amazon-fine-food-reviews-t-sne

February 18, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: 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).

We shall use the Score/Rating to determine whether a review is positive or negative. A rating of 4 or 5 is considered a positive review. A review of 1 or 2 is considered to be a negative one. A review of 3 is nuetral 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 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 [54]: %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.metrics import confusion_matrix
         from sklearn import metrics
         from sklearn.metrics import roc_curve, auc
         from nltk.stem.porter import PorterStemmer

         import 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
```

2 [1]. Reading Data

```
In [55]: # using the SQLite Table to read data.
         con = sqlite3.connect('../input/database.sqlite')
         #filtering only positive and negative reviews i.e.
         # not taking into consideration those reviews with Score=3
         # We'll be taking 5k data points given that t-sne is computationally expensive.

         filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50

         # Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative
         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
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[55]:

```
   Id          ...                I have bought several of the
0   1          ...                Product arrived labeled as J
1   2          ...                This is a confection that ha
2   3          ...

[3 rows x 10 columns]
```

In [56]:

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

In [57]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[57]:

```
              UserId   ...   COUNT(*)
0  #oc-R115TNMSPFT9I7  ...          2
1  #oc-R11D9D7SHXIJB9  ...          3
2  #oc-R11DNU2NBKQ23Z  ...          2
3  #oc-R11O5J5ZVQE25C  ...          3
4  #oc-R12KPBODL2B5ZD  ...          2

[5 rows x 7 columns]
```

In [5]:

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

Out[5]:

```
             UserId   ...   COUNT(*)
80638  AZY10LLTJ71NX  ...          5

[1 rows x 7 columns]
```

In [58]:

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

Out[58]: 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 [59]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND UserId="AR5J8UI46CURR"
         ORDER BY ProductID
         """, con)
         display.head()
```

```
Out[59]:        Id                    ...
         0   78445                    ...                     DELICIOUS WAFERS. I FIN
         1  138317                    ...                     DELICIOUS WAFERS. I FIN
         2  138277                    ...                     DELICIOUS WAFERS. I FIN
         3   73791                    ...                     DELICIOUS WAFERS. I FIN
         4  155049                    ...                     DELICIOUS WAFERS. I FIN

         [5 rows x 10 columns]
```

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

```
In [61]: #Deduplication of entries
         final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
         final.shape
```

```
Out[61]: (4986, 10)
```

```
In [62]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[62]: 99.72

In [63]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND HelpfulnessNumerator>HelpfulnessDenominator
""", con)
```

In [12]:
```
display.head()
```

Out[12]:
```
      Id              ...                  
0  44737              ...               It was almost a 'love at
1  64422              ...               My son loves spaghetti sc

[2 rows x 10 columns]
```

Observation:- It is seen that in two rows shown above the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible. Hence these two rows too are removed from calculations.

In [64]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [65]:
```
#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[65]:
```
positive    4178
negative     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 obsereved to be better than Porter Stemming)

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

```
In [66]: # 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.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
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 [67]: # 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_150 = 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 [68]: # 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 M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
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.This k cup is great coffee.  dca
```

In [70]:
```
# 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 [71]:
```
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 oth
==================================================
```

```
In [72]: #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 Victo
```

```
In [73]: #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 wa
```

```
In [75]: # 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', 'ourselve
                     "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', "
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
                     '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 [76]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 4986/4986 [00:01<00:00, 2808.34it/s]
```

In [77]:
```
len(preprocessed_reviews)
```

Out[77]:
```
4986
```

In [78]:
```
preprocessed_reviews[1000]
```

Out[78]:
```
'recently tried flavor brand surprised delicious chips best thing lot brown chips bsg
```

5 [4] Featurization and applying T-SNE models

5.1 [4.1] BAG OF WORDS

In [79]:
```
#BoW approach. Considering bigram features are considered as well.
# min_df is set to 10. max_df is not needed since we have already removed the stop wo
# More about min_df and max_df here : - https://stackoverflow.com/questions/27697766/
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_counts = count_vect.fit_transform(preprocessed_reviews)

print("the type of count vectorizer including unigrams and bigrams ",type(final_count
print("the shape of out text BOW vectorizer including unigrams and bigrams ",final_co
print("the number of unique words including unigrams and bigrams ", final_counts.get_
```

