02 Amazon Fine Food Reviews Analysis_TSNE

June 11, 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 [14]: %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.manifold import TSNE
         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

         from bs4 import BeautifulSoup
```

2 [1]. Reading Data

```
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
        # SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
        # you can change the number to any other number based on your computing power
```

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 400

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative r
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 (4000, 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...
```

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

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

```
(80668, 7)
```

```
Out[4]:                  UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

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

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

```
Out[5]:               UserId   ProductId                       ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

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

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

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

3 Exploratory Data Analysis

3.1 [2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

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

```
Out[7]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2
```

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

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

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

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and 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=Fals
```

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

```
Out[9]: (3989, 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]: 99.725
```

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()
```

```
(3989, 10)
```

Out[13]:
```
1    3328
0     661
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.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 [123]:
```
# 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 [15]: final['Text'].head()

Out[15]: 2546    Why is this $[...] when the same product is av...
         2547    We have used the Victor fly bait for 3 seasons...
         1145    I just received my shipment and could hardly w...
         1146    This was a really good idea and the final prod...
         2942    I'm glad my 45lb cocker/standard poodle puppy ...
         Name: Text, dtype: object
```

```
In [17]: # I was trying out some other ways

         """
         def remove_url(sentence): #function to remove the url from the review
             remove = re.compile(r"http\S+")
             removed = re.sub(remove, '', sentence)
             return removed

         length = len(final["Text"])

         for i in range(length):
             final["Text"].values[i] = remove_url(final["Text"].values[i])

         print(final["Text"].head())
         """

2546    Why is this $[...] when the same product is av...
2547    We have used the Victor fly bait for 3 seasons...
1145    I just received my shipment and could hardly w...
1146    This was a really good idea and the final prod...
2942    I'm glad my 45lb cocker/standard poodle puppy ...
Name: Text, dtype: object
```

```
In [19]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all


         for i in range(length):
             soup = BeautifulSoup(final["Text"].values[i], 'lxml')
             final["Text"].values[i] = soup.get_text()

         print(final["Text"].values[0])
         print("="*50)


         """
         soup = BeautifulSoup(final["Text"].values[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
==================================================
```

```
Out[19]: '\nsoup = BeautifulSoup(final["Text"].values[1000], \'lxml\')\ntext = soup.get_text()
```

```
In [15]: # 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 [21]: for i in range(length):
             final["Text"].values[i] = decontracted(final["Text"].values[i])

         print(final["Text"].values[4])
```

```
I am glad my 45lb cocker/standard poodle puppy loves the stuff because I trust the brand and i
```

```
In [22]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039

         for i in range(length):
             final["Text"].values[i] = re.sub("\S*\d\S*", "", final["Text"].values[i]).strip()
```

```
print(final["Text"].values[0])
```

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

```
In [23]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         for i in range(length):
             final["Text"].values[i] = re.sub('[^A-Za-z0-9]+', ' ', final["Text"].values[i])

         print(final["Text"].values[0])
```

```
Why is this when the same product is available for here The Victor and traps are unreal of cou
```

```
In [16]: # 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 [17]: # 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%|| 3989/3989 [00:02<00:00, 1601.28it/s]
```

```
In [26]: preprocessed_reviews[1]
```

```
Out[26]: 'used victor fly bait seasons ca not beat great product'
```

[3.2] Preprocess Summary

```
In [18]: ## Similartly you can do preprocessing for review summary also.

         # Combining all the above stundents
         preprocessed_summaries = []
         # tqdm is for printing the status bar

         for sentance in tqdm(final['Summary'].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 stopw
             preprocessed_summaries.append(sentance.strip())
```

```
100%|| 3989/3989 [00:01<00:00, 2510.98it/s]
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [19]: #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  ['aahhhs', 'aback', 'abates', 'abby', 'abdominal', 'abiding', 'ability', 'a
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (3989, 11520)
the number of unique words  11520
```

5.2 [4.2] TF-IDF

In [34]:
```
# TF-IDF scikit implementation

tf_idf_vect = TfidfVectorizer( min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_name
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) ['able', 'absolute', 'absolutely', 'according
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (3989, 1885)
the number of unique words including both unigrams and bigrams  1885
```

5.3 [4.4] Word2Vec

In [20]:
```
# Train your own Word2Vec model using your own text corpus

list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [21]:
```
# 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=-1)
    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.bi
        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, 
```

```
[('supply', 0.4664269685745239), ('quinoa', 0.4380897879600525), ('split', 0.423034131526947), 
==================================================
[('wrapper', 0.5002533793449402), ('experiment', 0.45653849840164185), ('fresh', 0.43980947136
```

