02 Amazon Fine Food Reviews Analysis_TSNE

March 19, 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 - 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).

[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 cosnidered a positive review. A review of 1 or 2 could be considered negative. 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 [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
```

2 [1]. Reading Data

In [2]:
```
# using the SQLite Table to read data.
con = sqlite3.connect('./Data/database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500
# for tsne assignment you can take 5k data points

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

```python
# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rat
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                           dll pa
        2   3  B000LQOCH0   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...
```

```python
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  B005ZBZLT4                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
```

```
3  #oc-R11O5J5ZVQE25C  B005HG9ESG          Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBEV0   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  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

       Score                                               Text  COUNT(*)
80638      5  I bought this 6 pack because for the price tha...         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 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 [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='f
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 calcualtions

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 obsereved 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.co
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The best
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the othe
==================================================
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_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 [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-t
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 M50
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The best
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the othe
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dcaf
```

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 othe
==================================================
```

```
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 want
```

```
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 revmoved 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', 't
                     'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                     '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', 'ov
                     '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", 'no
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
                     "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                     "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                     '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 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 stopwor
            preprocessed_reviews.append(sentance.strip())
```

```
100%|| 4986/4986 [00:01<00:00, 3181.77it/s]
```

```
In [23]: preprocessed_reviews[1500]
```

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

[3.2] Preprocess Summary

```
In [24]: ## Similartly 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', 'abdominal'
==================================================
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/modul
         # 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_c
```

```
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.g
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get', 'a
==================================================
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_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.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 varible 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 occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,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.bin
        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, to
```

```
[('alternative', 0.9929664731025696), ('want', 0.9927080869674683), ('excellent', 0.992087841033
==================================================
[('japanese', 0.9994041323661804), ('whatever', 0.9993966221809387), ('idea', 0.999384880065918)
```

In [30]:
```
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',
```

5.5 [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_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in 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, 1156.55it/s]

4986
50
```

In [32]:
```
sent_vectors[0]
```

Out[32]:
```
array([-0.2138532 ,  0.15966899,  0.07919481,  0.0558268 ,  0.38863745,
        0.07218608, -0.18779394, -0.20732247, -0.18082569, -0.26532944,
       -0.11265778,  0.31463975,  0.10026063, -0.74324407, -0.0418039 ,
       -0.21748892,  0.00335245,  0.24990973, -0.20649971,  0.15069421,
        0.17509579,  0.3513013 ,  0.25747652,  0.22244896, -0.12631988,
        0.13581549, -0.23051577,  0.28687755,  0.06848181,  0.64201889,
       -0.05859432, -0.34587256,  0.99790123,  0.34754718,  0.28295872,
       -0.2796431 ,  0.57256645, -0.29310036, -0.19692206, -0.11709556,
       -0.17686607, -0.77908973,  0.2925914 , -0.02015579,  0.70753166,
        0.01808066, -0.25909219,  0.40537512,  0.48079972, -0.30777761])
```

[4.4.1.2] TFIDF weighted W2v

In [33]:
```
# 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 [34]:
```
# 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 lis
row=0;
for sent in tqdm(list_of_sentance): # 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
```

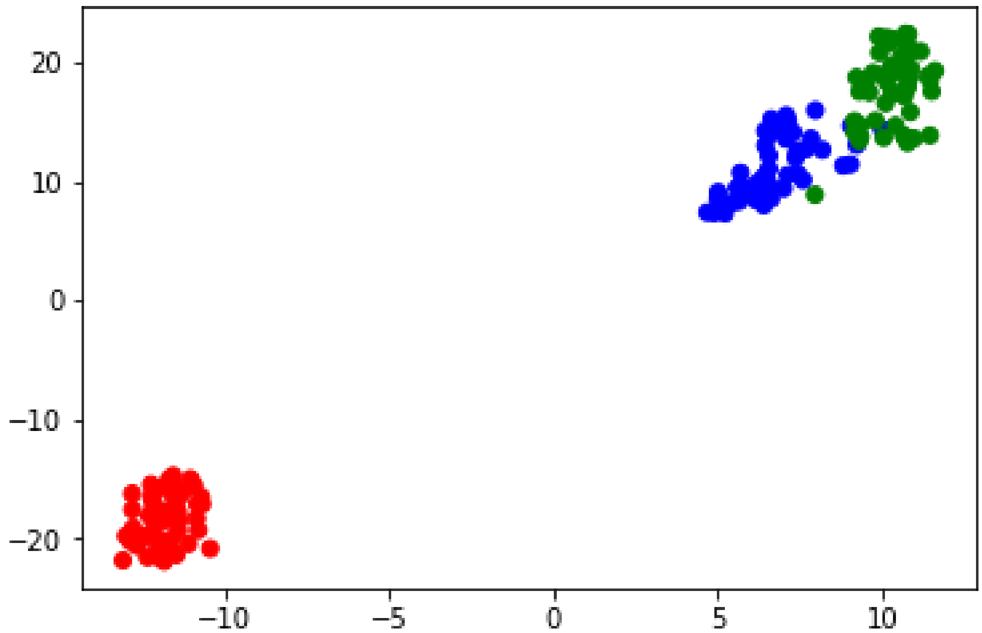
```
100%|| 4986/4986 [00:26<00:00, 189.91it/s]
```

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 [35]: # https://github.com/pavlin-policar/fastTSNE you can try this also, this version is lit
         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.toa

         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['Scor
plt.show()
```



6.1 Function to plot TSNE

```
In [36]: def get_tsne_plot(features, labels, featurization_method):
             """
             This function plot the t-SNE lower dimensional represntation of features into 2D.
             It runs multiple rounds (3 RUNS) to ensure that the visualizations are consistant.
             It runs with different perplexity values 10, 20, 30, 40
             """

             # run tnse for multiple rounds (3 times 1, 2, 3) with different perplexity values (
             param_list = [(1,10,), (2, 10,), (3, 10), (1,20,), (2, 20,), (3, 20), (1, 30,), (2,
                           (1,40,), (2, 40,), (3, 40)]

             # run tnse multiple rounds with different perplexity values
             for run_id, per in param_list:
                 # create an object of tnse with given parameters
                 tnse_obj = TSNE(n_components=2, perplexity=per, n_iter=800, learning_rate=200)
                 # reduce dimension to 2D for visualization
                 embedded_features = tnse_obj.fit_transform(features)

                 # create a dataframe for tnse plot
```

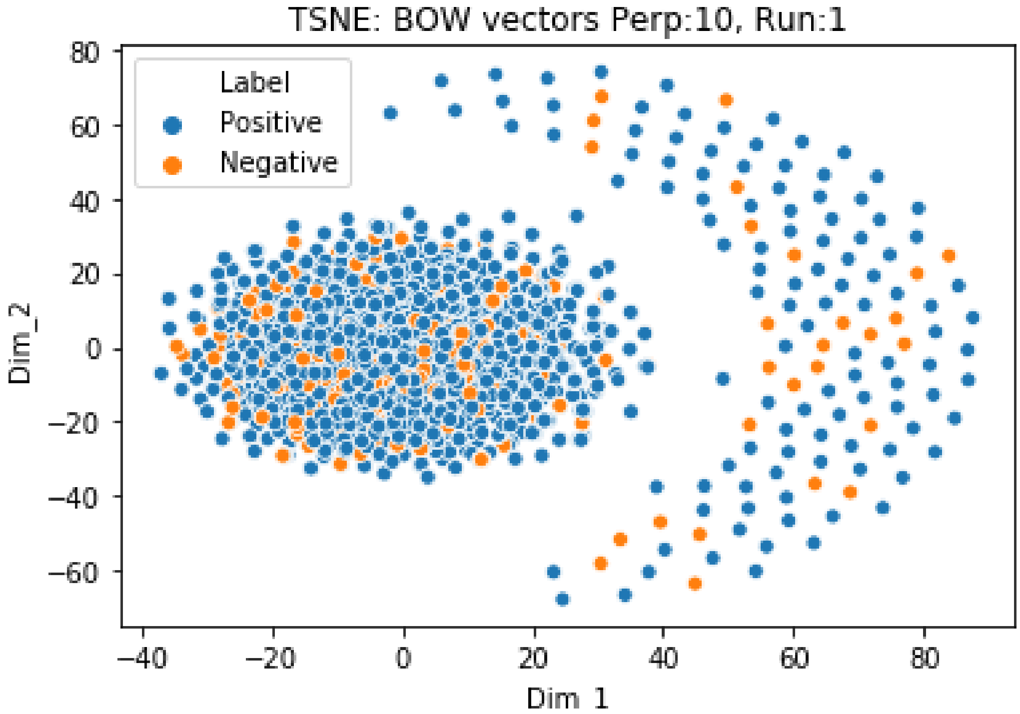
```
        tnse_df = pd.DataFrame(embedded_features, columns=['Dim_1', 'Dim_2'])
        tnse_df['Label'] = labels

        # plot the tsne using seaborn scatterplot
        sns.scatterplot(x='Dim_1', y='Dim_2', hue='Label', data=tnse_df)
        plt.title('TSNE: %s Perp:%d, Run:%d'%(featurization_method, per, run_id))
        plt.show()
```

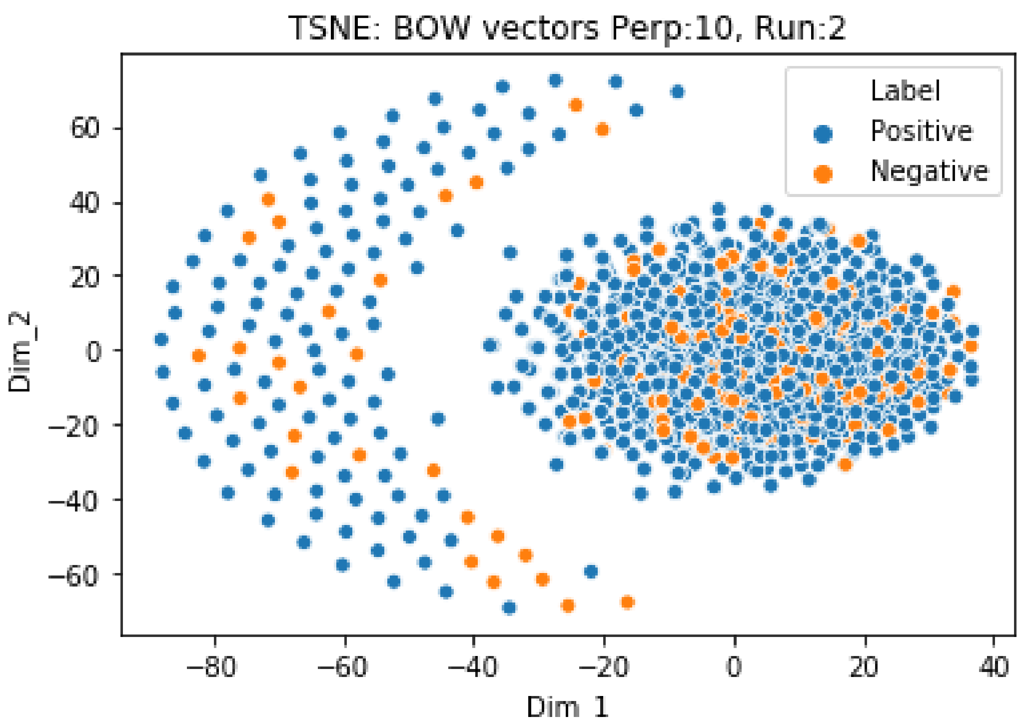
In [37]:
```
# a dictionary for assigning categorical type value to classes 0, 1
label_dict = {0 : 'Negative', 1 : 'Positive'}
from sklearn.preprocessing import StandardScaler
scaler_object = StandardScaler()
```

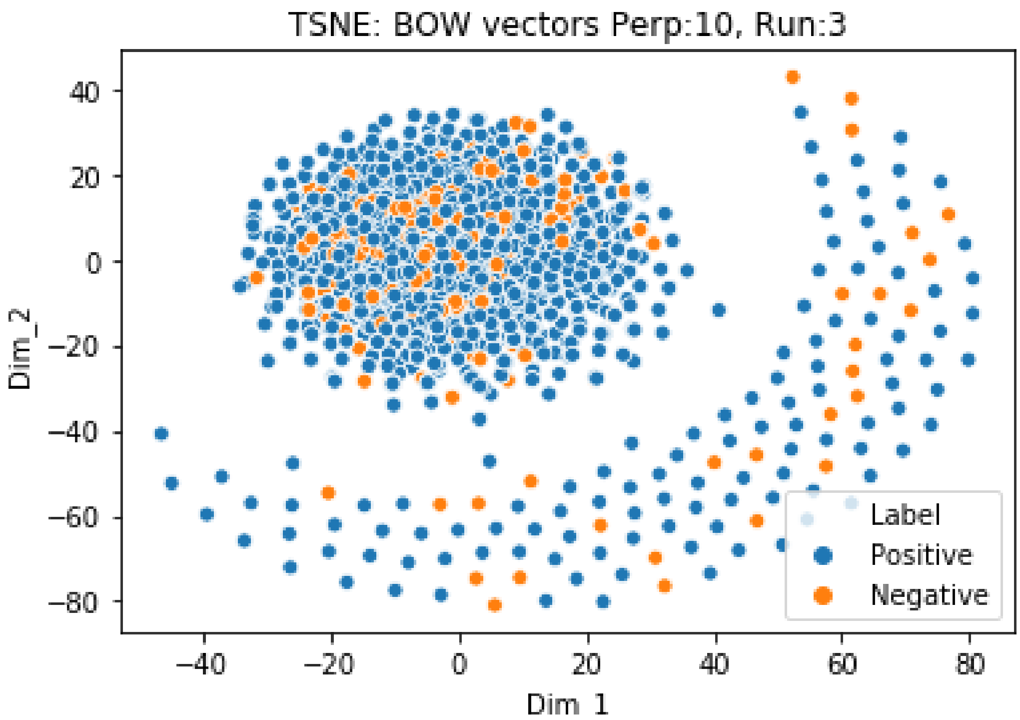
6.2 [5.1] Applying TNSE on Text BOW vectors

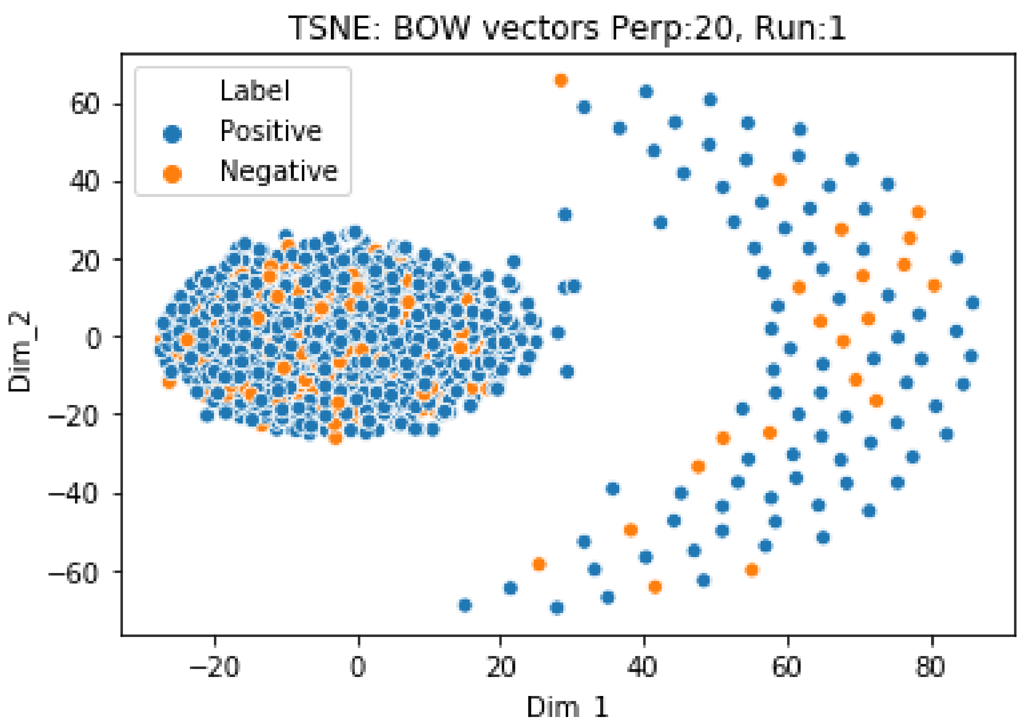
In [38]:
```
max_data_points = 3000
features_t = final_bigram_counts[0:max_data_points]
features_t = features_t.toarray()
# standardize the data
features_t = scaler_object.fit_transform(features_t)
# get lables
lables_t = final['Score'].tolist()[0:max_data_points]
lables_t = [label_dict[item] for item in lables_t]
# plot it
get_tsne_plot(features_t, lables_t, 'BOW vectors')
```