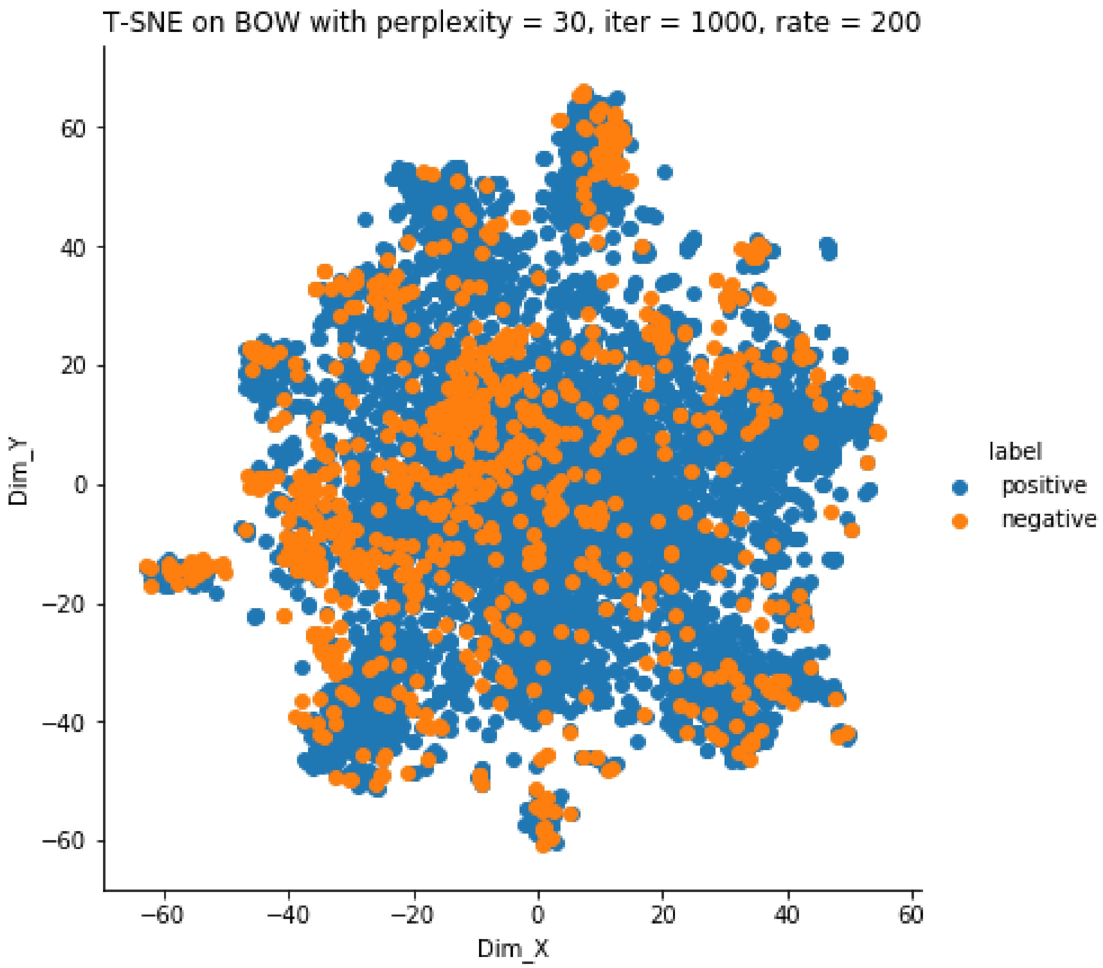
```
the type of count vectorizer including unigrams and bigrams  <class 'scipy.sparse.csr.csr_matr
the shape of out text BOW vectorizer including unigrams and bigrams  (4986, 3144)
the number of unique words including unigrams and bigrams  3144
```

5.2 [4.1.1] Applying T-SNE on BOW

In [80]:
```
%%time

# Before applying the model, convert the sparse matrix to dense matrix using truncate

# Standardize the data.
from sklearn.preprocessing import StandardScaler
final_standardized_data=StandardScaler(with_mean=False).fit_transform(final_counts)

# Apply truncated SVD setting the number of features as 50
# This will suppress some noise and speed up the computation of pairwise distances be
# Reference : https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE

from sklearn.decomposition import TruncatedSVD
tsvd_data = TruncatedSVD(n_components=50, random_state=0).fit_transform(final_standard

print(len(tsvd_data))
```

```
4986
CPU times: user 256 ms, sys: 4 ms, total: 260 ms
Wall time: 138 ms
```