In [22]:
```
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  3295
sample words  ['used', 'ca', 'not', 'beat', 'great', 'product', 'available', 'course', 'total'
```

5.4 [4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [23]:
```
# 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 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%|| 3989/3989 [00:05<00:00, 765.01it/s]
```

```
3989
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [24]: # 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 [25]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
         row=0;
         for sent in tqdm(list_of_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%|| 3989/3989 [00:27<00:00, 143.69it/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>
```

6.1 [5.1] Applying TNSE on Text BOW vectors

In [26]:
```
# please write all the code with proper documentation, and proper titles for each sub
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

print("the type of count vectorizer ",type(final_counts))

#convert sparse matrix into dense matrix
final_counts_dense = final_counts.todense()

print("the type of count vectorizer ",type(final_counts_dense))
```

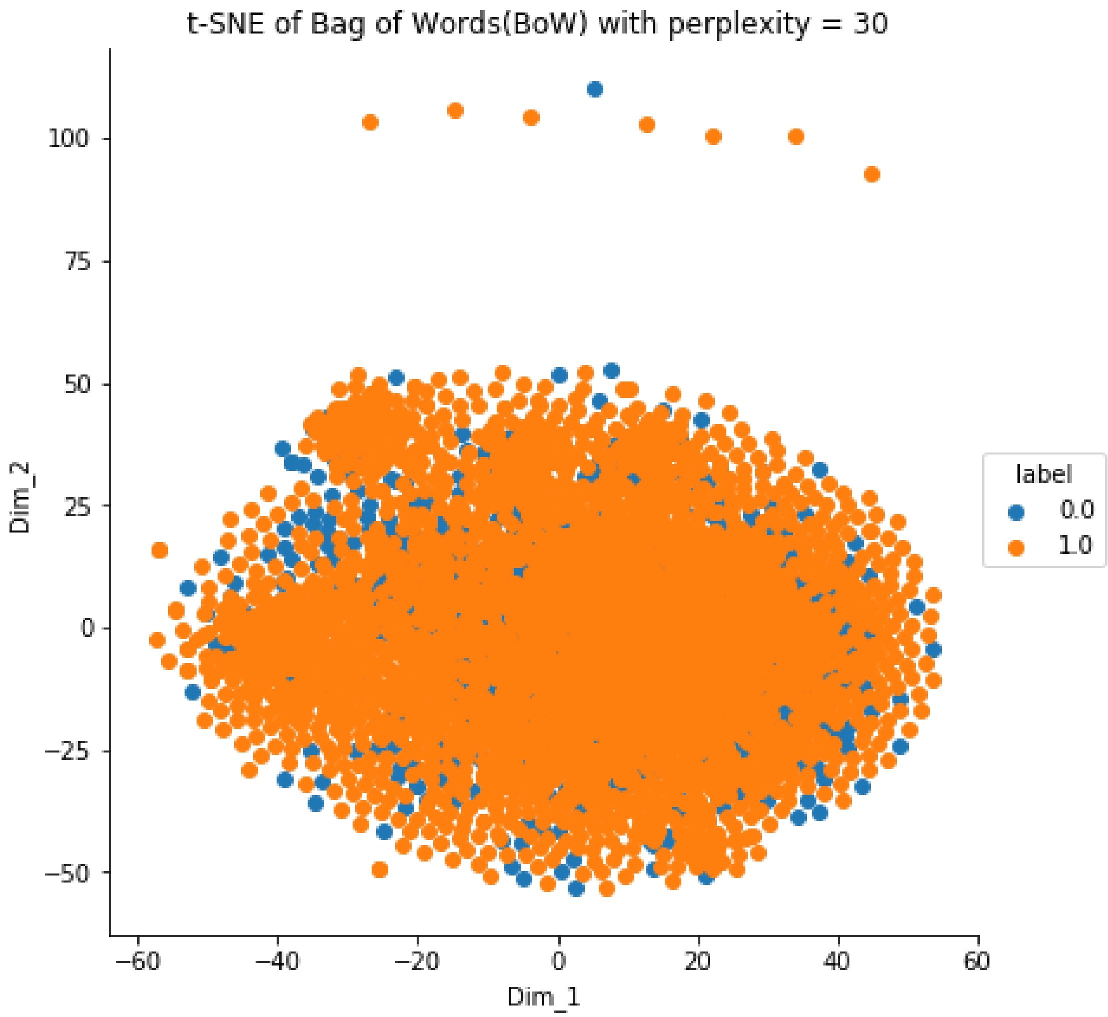
```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'numpy.matrix'>
```