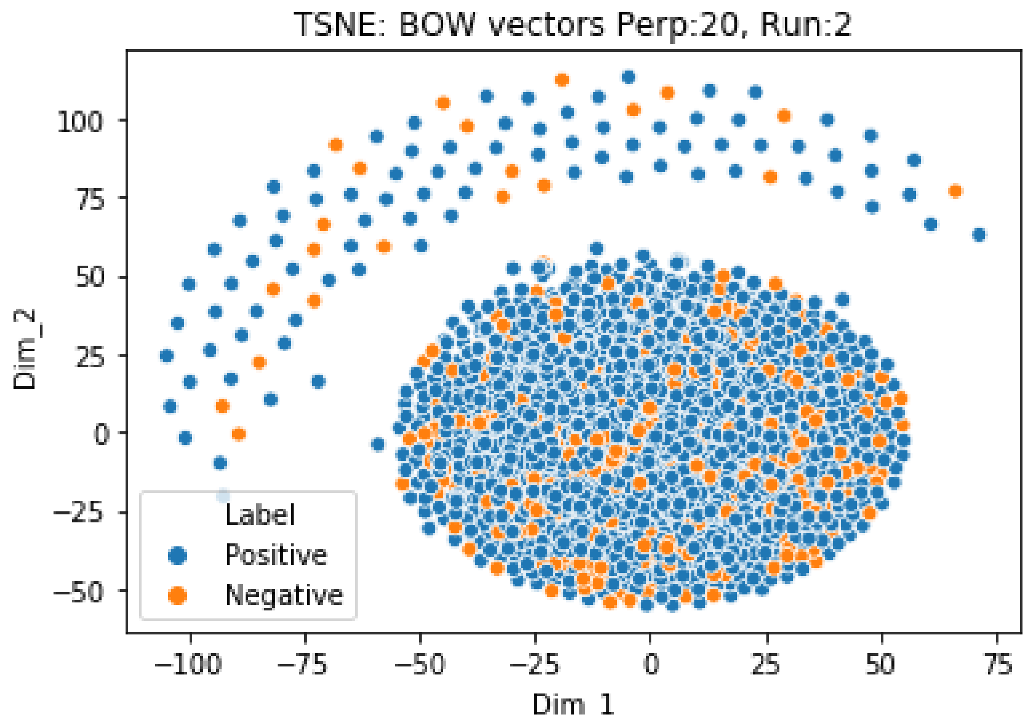
```
/home/nisheels/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConver
  warnings.warn(msg, DataConversionWarning)
/home/nisheels/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConver
  warnings.warn(msg, DataConversionWarning)
```

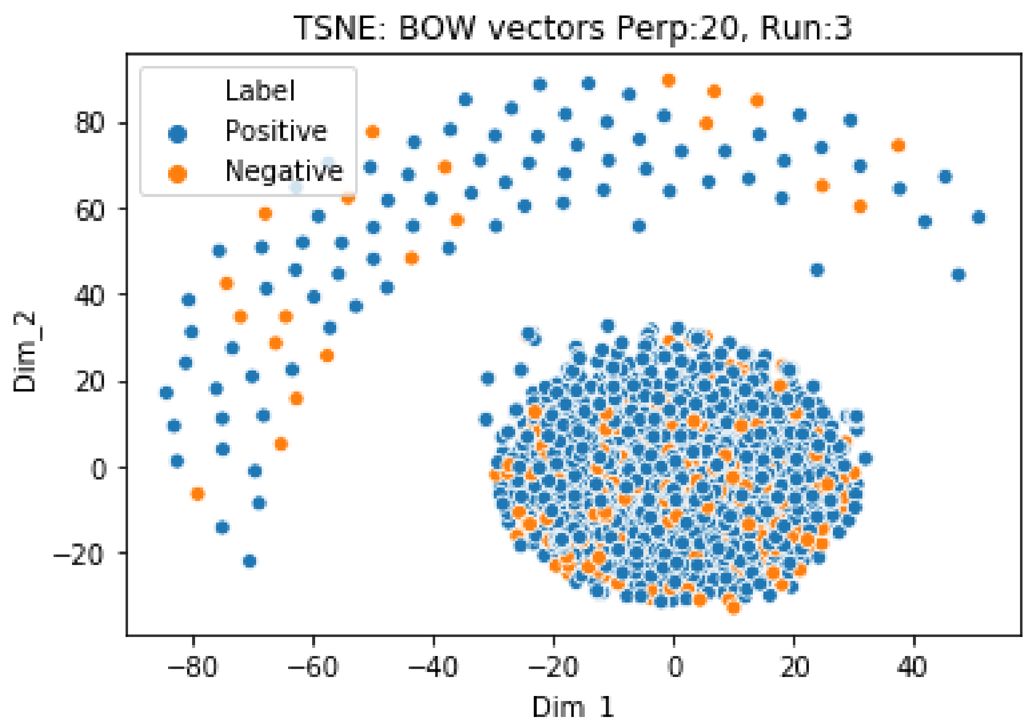

TSNE: BOW vectors Perp:10, Run:1
Label
Positive
Negative
Dim_2
Dim_1
80
60
40
20
0
−20
−40
−60
−40
−20
0
20
40
60
80

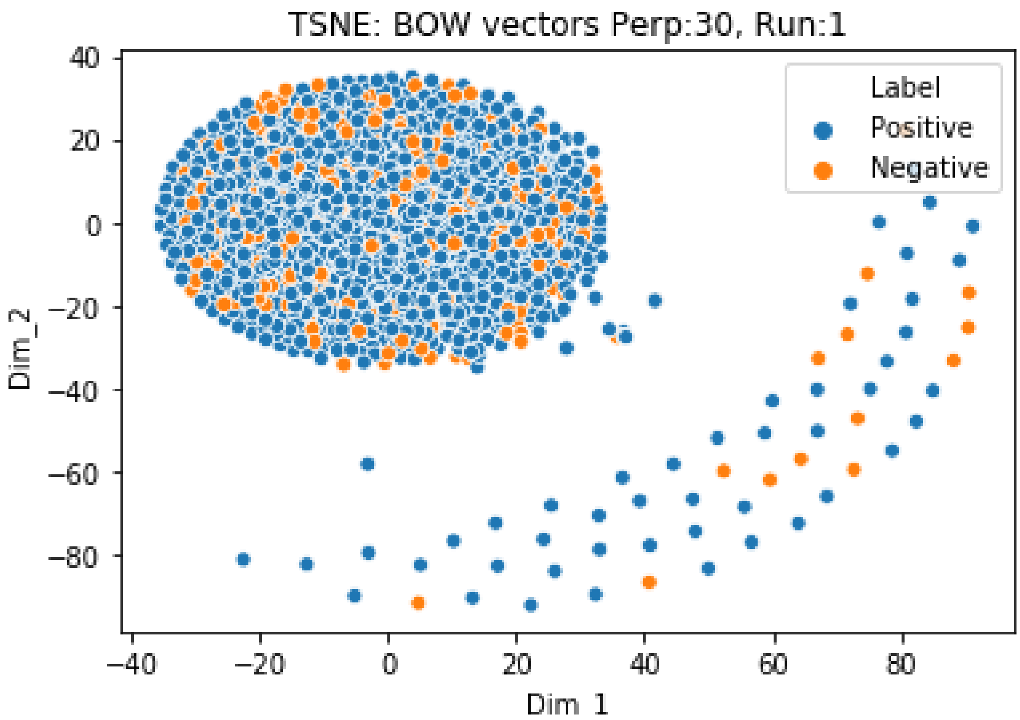

TSNE: BOW vectors Perp:10, Run:2
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−80
−60
−40
−20
0
20
40

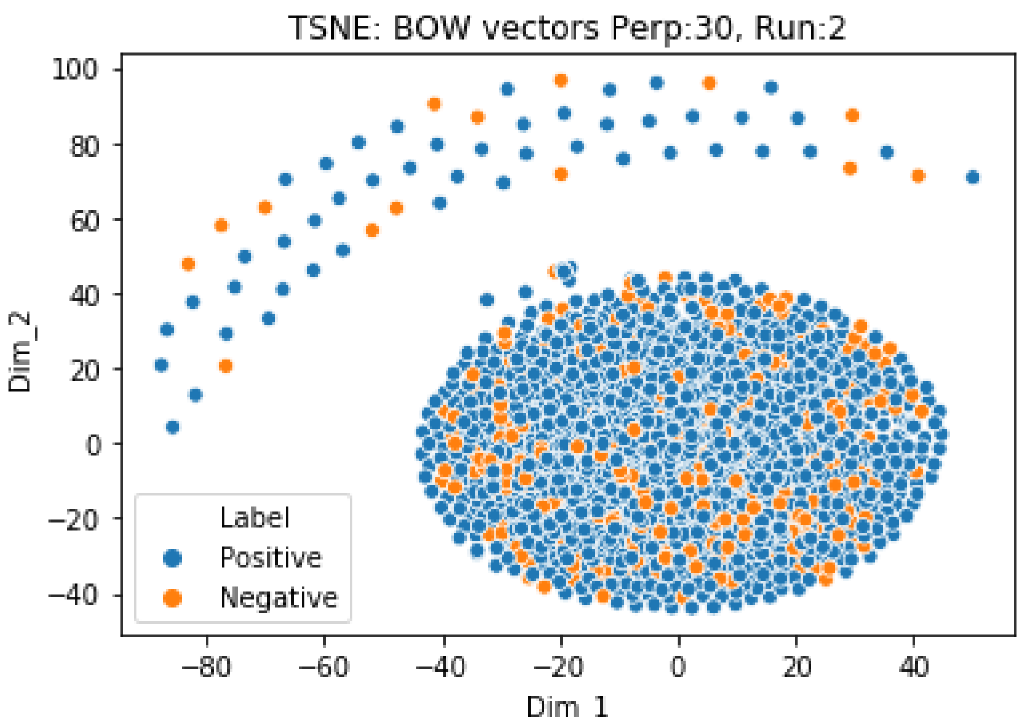

TSNE: BOW vectors Perp:10, Run:3
Dim_2
Dim_1
Label
Positive
Negative

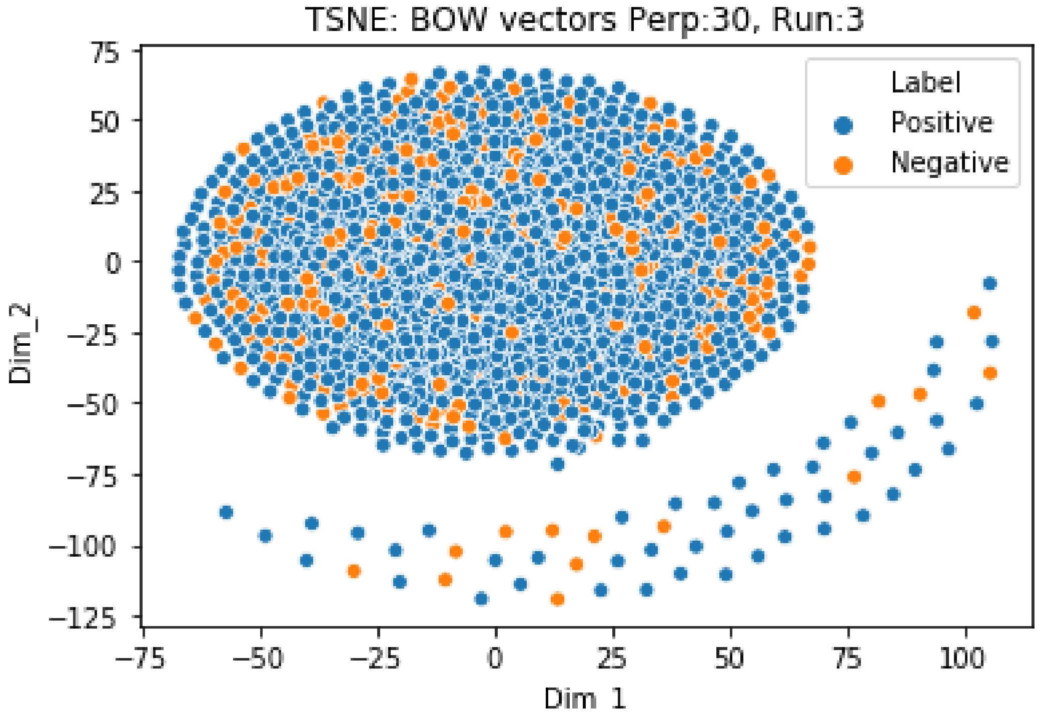

TSNE: BOW vectors Perp:20, Run:1
Dim_2
Dim_1
Label
Positive
Negative

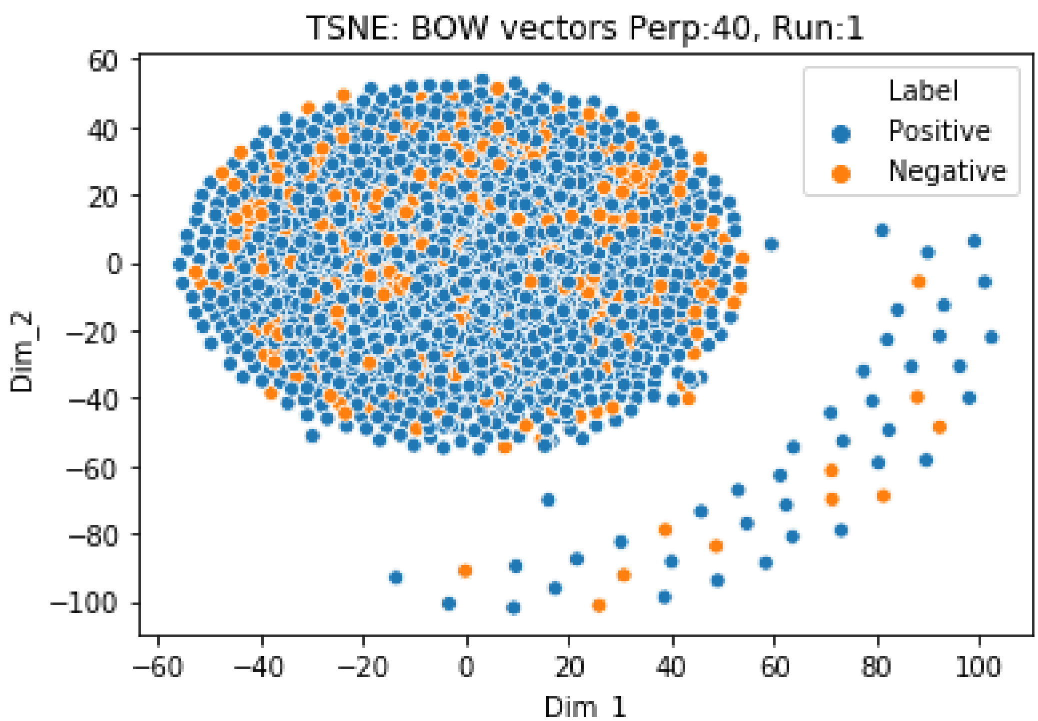

TSNE: BOW vectors Perp:20, Run:2
Dim_2
Dim_1
Label
Positive
Negative
100
75
50
25
0
−25
−50
−100
−75
−50
−25
0
25
50
75