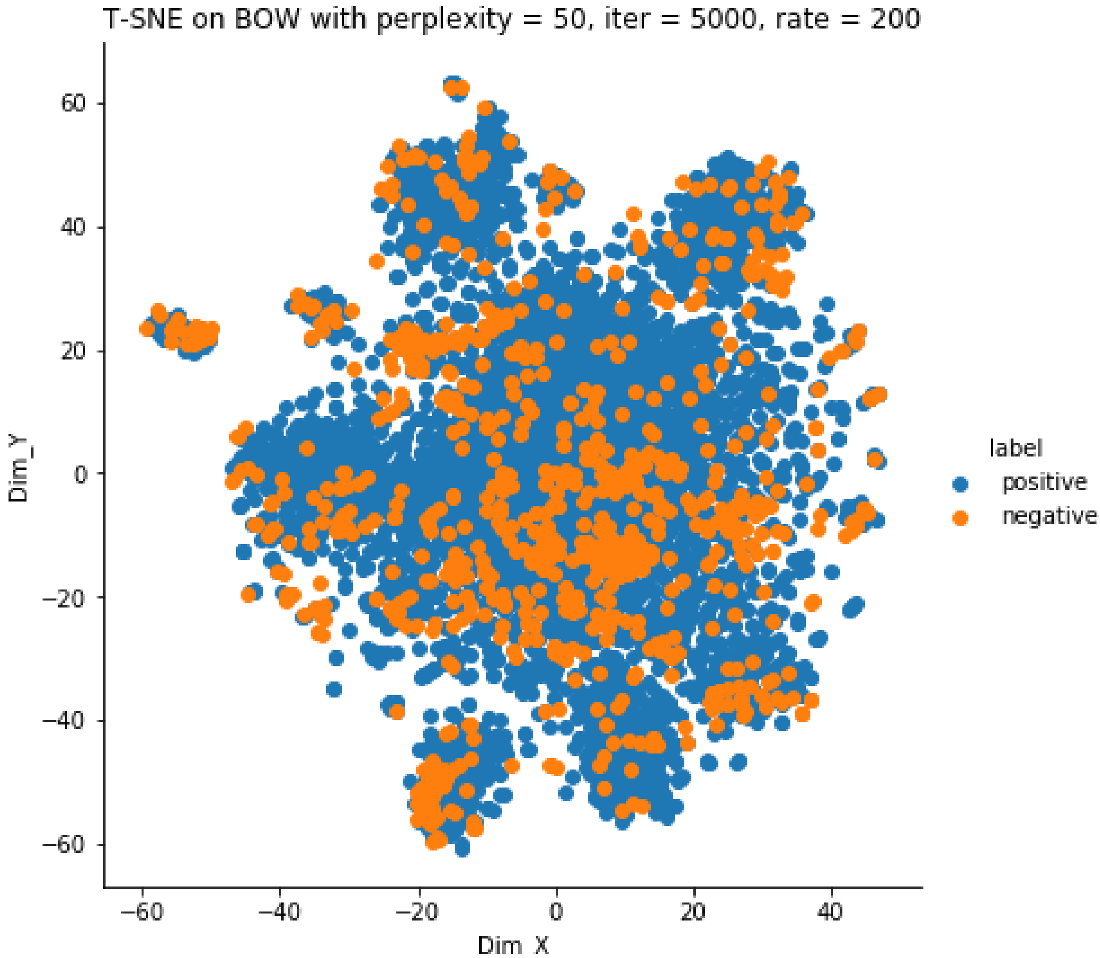
```
In [81]: %%time

         from sklearn.manifold import TSNE
         import matplotlib.pyplot as plt

         tsne_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 = tsne_model.fit_transform(tsvd_data)
         score = final['Score']

         # creating a new data frame which help us in ploting the result data
         tsne_data = np.vstack((tsne_data.T, score)).T
         tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

         # Plotting the result of tsne
         sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_l
         plt.title('T-SNE on BOW with perplexity = 30, iter = 1000, rate = 200')
         plt.show()
```




```
CPU times: user 52.1 s, sys: 456 ms, total: 52.6 s
Wall time: 51.9 s
```

In [82]:
```
#Adjust the hyper parameters of T-SNE in an effort to find a stable  model

# perplexity = 50
# default learning rate = 200
# Maximum number of iterations for the optimization = 5000 for getting a shot at a co
tsne_model_50_5000 = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=5000)

tsne_data = tsne_model_50_5000.fit_transform(tsvd_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))
```

```
# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_l
plt.title('T-SNE on BOW with perplexity = 50, iter = 5000, rate = 200')
plt.show()
```



5.3 Observations

1. T-SNE with the BOW approach was run to get different plots by adjusting its hyperparameters.
2. There is a huge amount of overlapping between positive and negative reviews and they cannot be visually separated by the BOW approach.