In [27]:
```
final_counts_dense.shape
#filtered_data["Score"].shape
```

Out[27]: (3989, 11520)

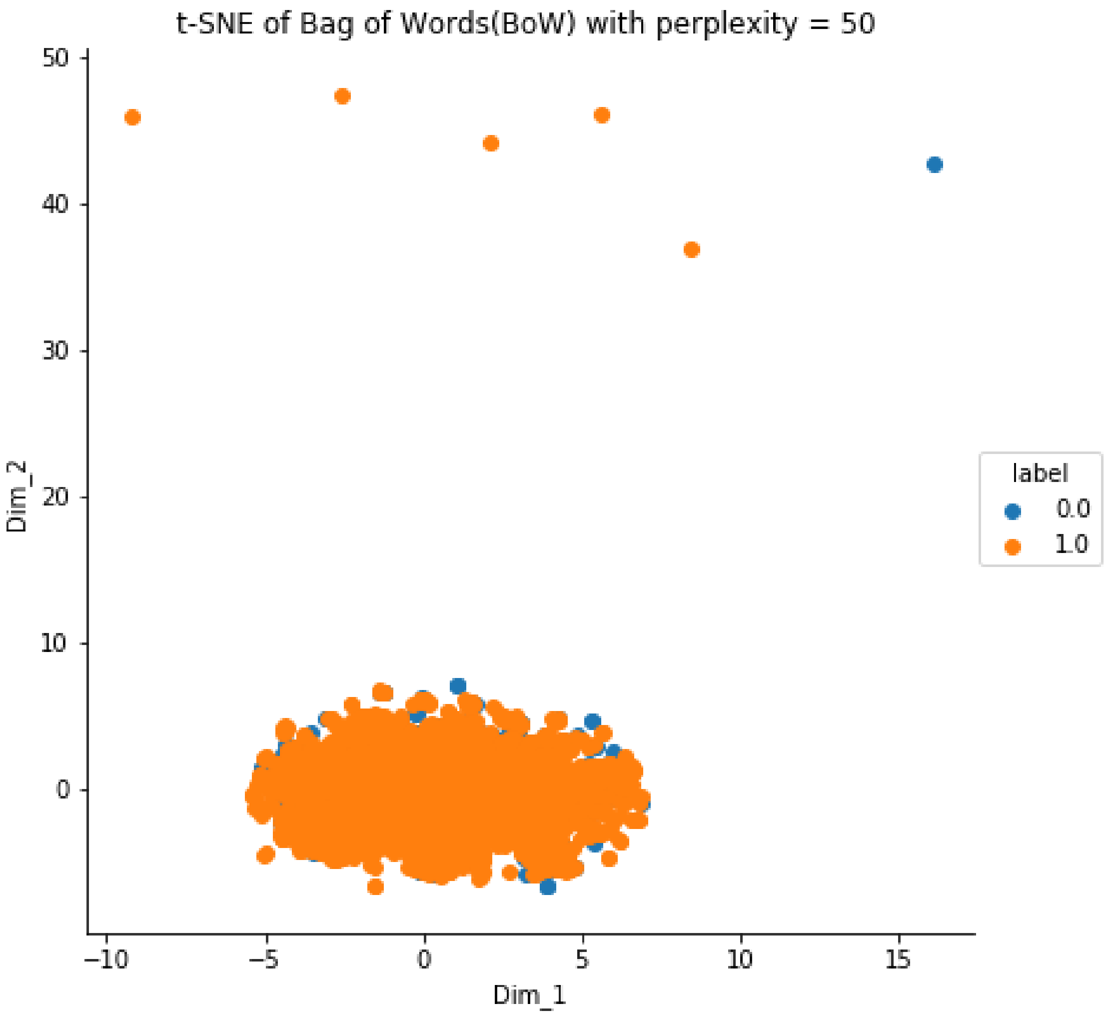
In [29]:
```
# Picking the top 1000 points as TSNE takes a lot of time for 5K points
#data_1000 = standardized_data[0:1000,:]
#labels_1000 = labels[0:1000]

data_bow = final_counts_dense
labels = final["Score"]


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