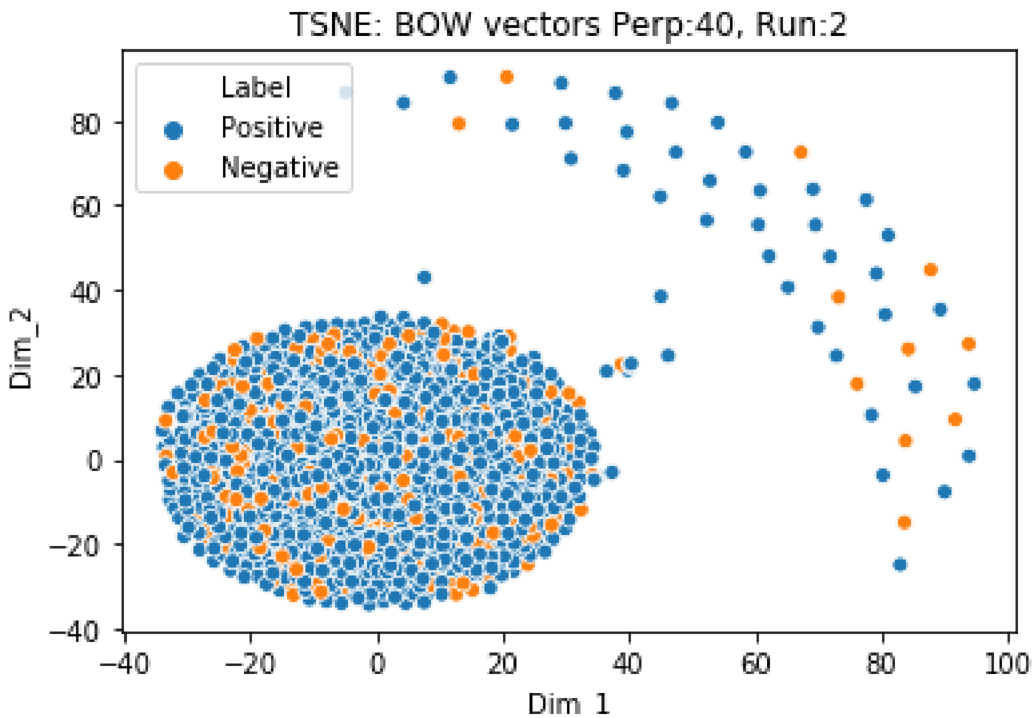

TSNE: BOW vectors Perp:20, Run:3
Dim_2
Dim_1
Label
Positive
Negative
80
60
40
20
0
−20
−80
−60
−40
−20
0
20
40

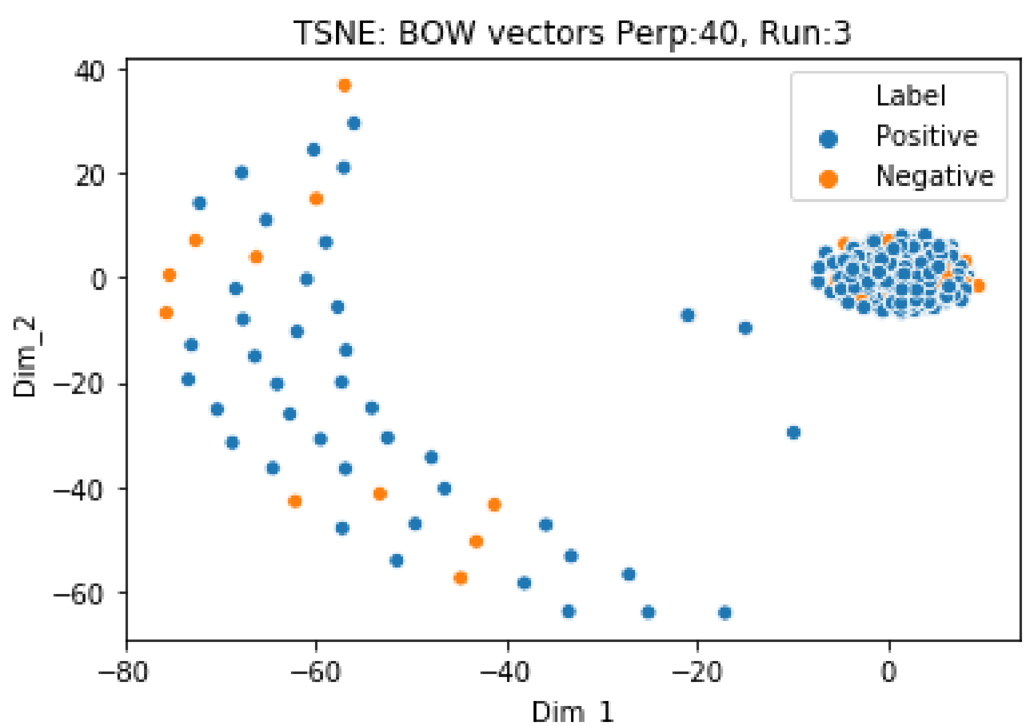

TSNE: BOW vectors Perp:30, Run:1
Label
Positive
Negative
Dim_1
Dim_2


TSNE: BOW vectors Perp:30, Run:2
Label
Positive
Negative
Dim_1
Dim_2


TSNE: BOW vectors Perp:30, Run:3
Label
Positive
Negative
Dim_2
Dim_1
75
50
25
0
−25
−50
−75
−100
−125
−75
−50
−25
0
25
50
75
100


TSNE: BOW vectors Perp:40, Run:1
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−80
−100
−60
−40
−20
0
20
40
60
80
100

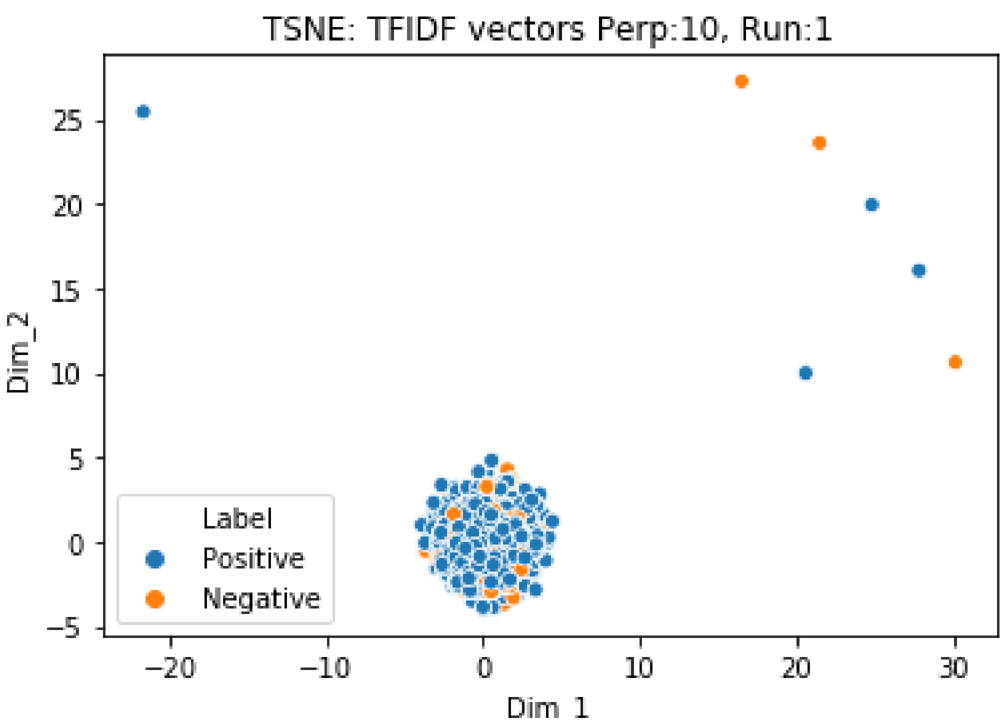

TSNE: BOW vectors Perp:40, Run:2
Label
Positive
Negative
Dim_1
Dim_2

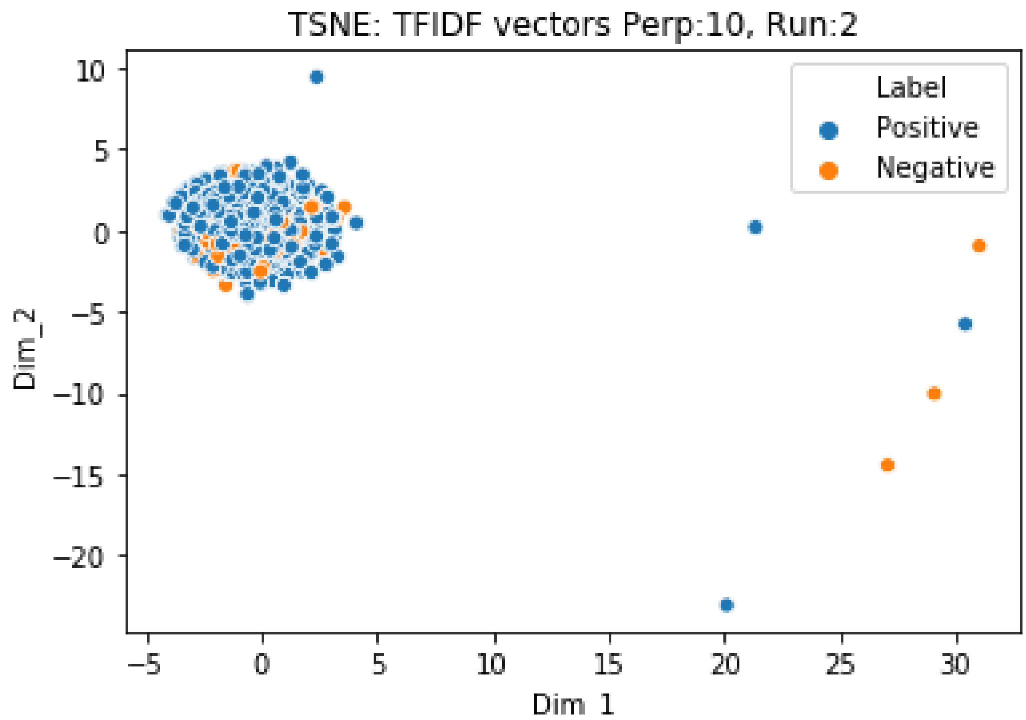

TSNE: BOW vectors Perp:40, Run:3
Label
Positive
Negative
Dim_1
Dim_2

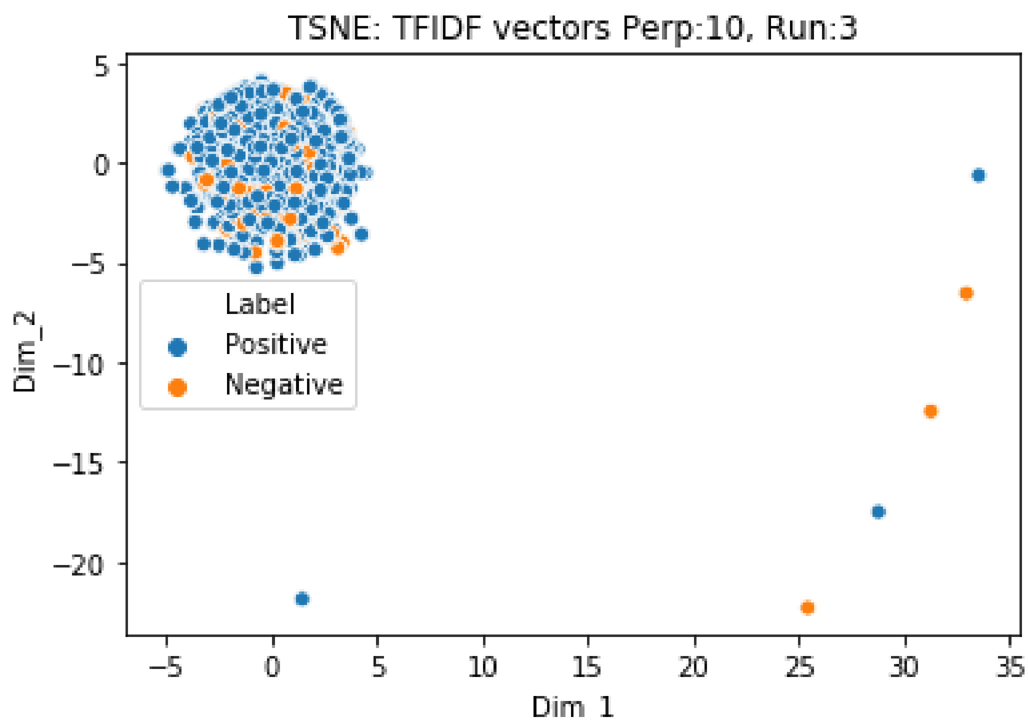
Observations
None of the plots showed any significant separation between poistive class and negative class

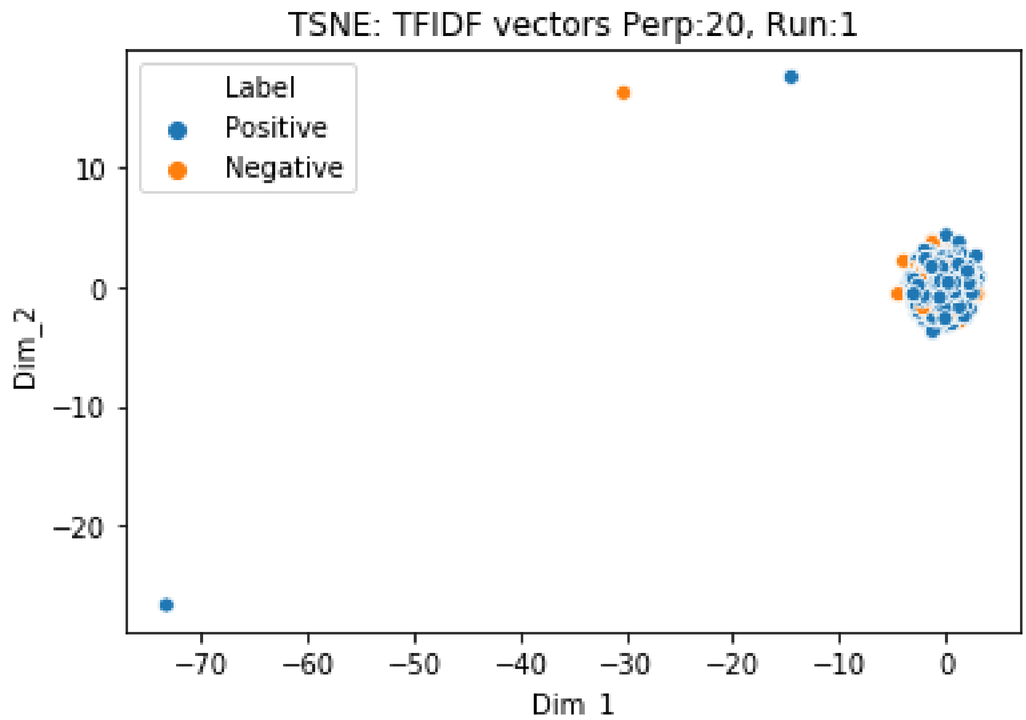
6.3 [5.1] Applying TNSE on Text TFIDF vectors

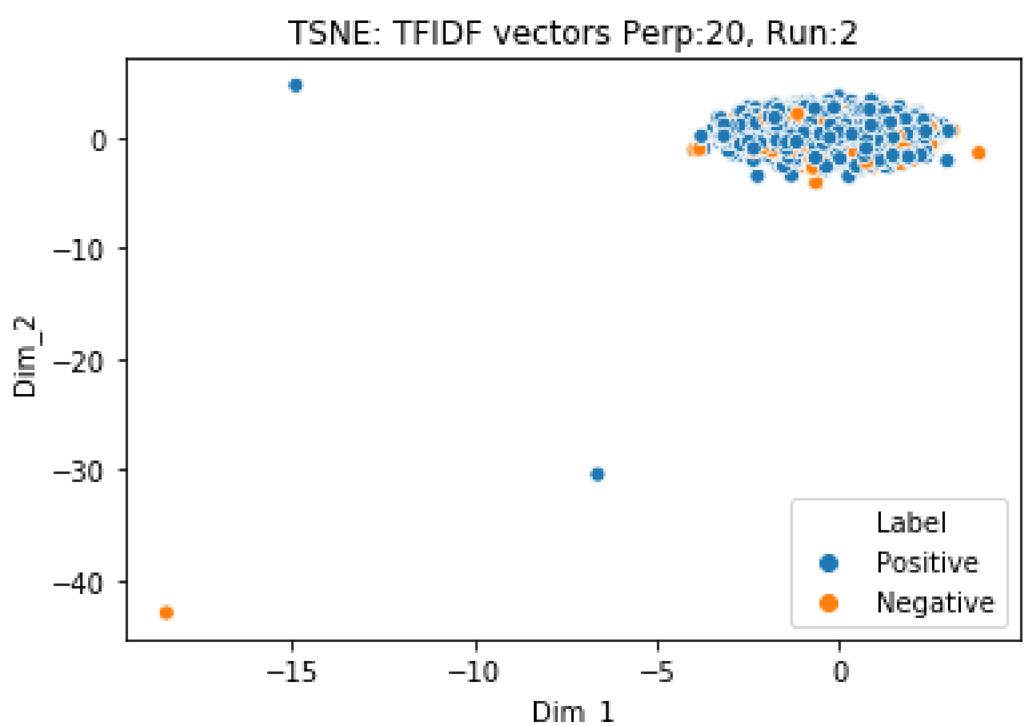
In [39]:
```
max_data_points = 3000
features_t = final_tf_idf[0:max_data_points]
features_t = features_t.toarray()
# standardize the data
features_t = scaler_object.fit_transform(features_t)
# get lables
lables_t = final['Score'].tolist()[0:max_data_points]
lables_t = [label_dict[item] for item in lables_t]
# plot it
get_tsne_plot(features_t, lables_t, 'TFIDF vectors')
```