5.4 4.2 TF-IDF

```
In [83]: 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
```

```
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
```

5.5 4.2.1 Applying T-SNE on TF-IDF

In [84]:
```
%%time

# Before applying the model, convert the sparse matrix to dense matrix using truncate

# Standardize the data.
from sklearn.preprocessing import StandardScaler
final_standardized_data=StandardScaler(with_mean=False).fit_transform(final_tf_idf)

# Apply truncated SVD setting the number of features as 50
# This will suppress some noise and speed up the computation of pairwise distances be
# Reference : https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE

from sklearn.decomposition import TruncatedSVD
tsvd_data = TruncatedSVD(n_components=50, random_state=0).fit_transform(final_standar

print(len(tsvd_data))
```

```
4986
CPU times: user 276 ms, sys: 20 ms, total: 296 ms
Wall time: 150 ms
```

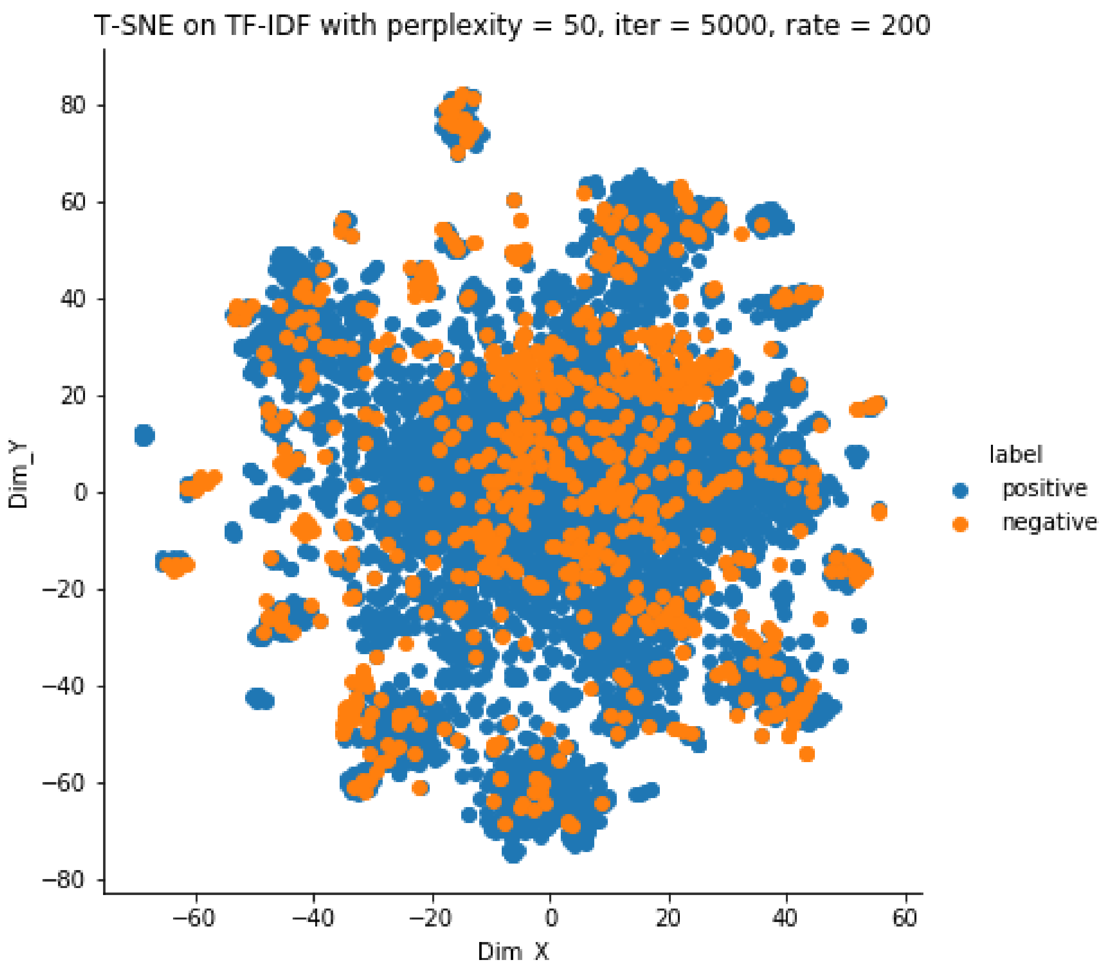
In [85]:
```
%%time

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

tsne_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 = tsne_model.fit_transform(tsvd_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_l
plt.title('T-SNE on TF-IDF with perplexity = 30, iter = 1000, rate = 200')
plt.show()
```


```
CPU times: user 52.1 s, sys: 448 ms, total: 52.6 s
Wall time: 51.9 s
```

