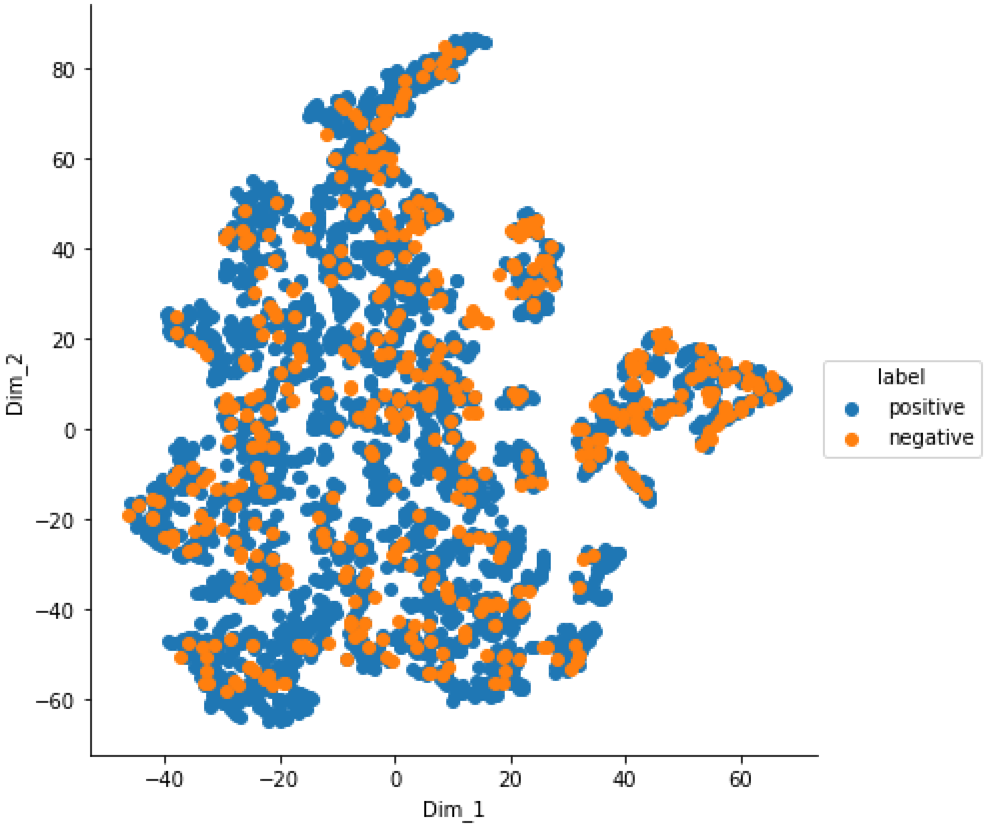
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final[:3000]['CleanedText'].values)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

TSNE for w2v(avg w2v)

```
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 list
         row=0;
         for sent in tqdm(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]
         #             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

         100%|██████████| 3000/3000 [00:04<00:00, 696.66it/s]
```

```
In [34]: X_tsne = TSNE(n_components=2, random_state=0).fit_transform(tfidf_sent_vectors
         )
         print(X_tsne[:,1])
         labels_100= final['Score'][:3000]
         tsne_data = np.vstack((X_tsne.T, labels_100)).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.show()
```

```
[ 65.6732     27.026335   82.143196  ...  23.911514   -6.9300694
 -51.299923 ]
```


Dim_2
Dim_1
label
● positive
● negative