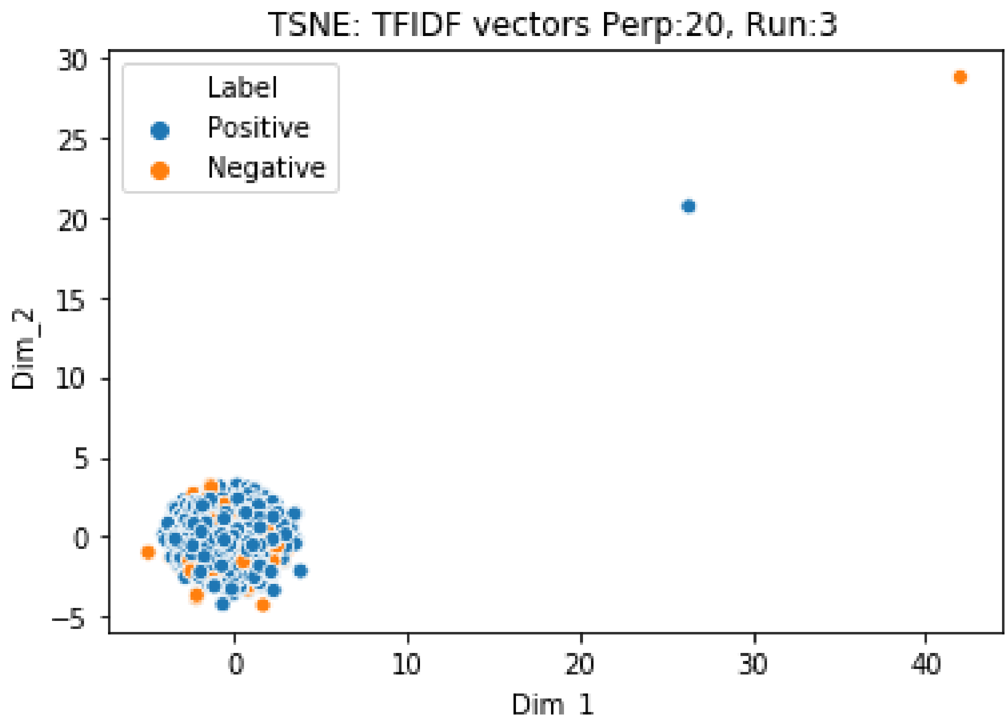



TSNE: TFIDF vectors Perp:10, Run:2
Label
Positive
Negative
Dim_2
Dim_1
10
5
0
−5
−10
−15
−20
−5
0
5
10
15
20
25
30

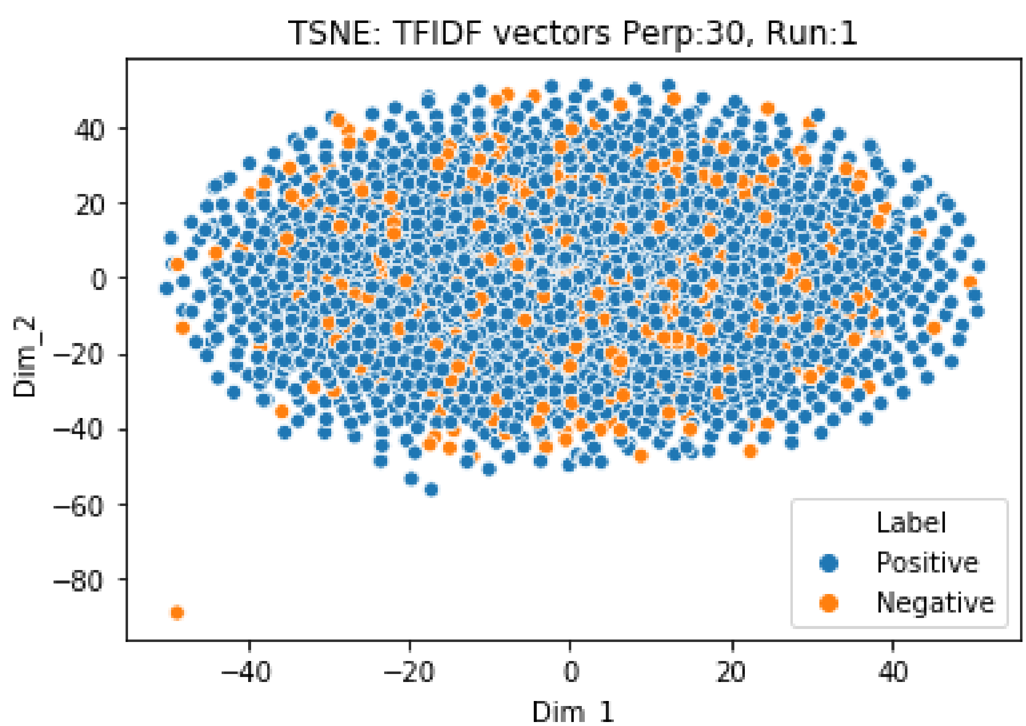

TSNE: TFIDF vectors Perp:10, Run:3
Label
Positive
Negative
Dim_2
Dim_1
5
0
−5
−10
−15
−20
−5
0
5
10
15
20
25
30
35


TSNE: TFIDF vectors Perp:20, Run:1
Label
Positive
Negative
10
0
−10
−20
Dim_2
−70
−60
−50
−40
−30
−20
−10
0
Dim_1

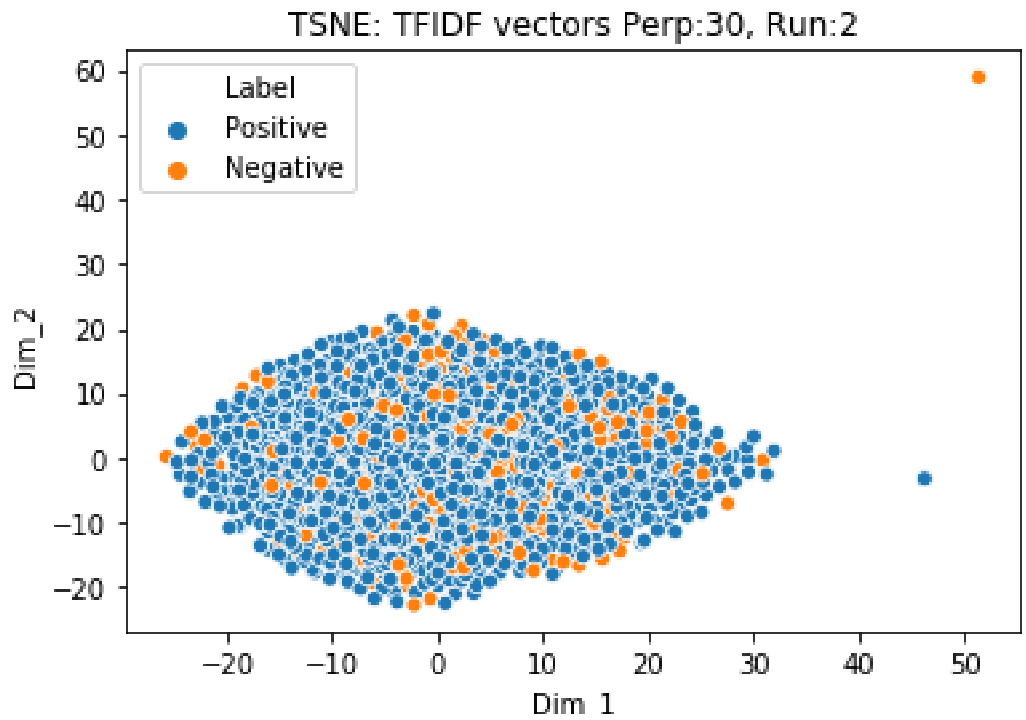

TSNE: TFIDF vectors Perp:20, Run:2
0
−10
−20
−30
−40
Dim_2
Label
Positive
Negative
−15
−10
−5
0
Dim_1

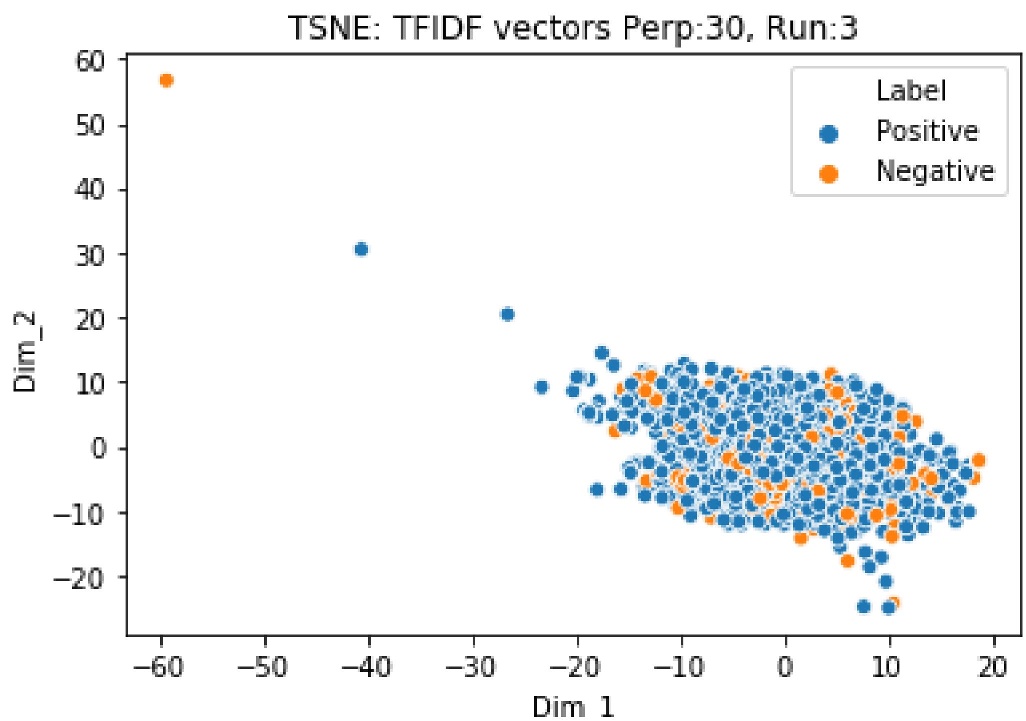


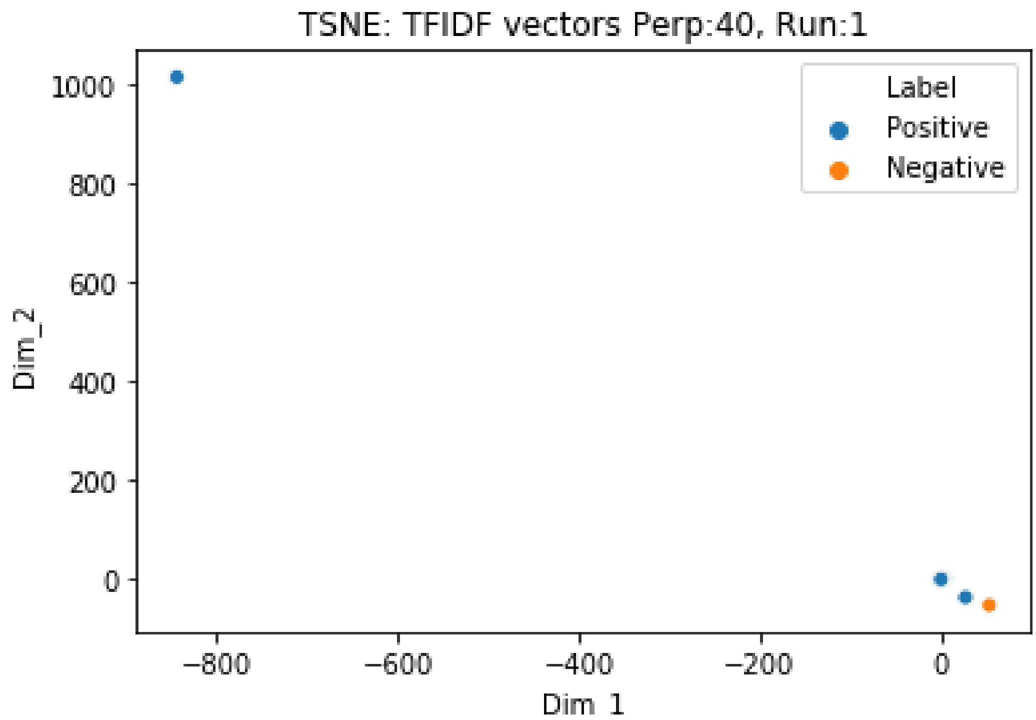
TSNE: TFIDF vectors Perp:20, Run:3
Label
Positive
Negative
Dim_2
Dim_1
30
25
20
15
10
5
0
−5
0
10
20
30
40