In [86]: %%time

```
#Adjust the hyper parameters of T-SNE in an effort to find a stable  model

# perplexity = 50
# default learning rate = 200
# Maximum number of iterations for the optimization = 5000 for getting a shot at a co
tsne_model_50_5000 = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=5000)

tsne_data = tsne_model_50_5000.fit_transform(tsvd_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_
plt.title('T-SNE on TF-IDF with perplexity = 50, iter = 5000, rate = 200')
plt.show()
```



```
CPU times: user 4min 22s, sys: 516 ms, total: 4min 22s
Wall time: 4min 22s
```

5.6 Observations

1. Similar to that of T-SNE with BOW.

5.7 4.3 Word2Vec

```
In [87]: # 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 [90]: # min_count = 5 considers only words that occured atleast 5 times
         w2v_model=Word2Vec(list_of_sentence,min_count=5,size=50, workers=4)
         print(w2v_model.wv.most_similar('great'))
```

```
[('excellent', 0.9960392117500305), ('especially', 0.9959850311279297), ('calorie', 0.99584764
```

```
In [91]: 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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.8 [4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

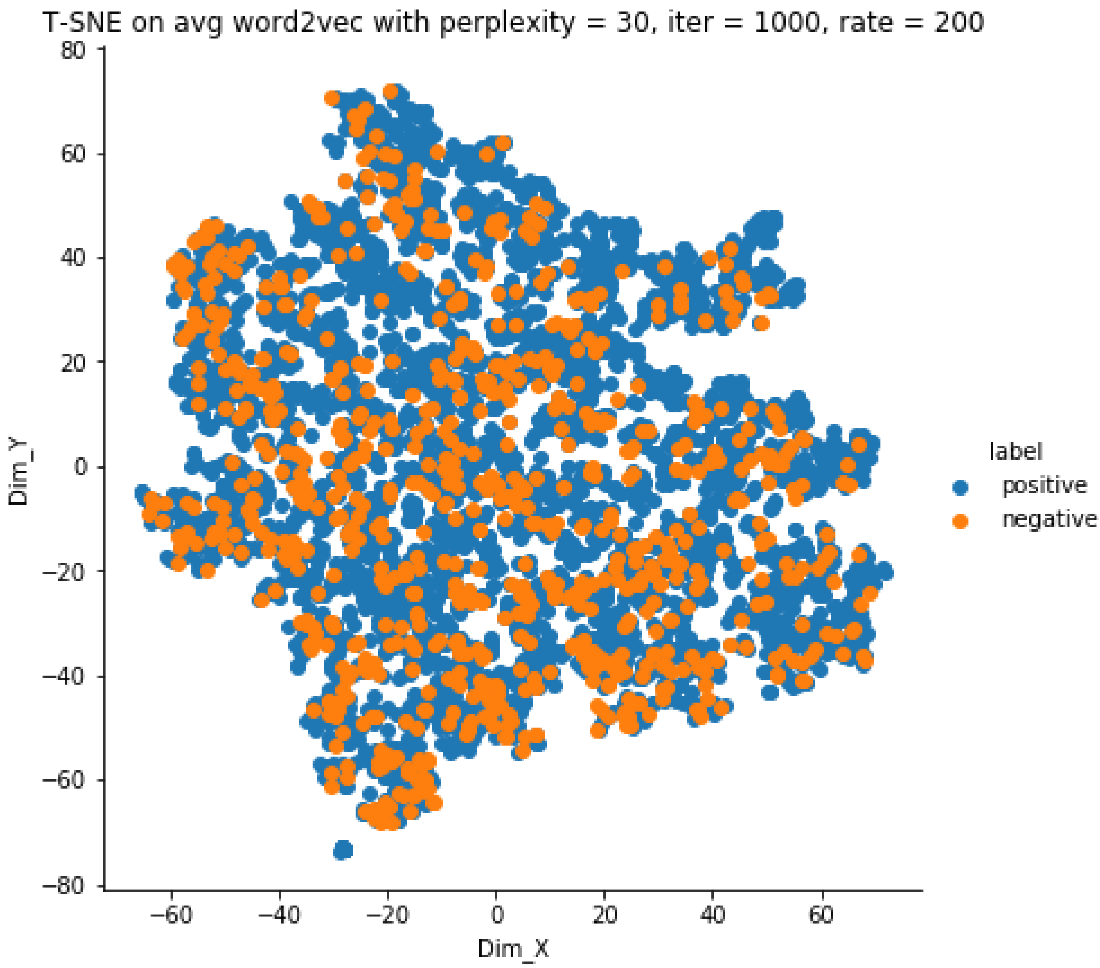
```
In [93]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(list_of_sentence): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors 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 [00:04<00:00, 1162.12it/s]