```
#tsne_data = model.fit_transform(data_1000)
tsne_data = model.fit_transform(data_bow)


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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of Bag of Words(BoW) with perplexity = 30 ")
plt.show()
```

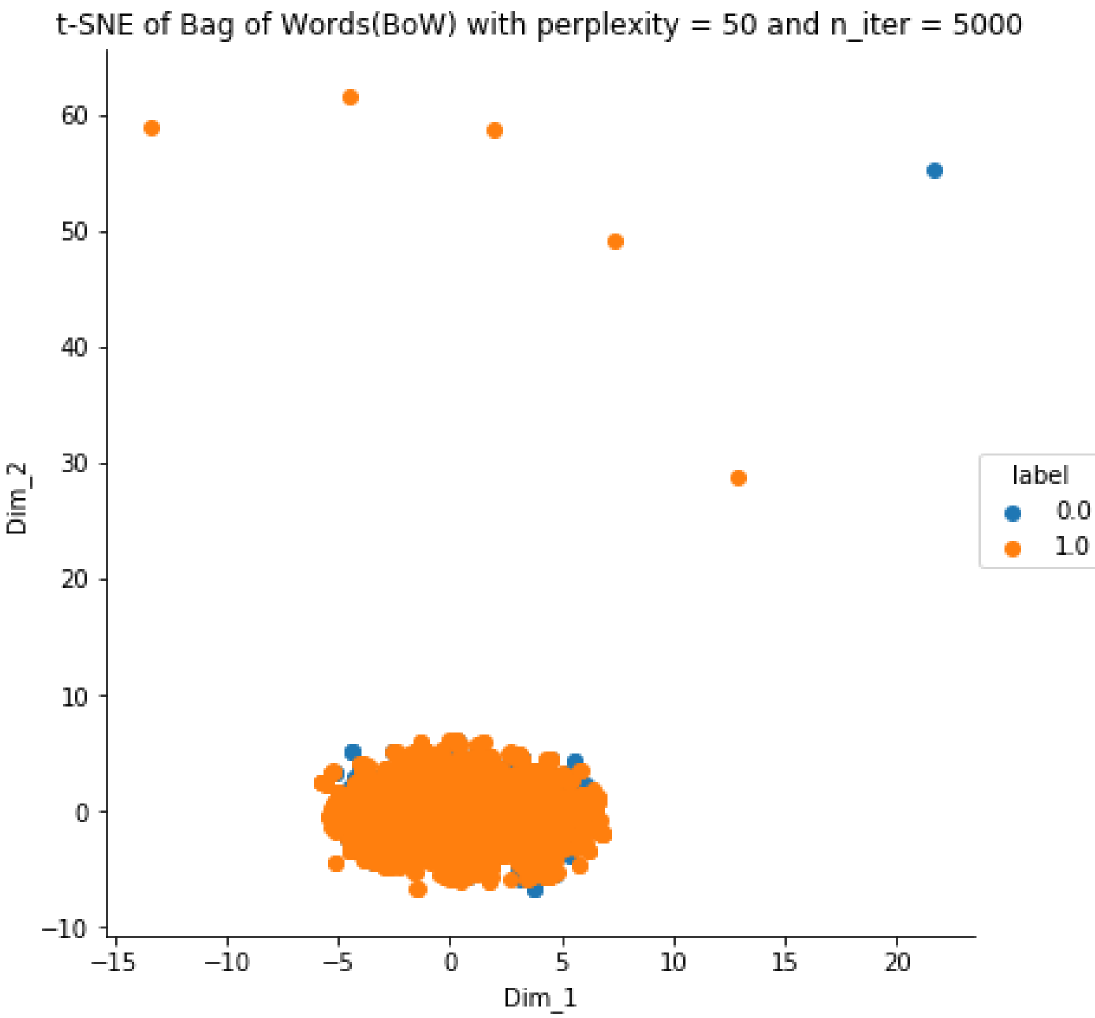


In [30]:
```
data_bow = final_counts_dense
labels = final["Score"]
```

In [31]:
```
# with perplexity 50