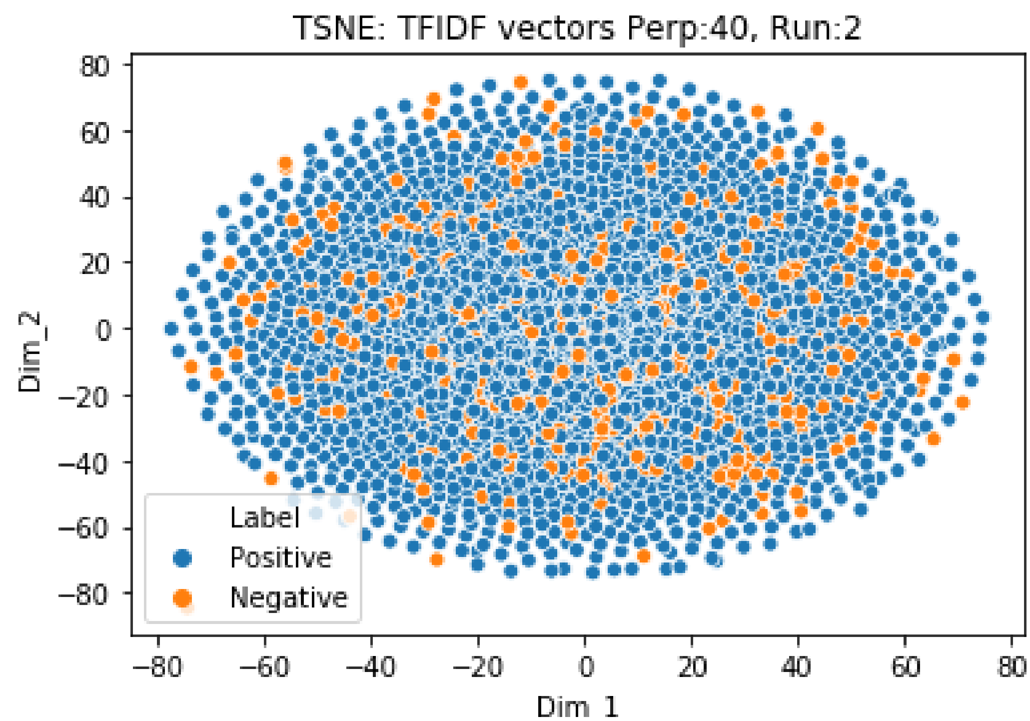


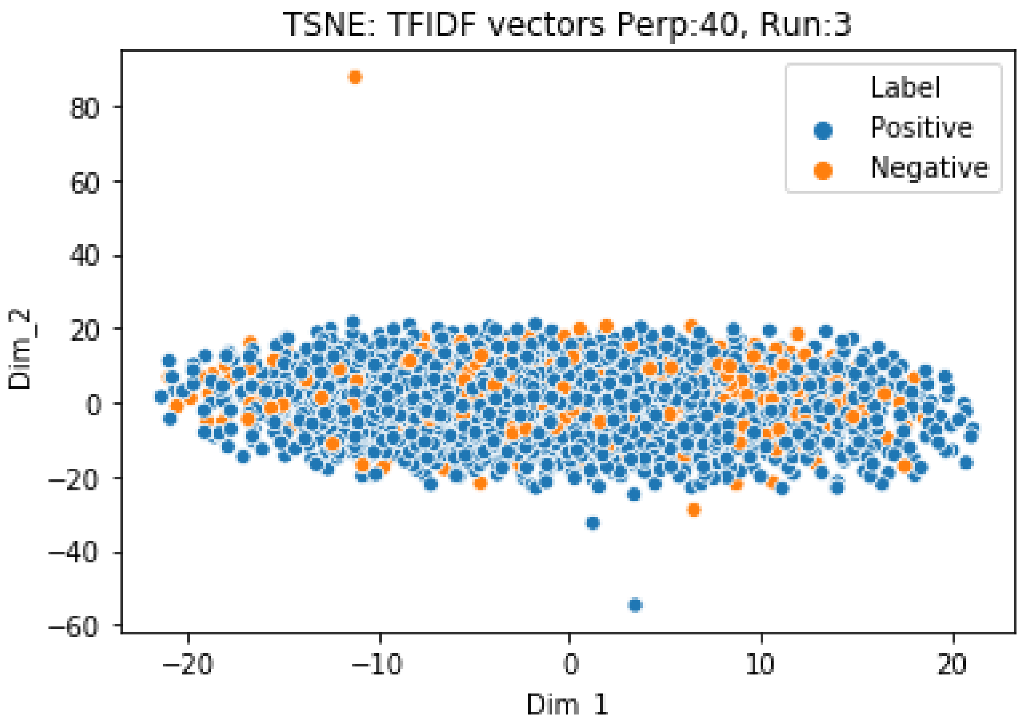
TSNE: TFIDF vectors Perp:30, Run:1
Label
Positive
Negative
Dim_2
Dim_1
40
20
0
−20
−40
−60
−80
−40
−20
0
20
40


TSNE: TFIDF vectors Perp:30, Run:2
Label
Positive
Negative
Dim_2
Dim_1
60
50
40
30
20
10
0
−10
−20
−20
−10
0
10
20
30
40
50


TSNE: TFIDF vectors Perp:30, Run:3
Label
Positive
Negative
Dim_2
Dim_1
60
50
40
30
20
10
0
−10
−20
−60
−50
−40
−30
−20
−10
0
10
20


TSNE: TFIDF vectors Perp:40, Run:1
Label
Positive
Negative
Dim_2
Dim_1
1000
800
600
400
200
0
−800
−600
−400
−200
0

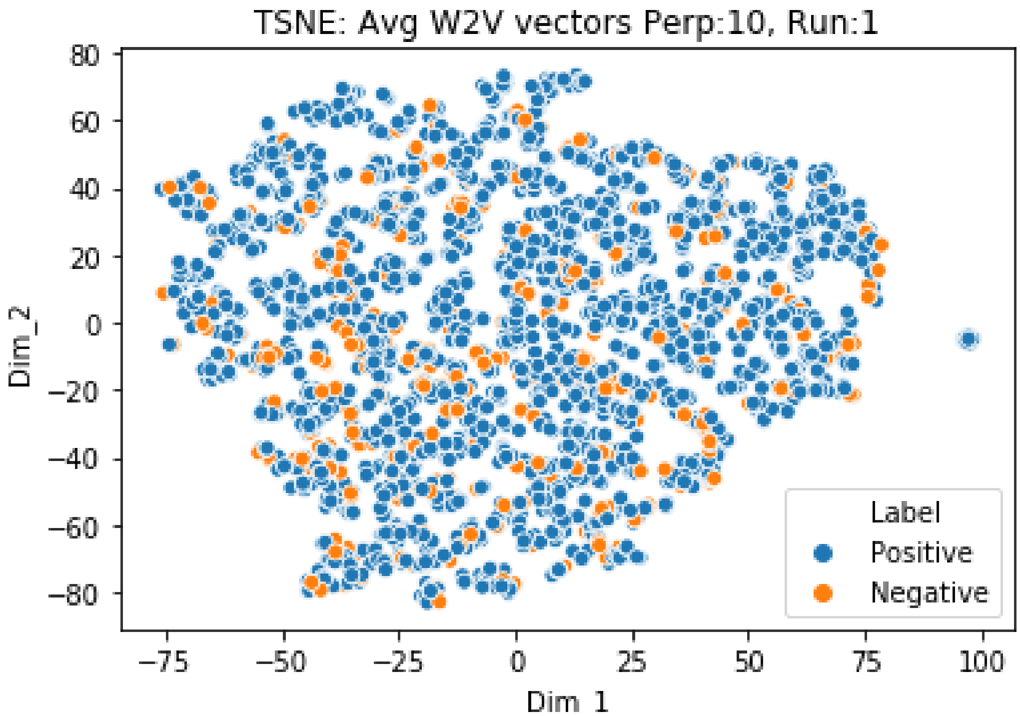

TSNE: TFIDF vectors Perp:40, Run:2
Label
Positive
Negative
Dim_2
Dim_1
80
60
40
20
0
−20
−40
−60
−80
−80
−60
−40
−20
0
20
40
60
80

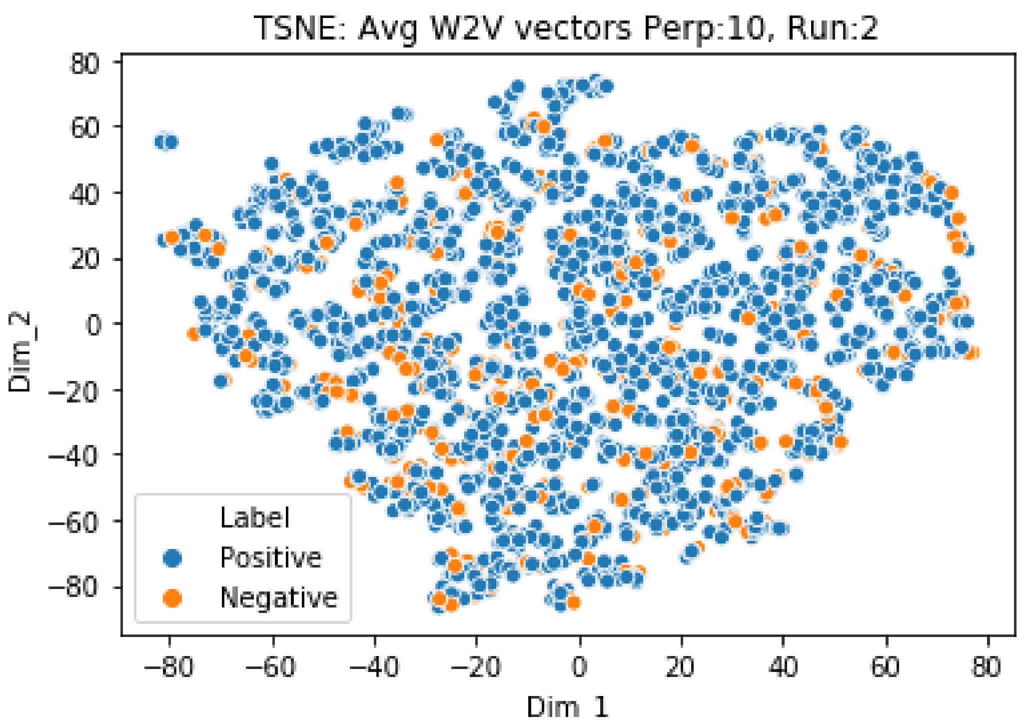


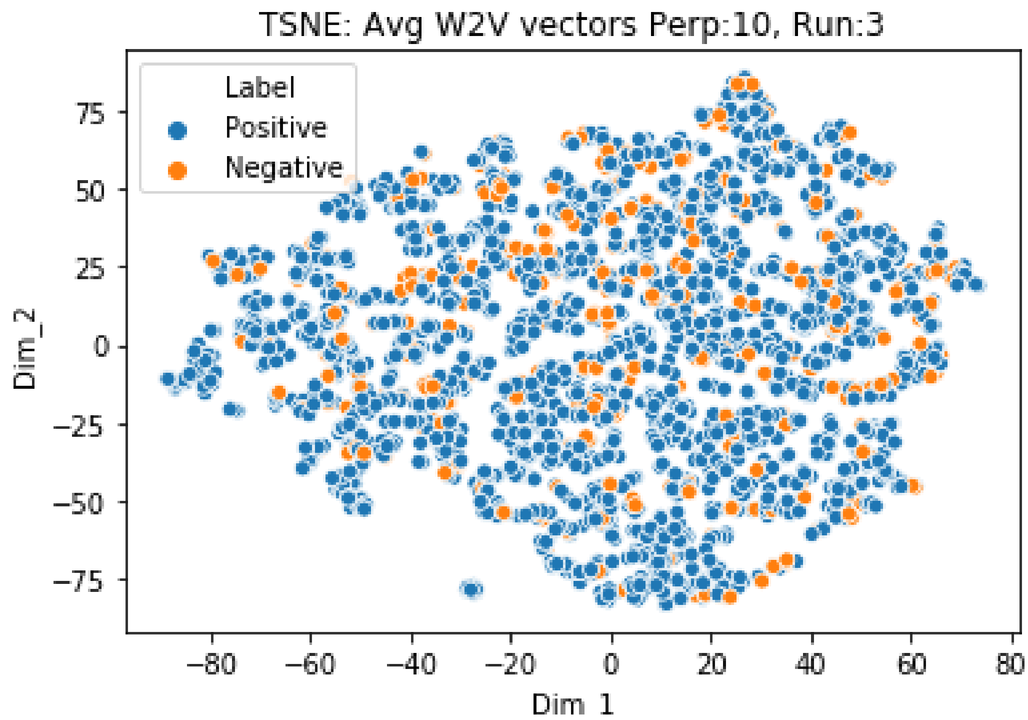
Observations
None of the plots showed any significant separation between poistive class and negative class

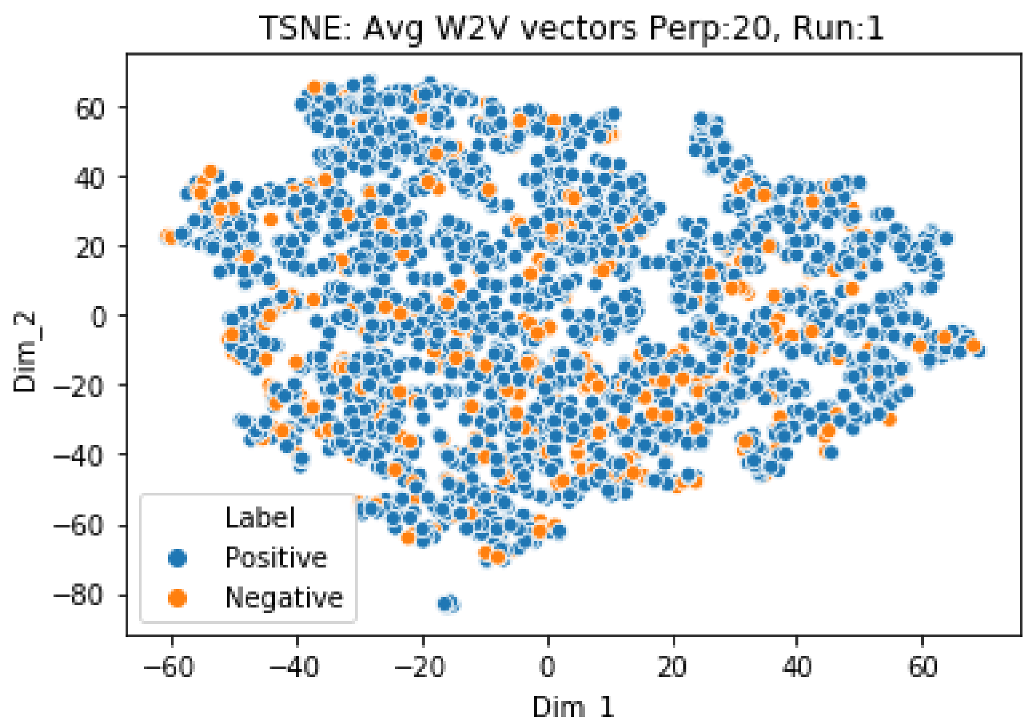
6.4 [5.3] Applying TNSE on Text Avg W2V vectors

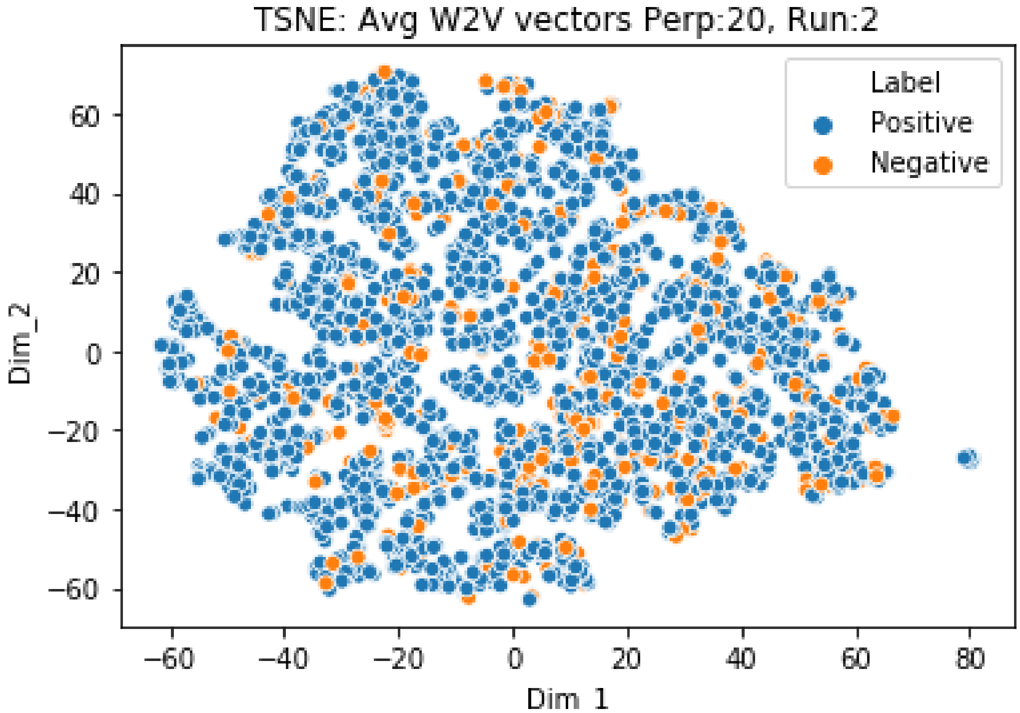
```
In [40]: max_data_points = 3000
         features_t = sent_vectors[0:max_data_points]
         # standardize the data
         features_t = scaler_object.fit_transform(features_t)
         # get lables
         lables_t = final['Score'].tolist()[0:max_data_points]
         lables_t = [label_dict[item] for item in lables_t]
         # plot it
         get_tsne_plot(features_t, lables_t, 'Avg W2V vectors')
```