4986
50
```

5.9 4.4.2 Applying T-SNE on avg word2vec model

```
In [95]: %%time

         # Before applying the model, convert the sparse matrix to dense matrix using truncated

         # Standardize the data.
         from sklearn.preprocessing import StandardScaler
         final_standardized_data=StandardScaler(with_mean=False).fit_transform(sent_vectors)

CPU times: user 16 ms, sys: 0 ns, total: 16 ms
Wall time: 14.9 ms
```

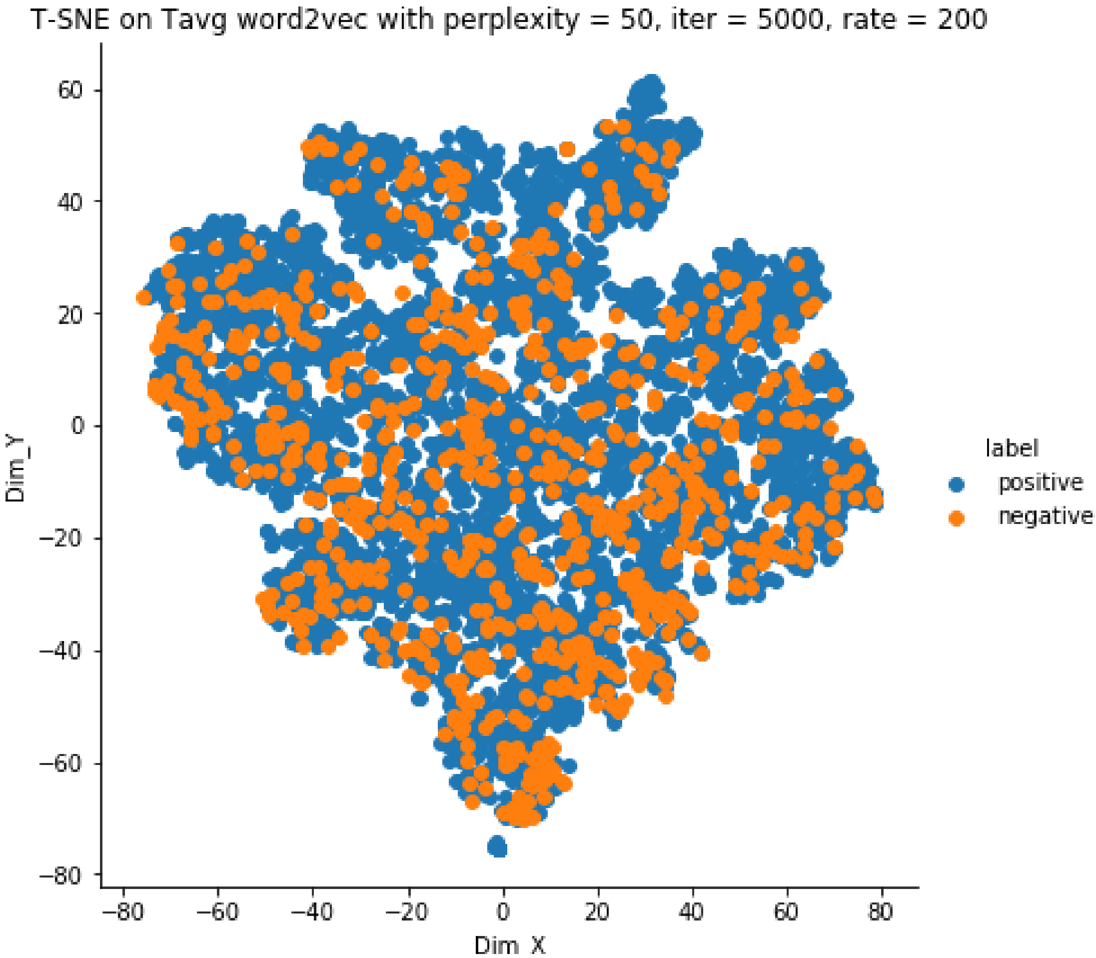
```
In [97]: %%time

         from sklearn.manifold import TSNE
         import matplotlib.pyplot as plt

         tsne_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 = tsne_model.fit_transform(final_standardized_data)

         # creating a new data frame which help us in ploting the result data
         tsne_data = np.vstack((tsne_data.T, score)).T
         tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

         # Plotting the result of tsne
         sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_l
         plt.title('T-SNE on avg word2vec with perplexity = 30, iter = 1000, rate = 200')
         plt.show()
```




```
CPU times: user 41.2 s, sys: 444 ms, total: 41.6 s
Wall time: 41 s
```

In [98]:
```
%%time