model = TSNE(n_components=2, random_state=0, perplexity=50)
tsne_data = model.fit_transform(data_bow)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of Bag of Words(BoW) with perplexity = 50 ")
plt.show()
```



In [32]:
```
# with perplexity 5o and 5000 number of iterations

model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
tsne_data = model.fit_transform(data_bow)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_le
plt.title("t-SNE of Bag of Words(BoW) with perplexity = 50 and n_iter = 5000" )
plt.show()
```



6.2 [5.2] Applying TNSE on Text TFIDF vectors

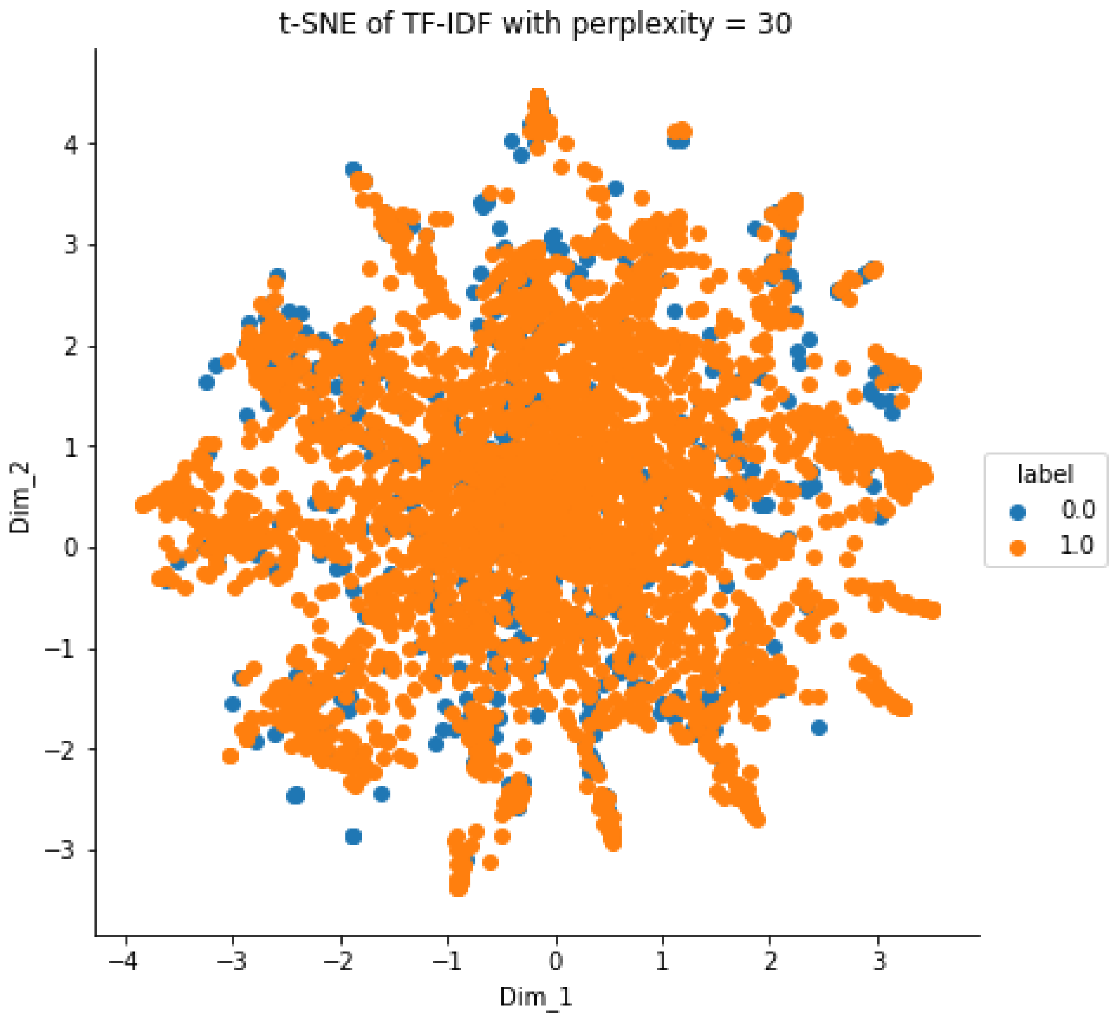
In [35]:
```
# please write all the code with proper documentation, and proper titles for each sub
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

print("the type of count vectorizer ",type(final_tf_idf))