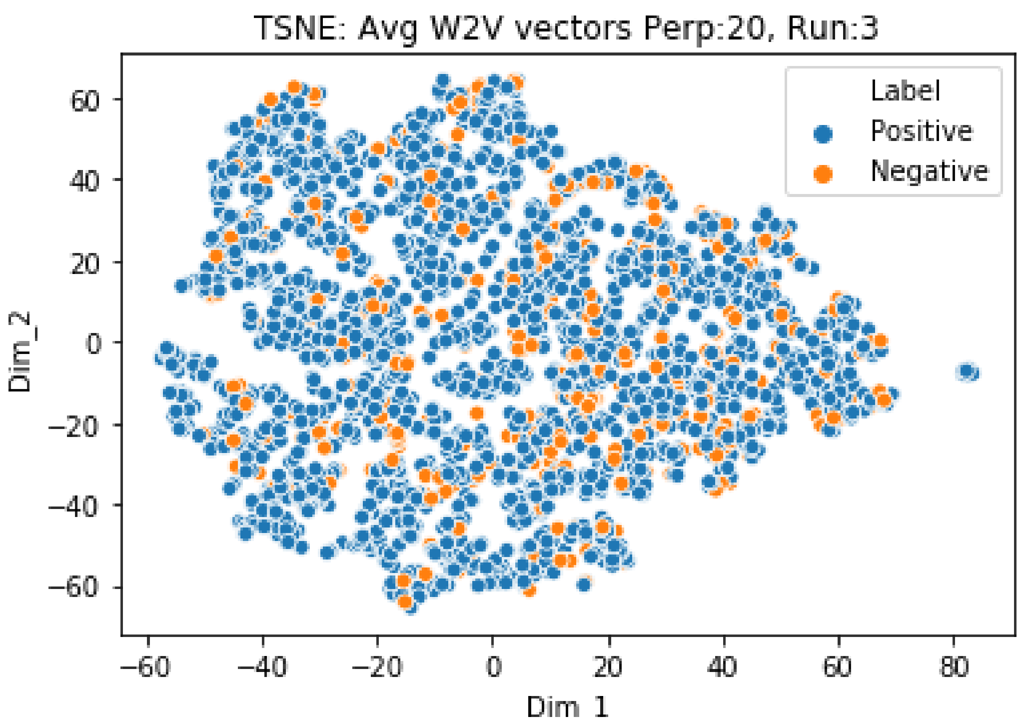

TSNE: Avg W2V vectors Perp:10, Run:1
Dim_2
Dim_1
Label
Positive
Negative
80
60
40
20
0
−20
−40
−60
−80
−75
−50
−25
0
25
50
75
100

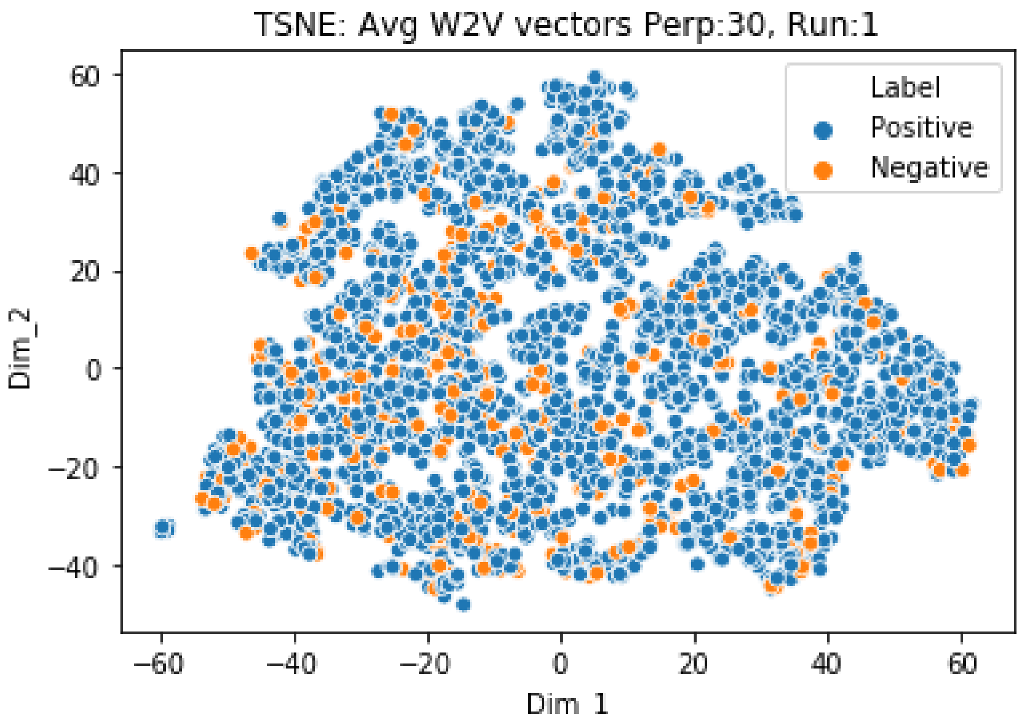

TSNE: Avg W2V vectors Perp:10, Run:2
Dim_2
Dim_1
Label
Positive
Negative
80
60
40
20
0
−20
−40
−60
−80
−80
−60
−40
−20
0
20
40
60
80

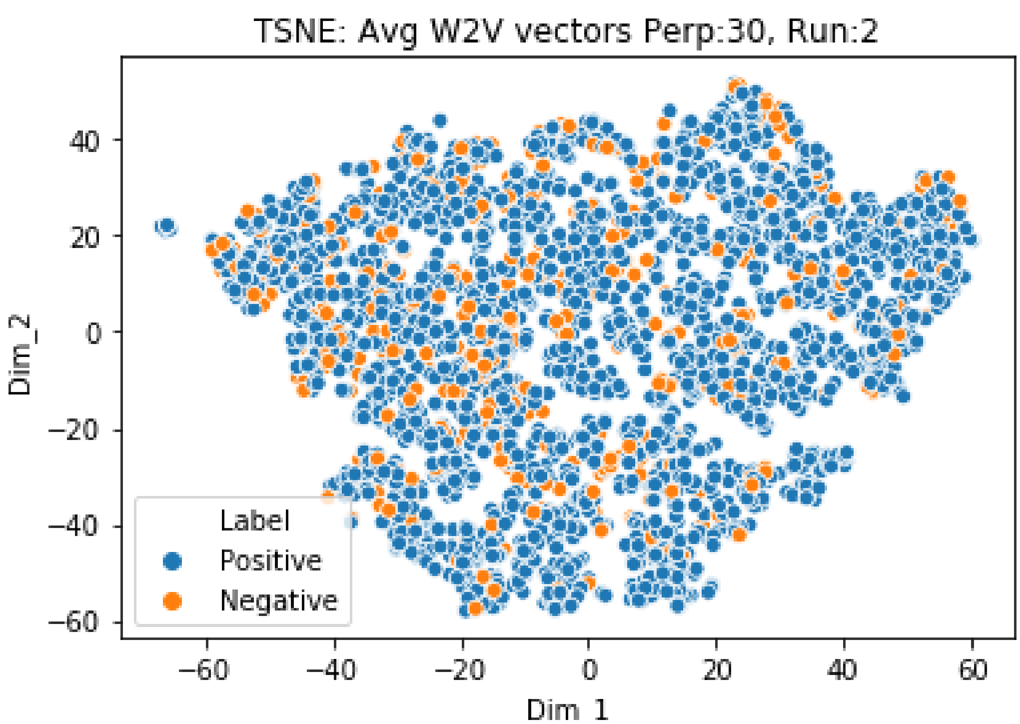

TSNE: Avg W2V vectors Perp:10, Run:3
Label
Positive
Negative
Dim_2
Dim_1

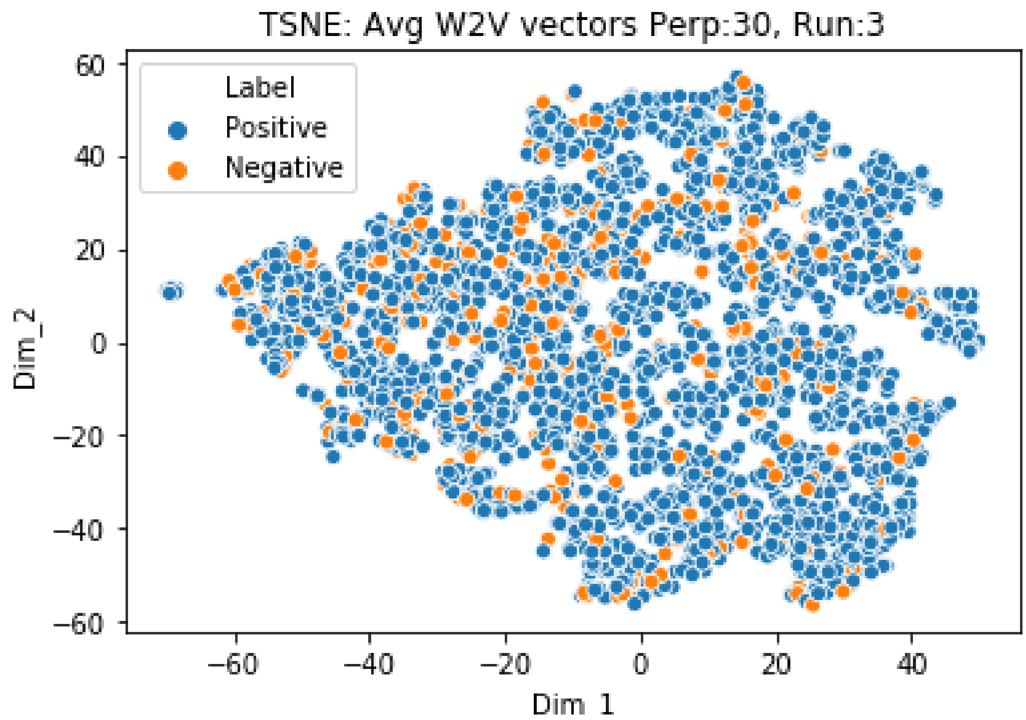

TSNE: Avg W2V vectors Perp:20, Run:1
Label
Positive
Negative
Dim_2
Dim_1

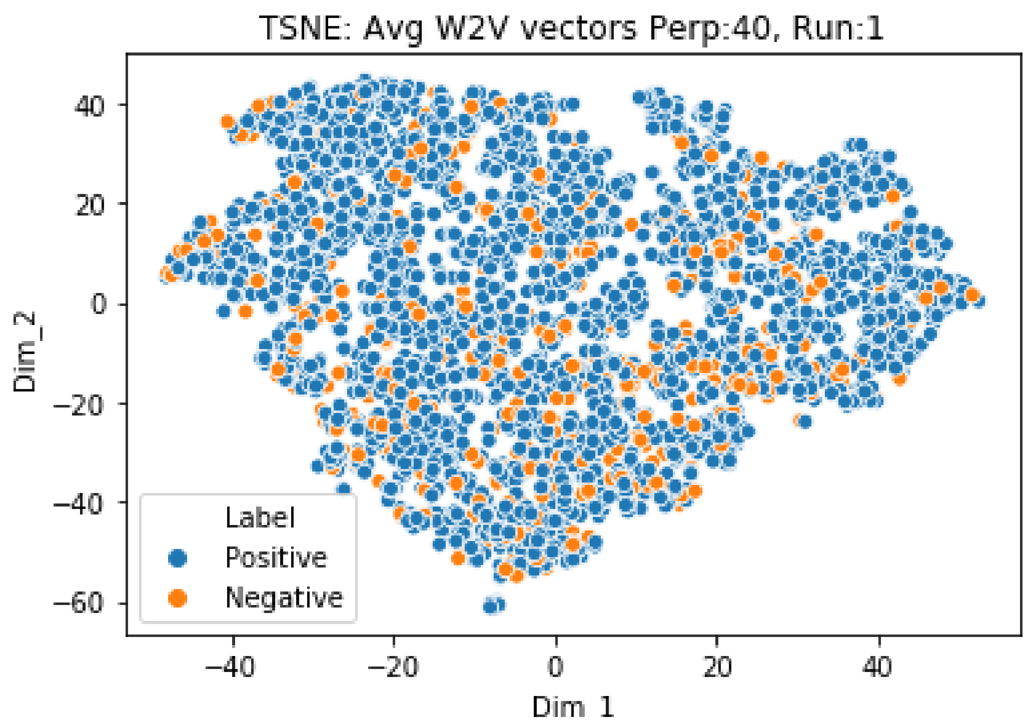

TSNE: Avg W2V vectors Perp:20, Run:2
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
80

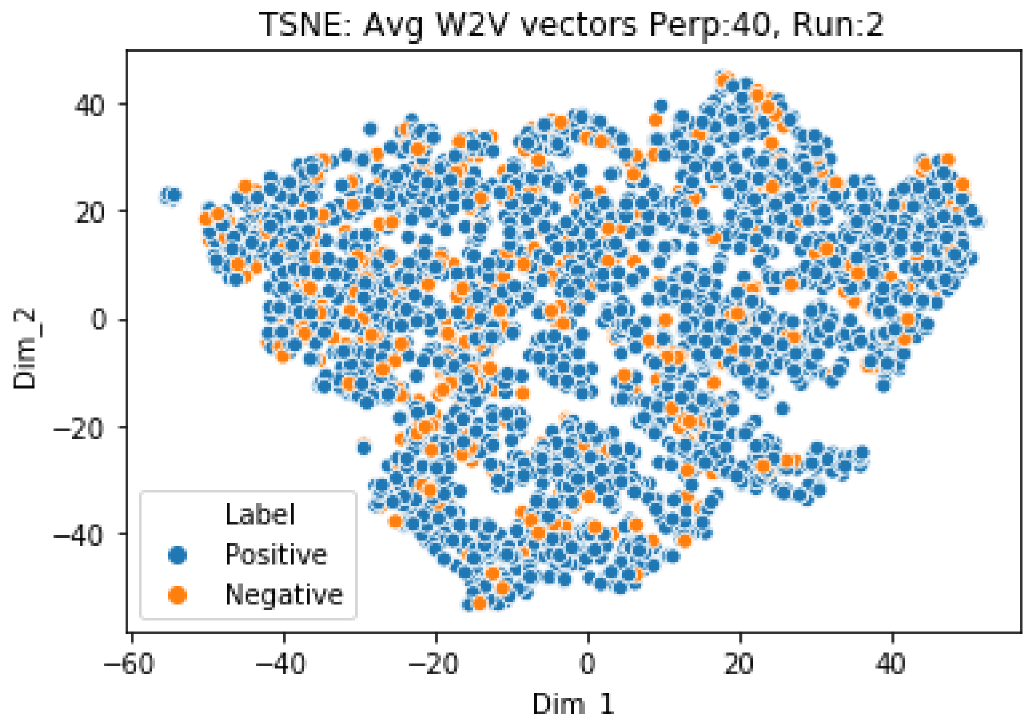

TSNE: Avg W2V vectors Perp:20, Run:3
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
80

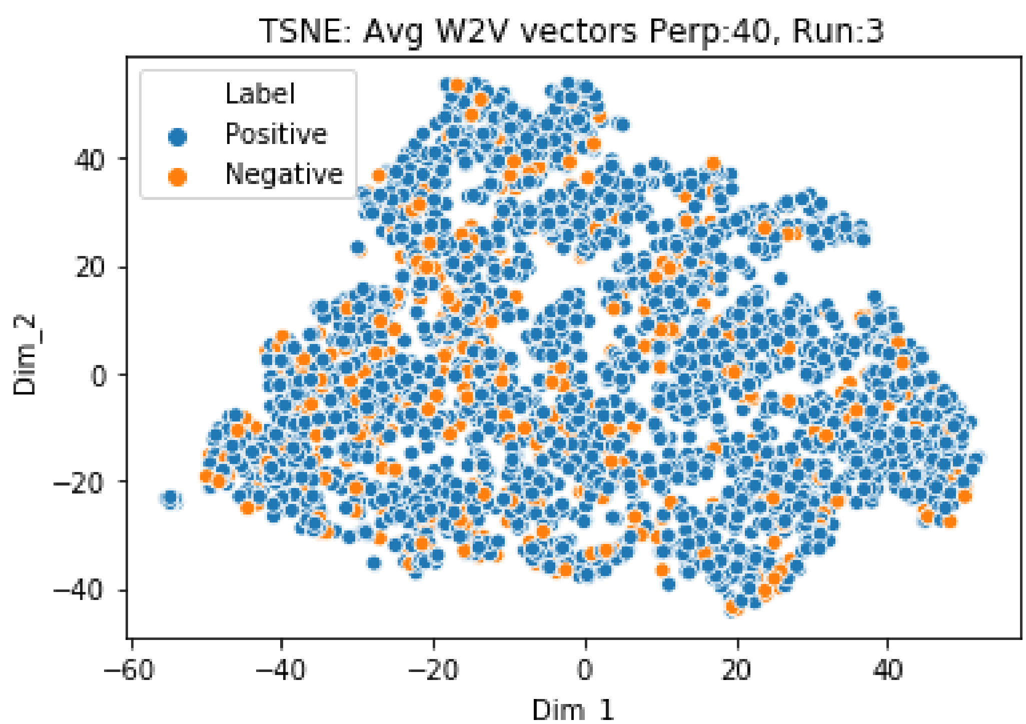

TSNE: Avg W2V vectors Perp:30, Run:1
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−40
−20
0
20
40
60