#Adjust the hyper parameters of T-SNE in an effort to find a stable  model

# perplexity = 50
# default learning rate = 200
# Maximum number of iterations for the optimization = 5000 for getting a shot at a co
tsne_model_50_5000 = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=5000)

tsne_data = tsne_model_50_5000.fit_transform(final_standardized_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
```

```
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_l
plt.title('T-SNE on Tavg word2vec with perplexity = 50, iter = 5000, rate = 200')
plt.show()
```




```
CPU times: user 3min 42s, sys: 568 ms, total: 3min 43s
Wall time: 3min 42s
```

5.10 Observations

1. Similar to that of T-SNE with BOW and TF-IDF.

5.11 4.5 TF-IDF weighed word2vec

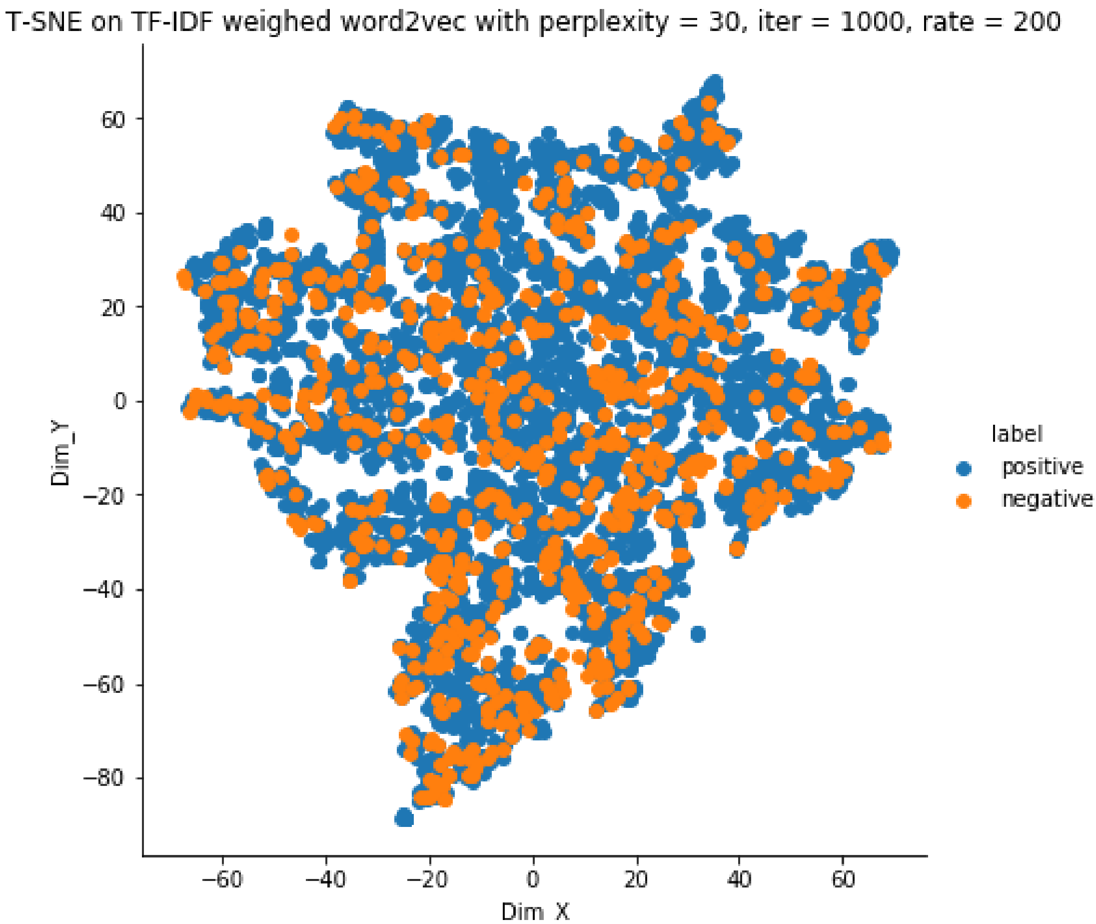
In [100]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

# 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 = tfid

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
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 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%|| 4986/4986 [00:30<00:00, 162.29it/s]
```

5.12 4.5.1 Applying T-SNE on TF-IDF weighed word2vec

In [102]:
```
%%time

# Standardize the data.
from sklearn.preprocessing import StandardScaler
final_standardized_data=StandardScaler(with_mean=False).fit_transform(tfidf_sent_vec

tsne_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
```