#convert sparse matrix into dense matrix
final_tf_idf_dense = final_tf_idf.todense()

print("the type of count vectorizer ",type(final_tf_idf_dense))
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer  <class 'numpy.matrix'>
```

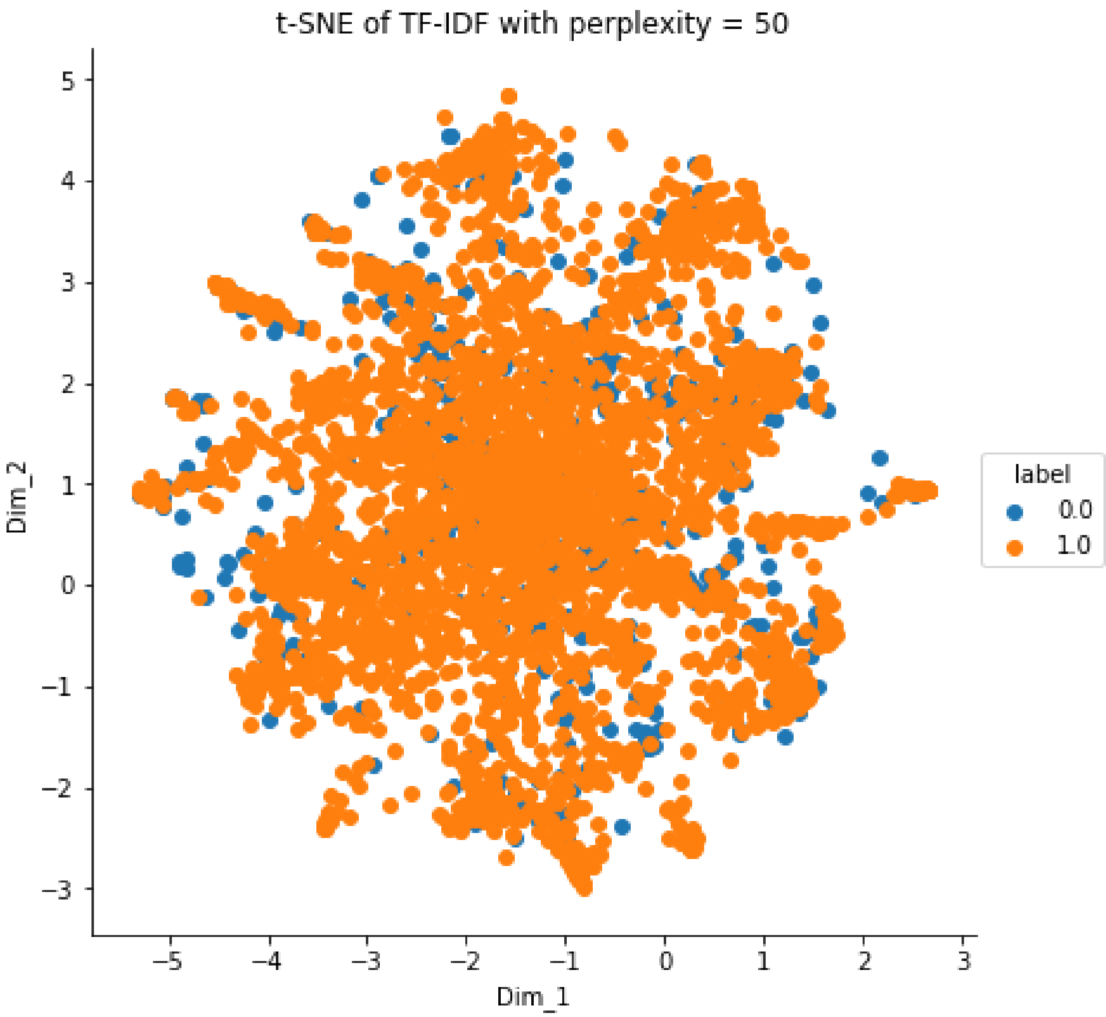
In [36]:
```
final_tf_idf_dense.shape