TSNE: Avg W2V vectors Perp:30, Run:2
Label
Positive
Negative
Dim_2
Dim_1
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60


TSNE: Avg W2V vectors Perp:30, Run:3
Label
Positive
Negative
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40


TSNE: Avg W2V vectors Perp:40, Run:1
Label
Positive
Negative
Dim_2
Dim_1
40
20
0
−20
−40
−60
−40
−20
0
20
40

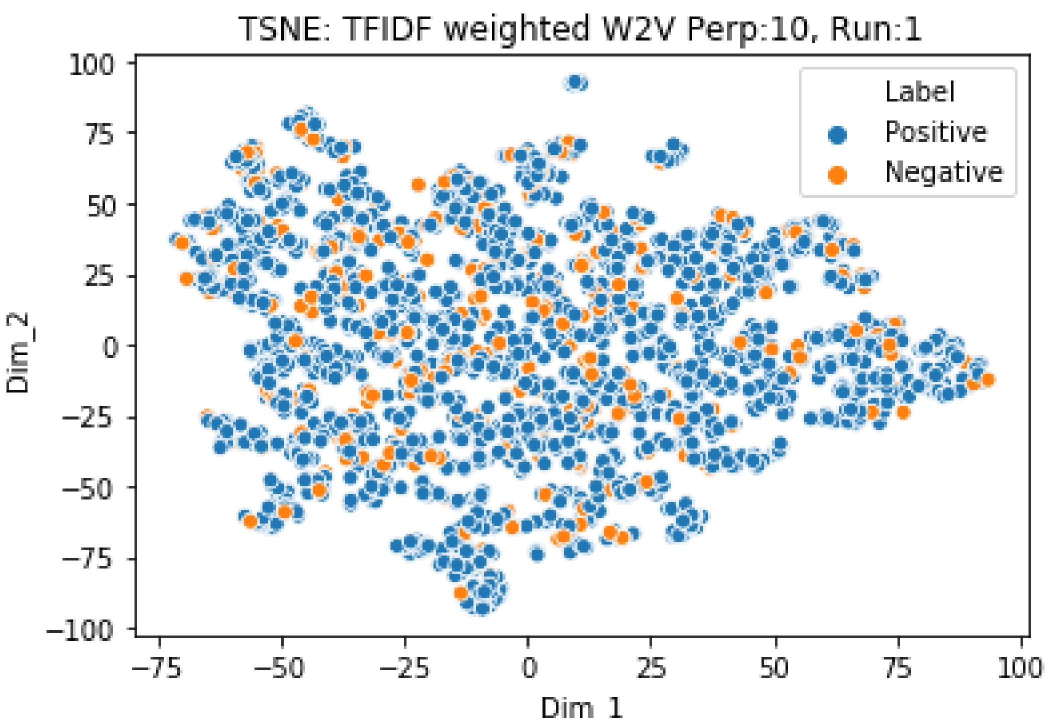

TSNE: Avg W2V vectors Perp:40, Run:2
Label
Positive
Negative
Dim_2
Dim_1
40
20
0
−20
−40
−60
−40
−20
0
20
40

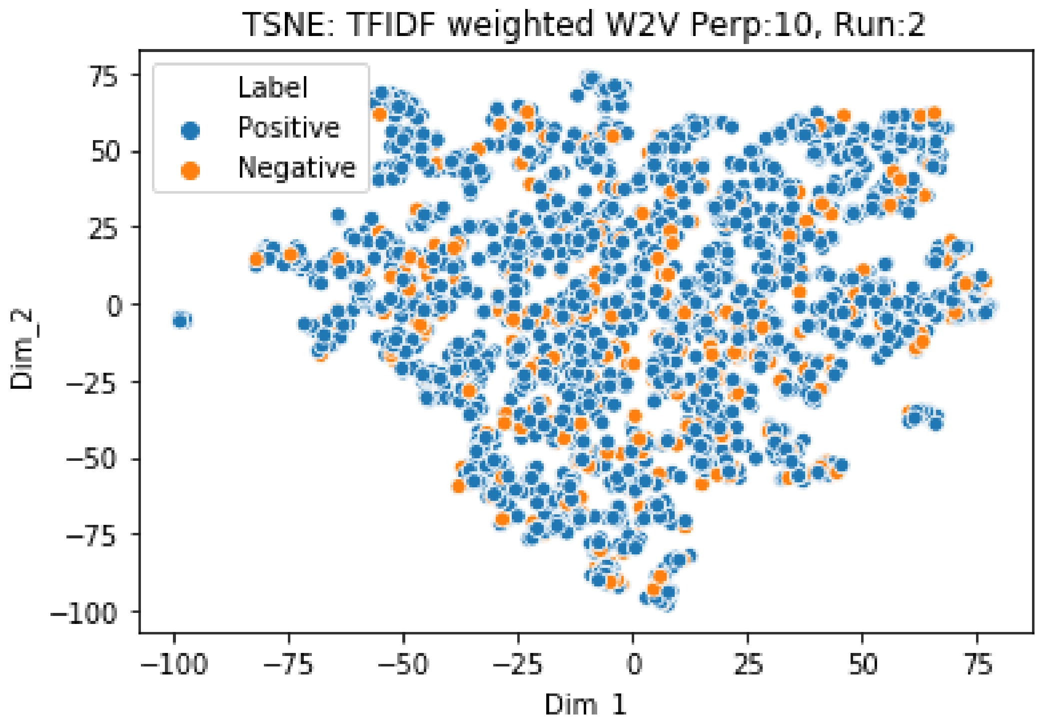

TSNE: Avg W2V vectors Perp:40, Run:3
Label
Positive
Negative
Dim_2
Dim_1
40
20
0
−20
−40
−60
−40
−20
0
20
40

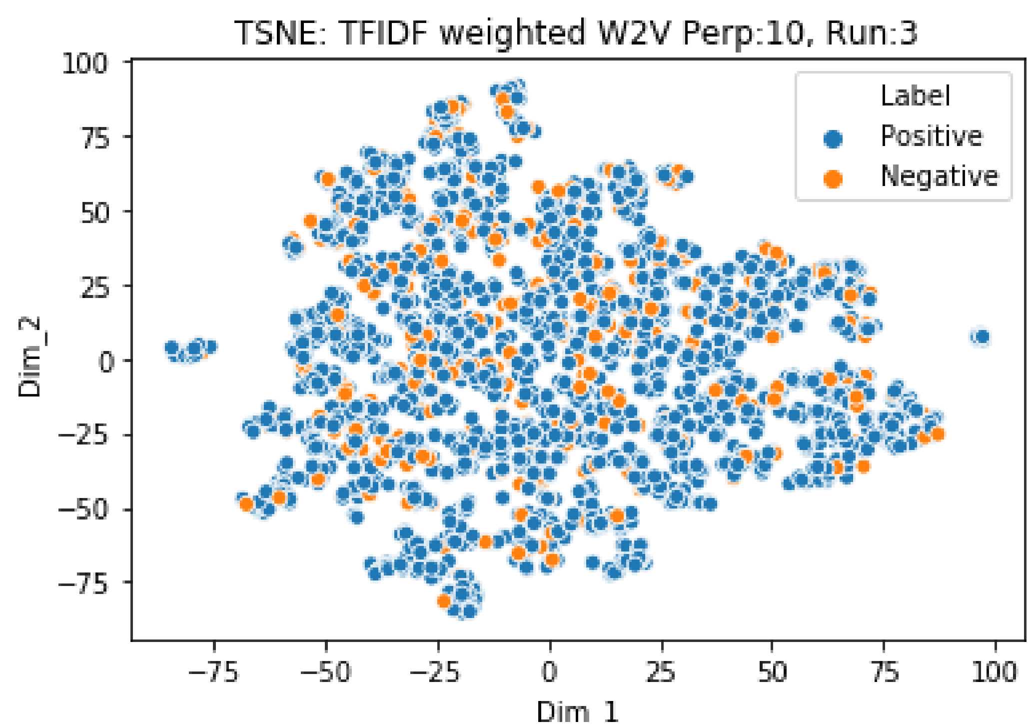
Observations
None of the plots showed any significant separation between poistive class and negative class

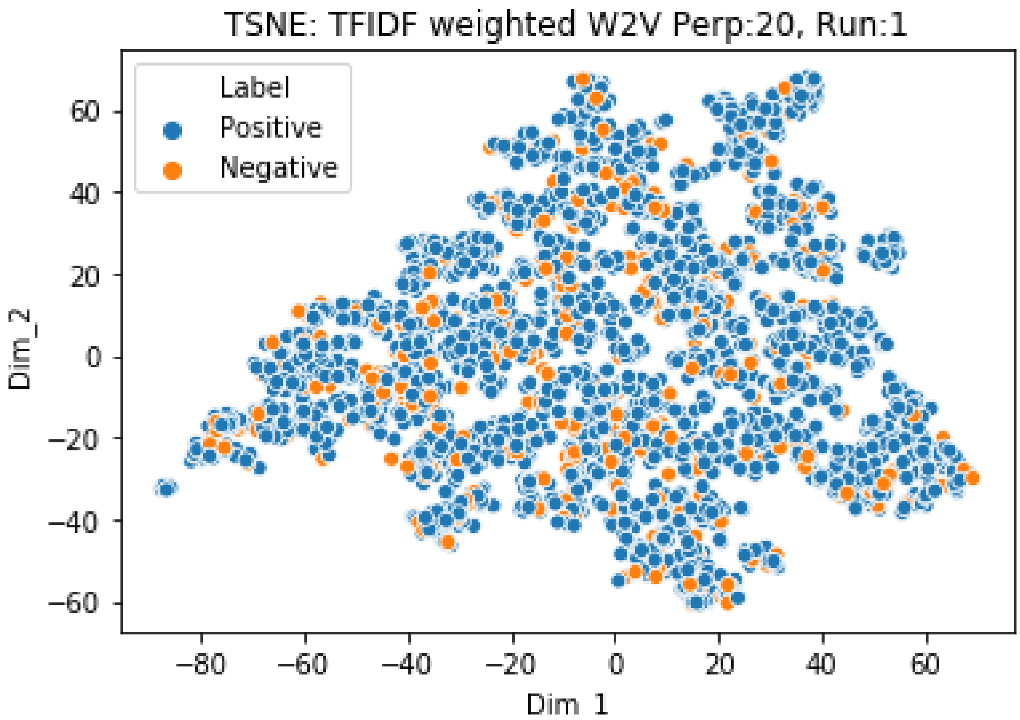
6.5 [5.4] Applying TNSE on Text TFIDF weighted W2V vectors

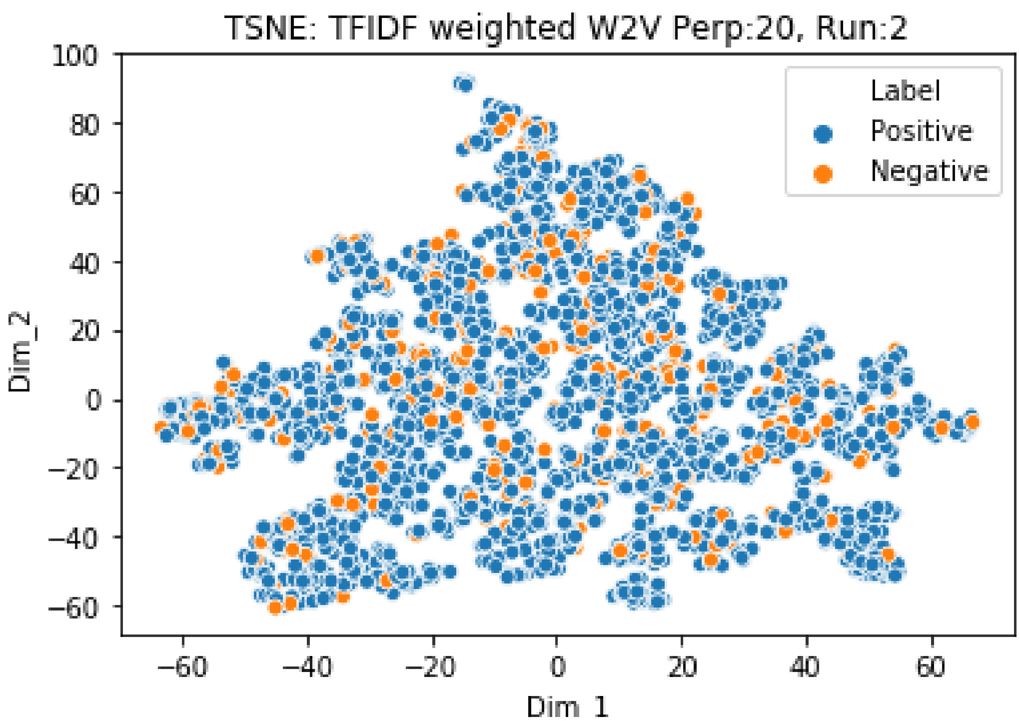
In [41]:
```
max_data_points = 3000
features_t = tfidf_sent_vectors[0:max_data_points]
# standardize the data
features_t = scaler_object.fit_transform(features_t)
# get lables
lables_t = final['Score'].tolist()[0:max_data_points]
lables_t = [label_dict[item] for item in lables_t]
# plot it
get_tsne_plot(features_t, lables_t, 'TFIDF weighted W2V')
```

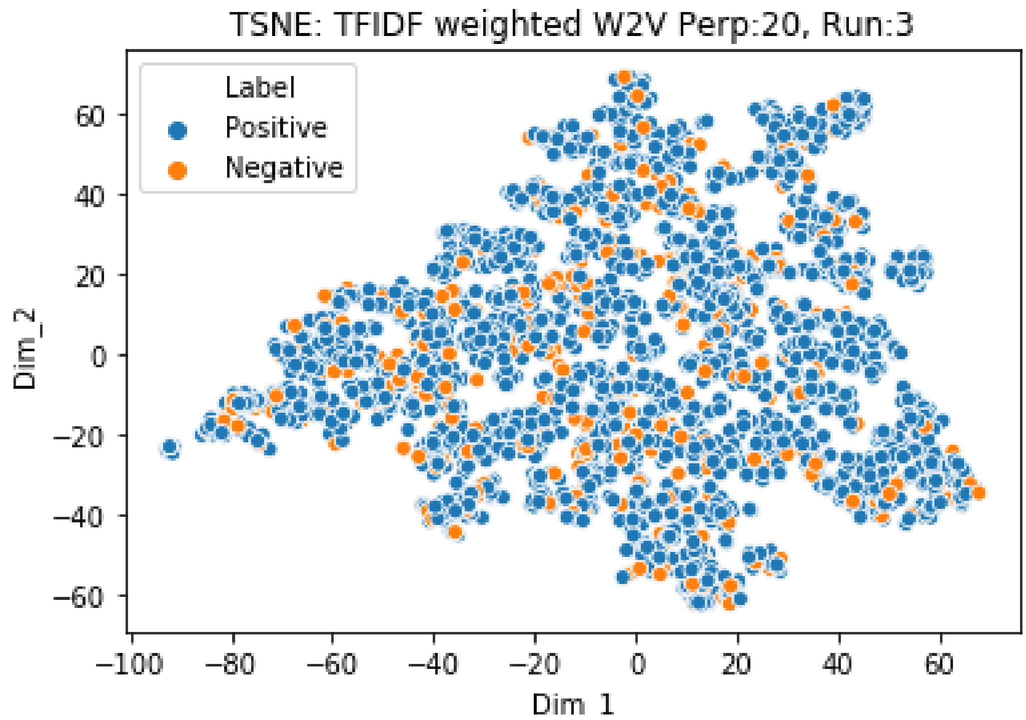



TSNE: TFIDF weighted W2V Perp:10, Run:2
Label
Positive
Negative
Dim_2
Dim_1
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75