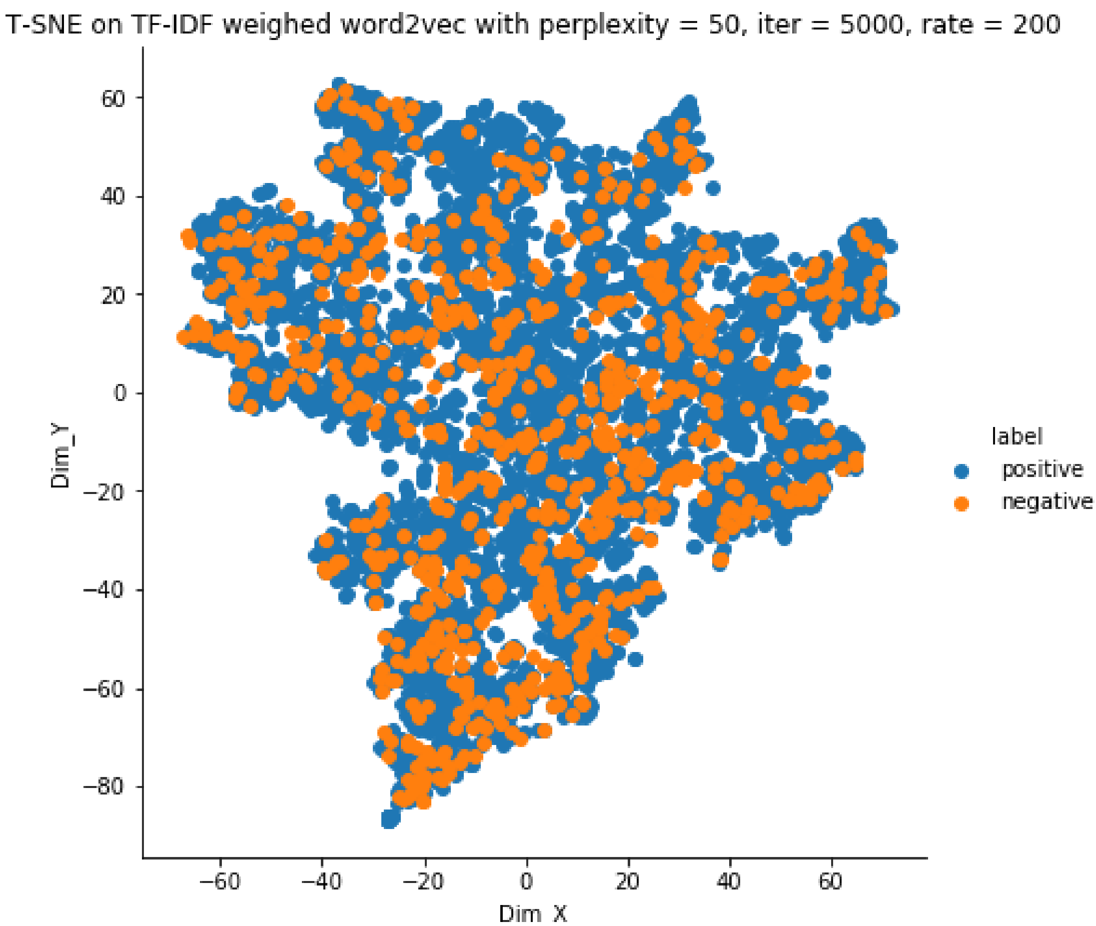
```
tsne_data = tsne_model.fit_transform(final_standardized_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_
plt.title('T-SNE on TF-IDF weighed word2vec with perplexity = 30, iter = 1000, rate =
plt.show()
```




```
CPU times: user 41.9 s, sys: 524 ms, total: 42.4 s
Wall time: 41.8 s
```

```
In [104]: %%time
```

```
#Adjust the hyper parameters of T-SNE in an effort to find a stable  model

# perplexity = 50
# default learning rate = 200
# Maximum number of iterations for the optimization = 5000 for getting a shot at a co
tsne_model_50_5000 = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=5000)

tsne_data = tsne_model_50_5000.fit_transform(final_standardized_data)

# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_X", "Dim_Y", "label"))

# Plotting the result of tsne
sns.FacetGrid(tsne_df, hue="label", height=6).map(plt.scatter, "Dim_X", "Dim_Y").add_
plt.title('T-SNE on TF-IDF weighed word2vec with perplexity = 50, iter = 5000, rate =
plt.show()
```



5.13 Conclusion(s)

1. The T-SNE algorithm was tried with different combinations of hyperparameters in order to find a stable model so as to develop an intuition of the datset.
2. However, a huge amount of overlapping between positive and negative reviews was observed and they could not be visually separated by the T-SNE model. Hence, T-SNE proved to be unsuccessful w.r.t this dataset.
3. T-SNE is a computationally expensive algorithm since it operates on dense matrices only and is unable to leverage the power of sparse matrices.