```

Out[36]:
```
(3989, 1885)
```

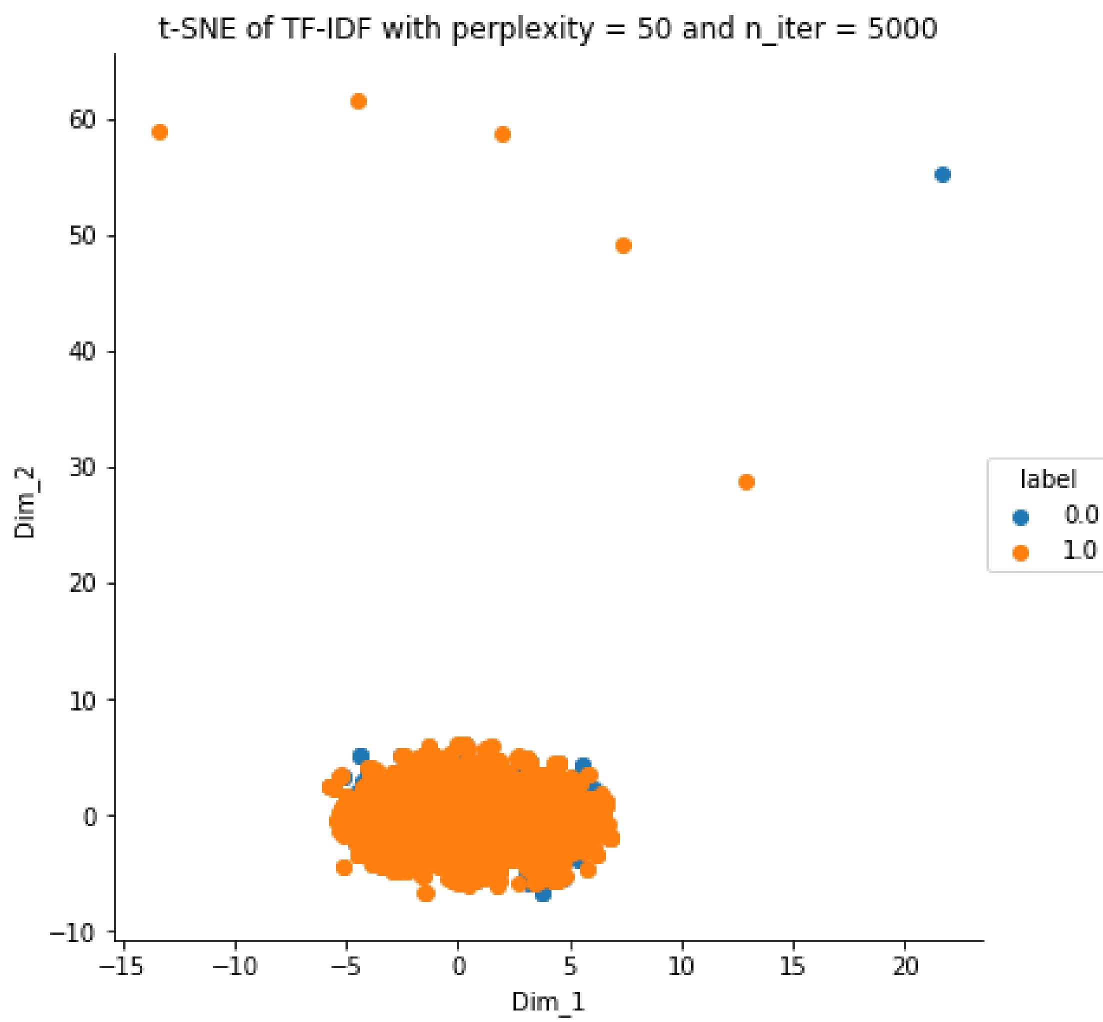
In [38]:
```
data_tfidf = final_tf_idf_dense
labels = final["Score"]


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

#tsne_data = model.fit_transform(data_1000)
tsne_data = model.fit_transform(data_tfidf)