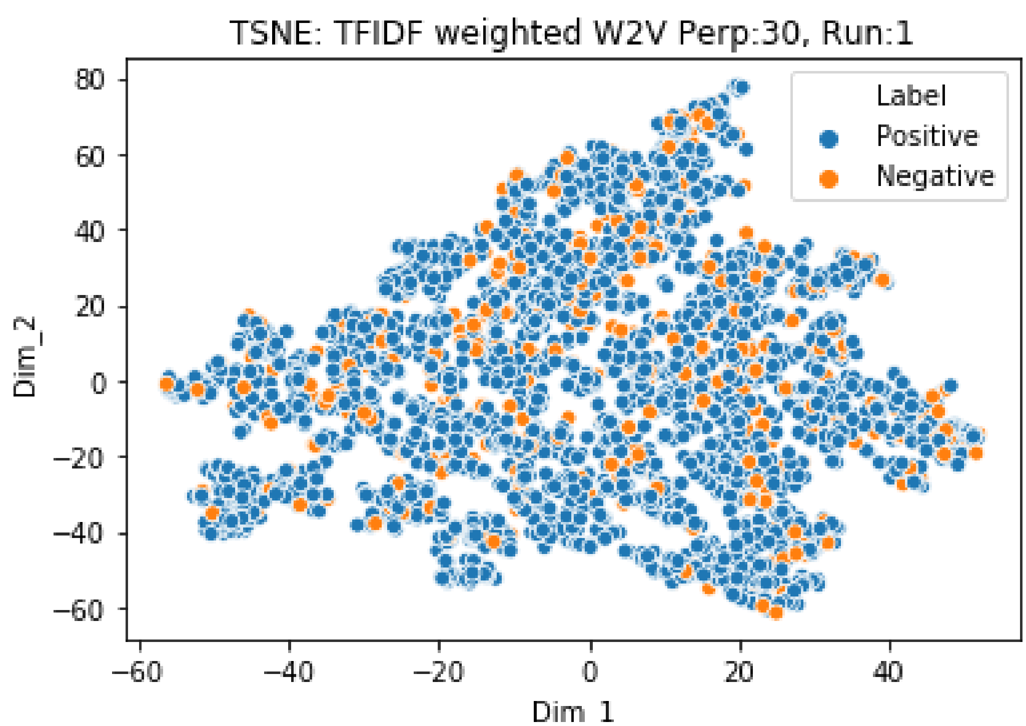

TSNE: TFIDF weighted W2V Perp:10, Run:3
Label
Positive
Negative
Dim_2
Dim_1
100
75
50
25
0
−25
−50
−75
−75
−50
−25
0
25
50
75
100


TSNE: TFIDF weighted W2V Perp:20, Run:1
Label
Positive
Negative
60
40
20
0
−20
−40
−60
Dim_2
−80
−60
−40
−20
0
20
40
60
Dim_1

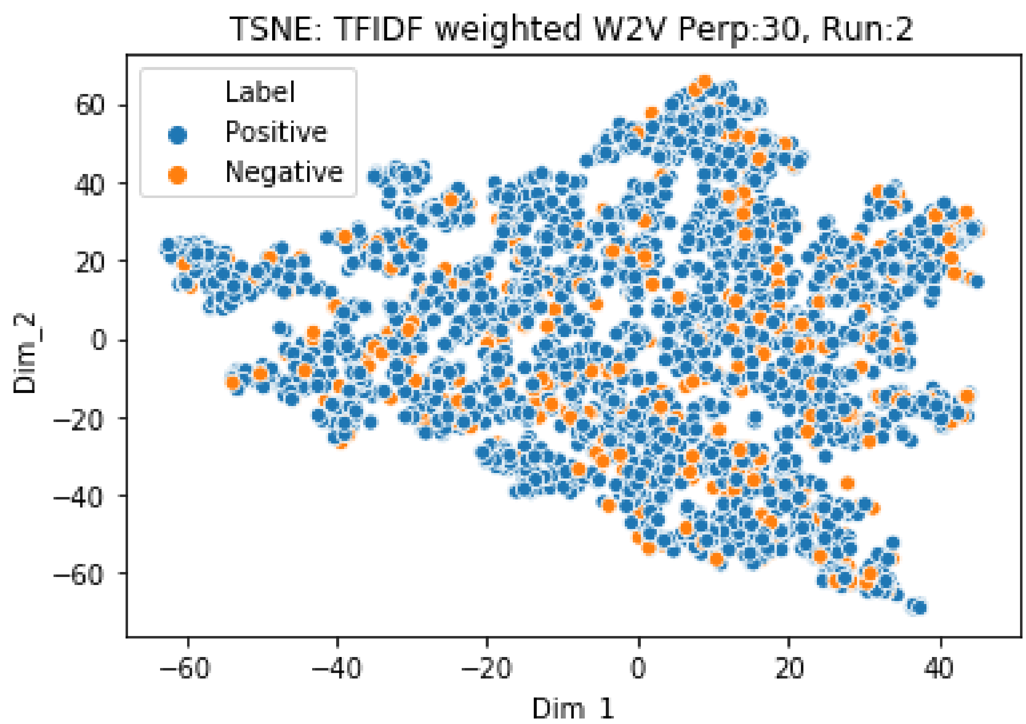

TSNE: TFIDF weighted W2V Perp:20, Run:2
Label
Positive
Negative
100
80
60
40
20
0
−20
−40
−60
Dim_2
−60
−40
−20
0
20
40
60
Dim_1

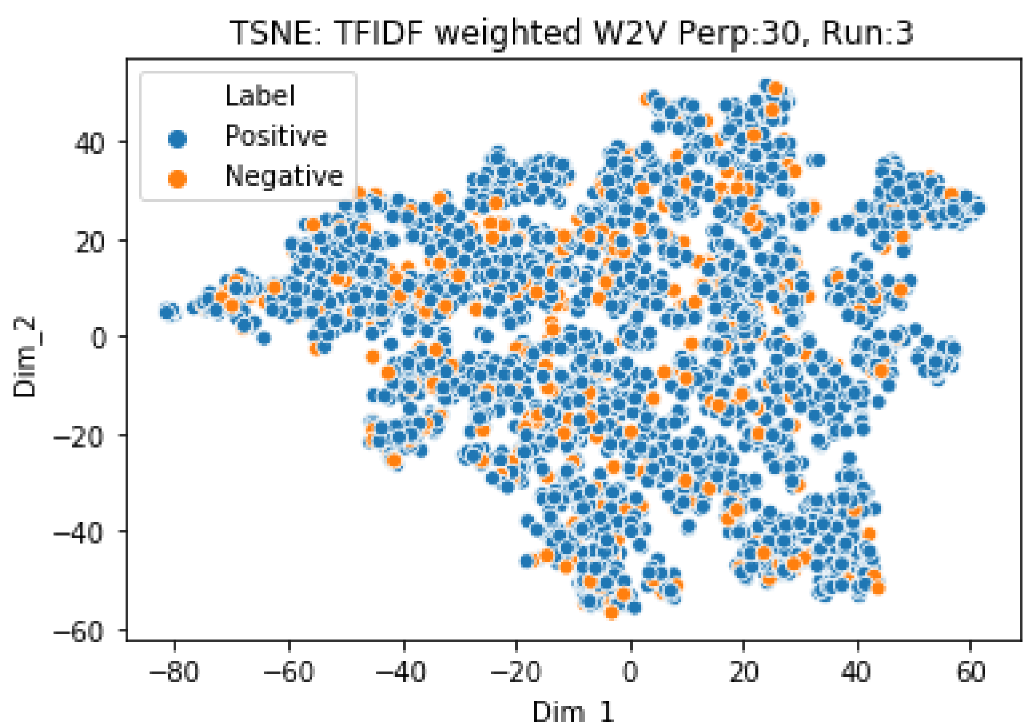


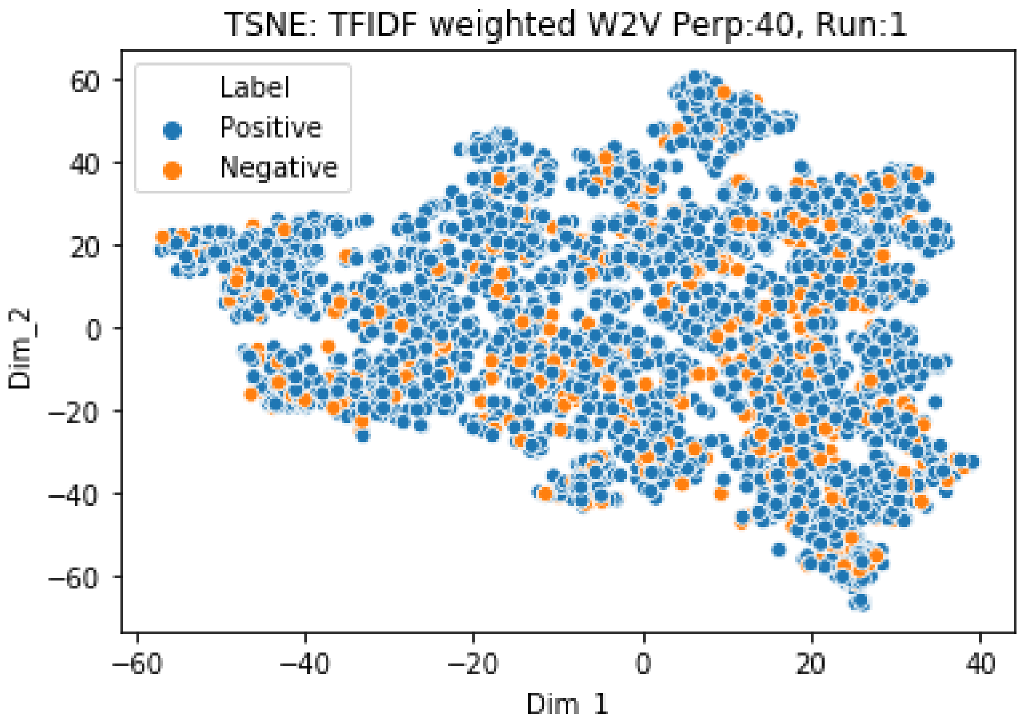
TSNE: TFIDF weighted W2V Perp:20, Run:3
Label
Positive
Negative
60
40
20
0
−20
−40
−60
Dim_2
−100
−80
−60
−40
−20
0
20
40
60
Dim_1

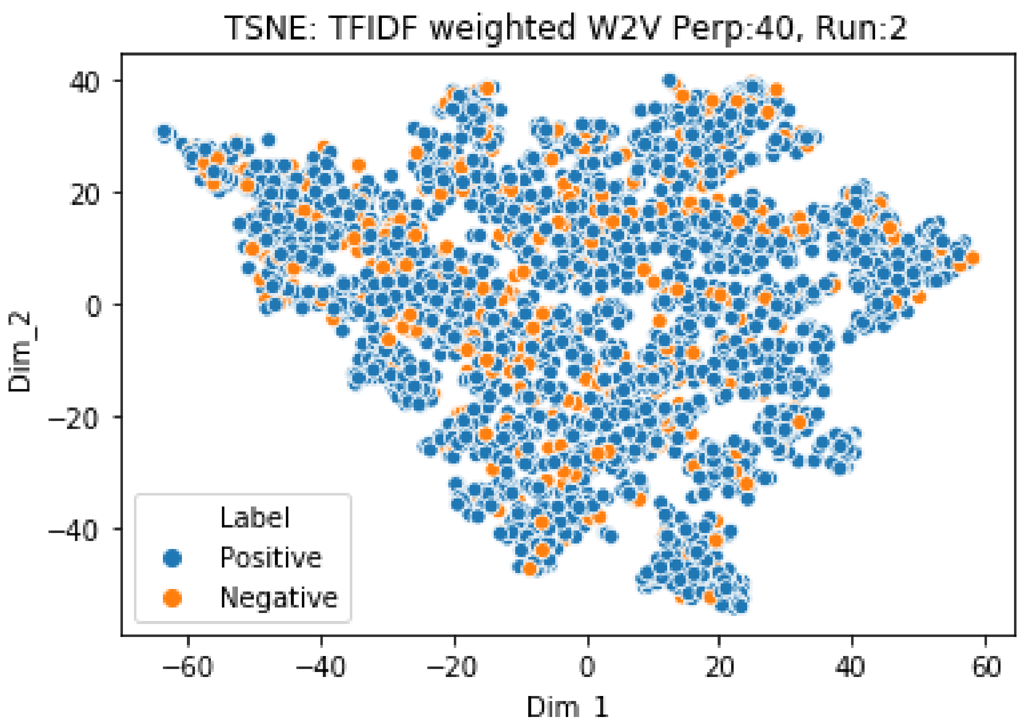


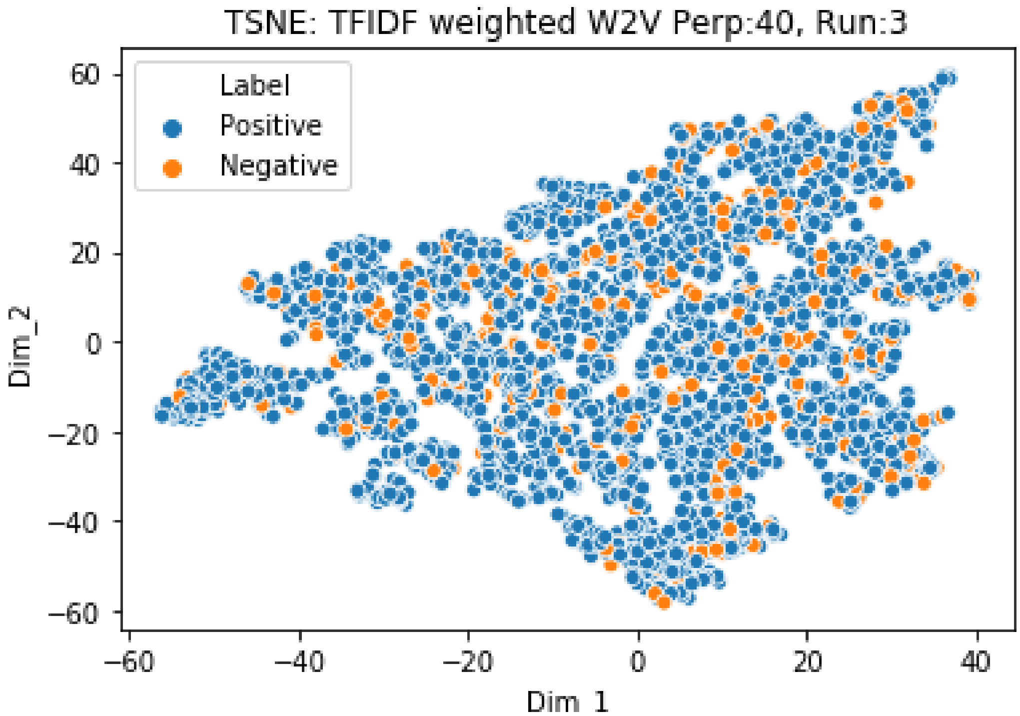
TSNE: TFIDF weighted W2V Perp:30, Run:1
Label
Positive
Negative
80
60
40
20
0
−20
−40
−60
Dim_2
−60
−40
−20
0
20
40
Dim_1


TSNE: TFIDF weighted W2V Perp:30, Run:2
Label
Positive
Negative
60
40
20
0
−20
−40
−60
Dim_2
−60
−40
−20
0
20
40
Dim_1


TSNE: TFIDF weighted W2V Perp:30, Run:3
Label
Positive
Negative
40
20
0
−20
−40
−60
Dim_2
−80
−60
−40
−20
0
20
40
60
Dim_1


TSNE: TFIDF weighted W2V Perp:40, Run:1
Label
Positive
Negative
Dim_2
Dim_1


TSNE: TFIDF weighted W2V Perp:40, Run:2
Label
Positive
Negative
Dim_2
Dim_1



Observations
None of the plots showed any significant separation between poistive class and negative class

7 [6] Conclusions

Observations
None of the plots of any vectorization method showed any significant separation between poistive class and negative class
We need to try more complex models as the points are difficult to separate