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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of TF-IDF with perplexity = 30")
plt.show()
```



In [41]:
```
# tsne tfidf with perplexity 50

model = TSNE(n_components=2, random_state=0, perplexity=50)
tsne_data = model.fit_transform(data_tfidf)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of TF-IDF with perplexity = 50")
plt.show()
```



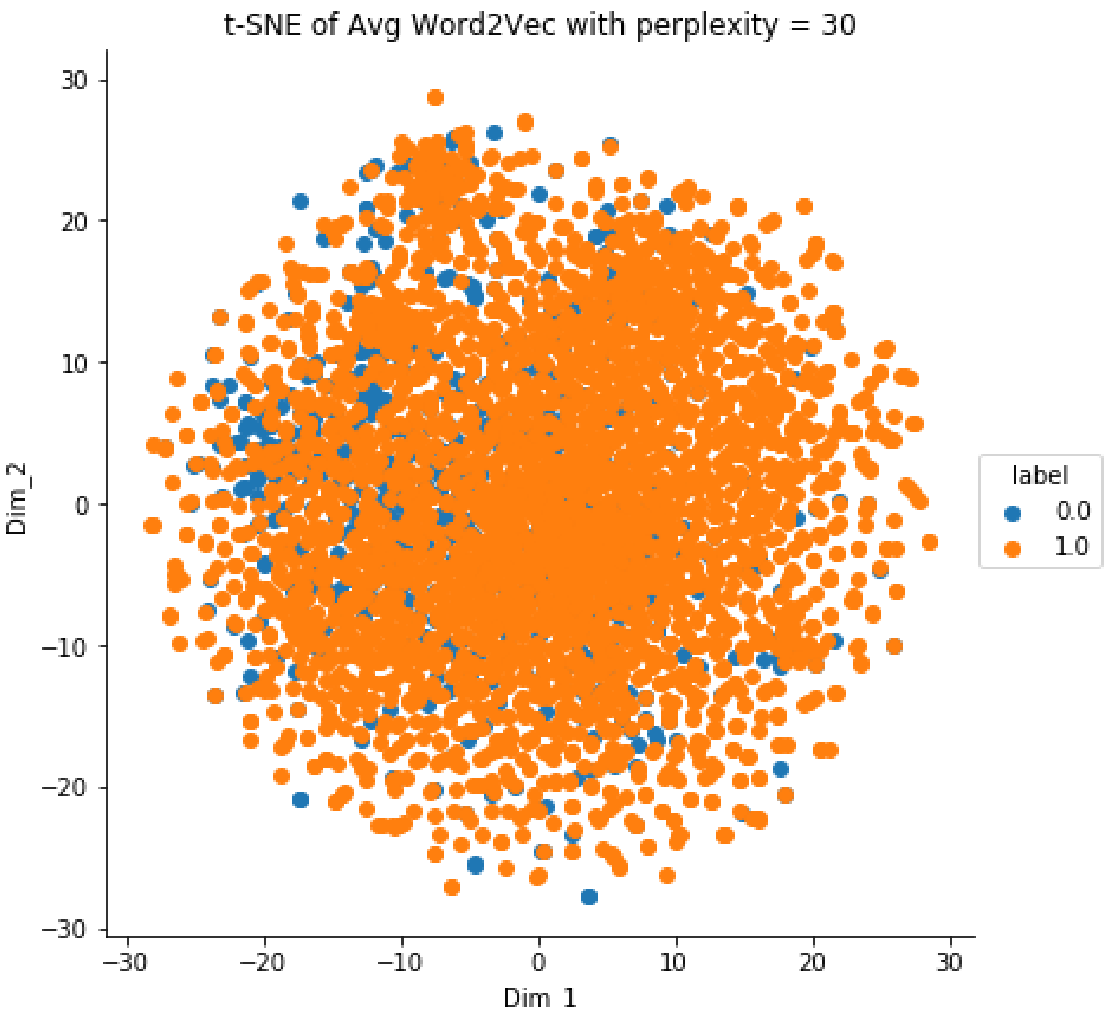
In [42]:
```
# with perplexity 5o and 5000 number of iterations

model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
tsne_data = model.fit_transform(data_bow)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title('t-SNE of TF-IDF with perplexity = 50 and n_iter = 5000')
plt.show()
```



6.3 [5.3] Applying TNSE on Text Avg W2V vectors

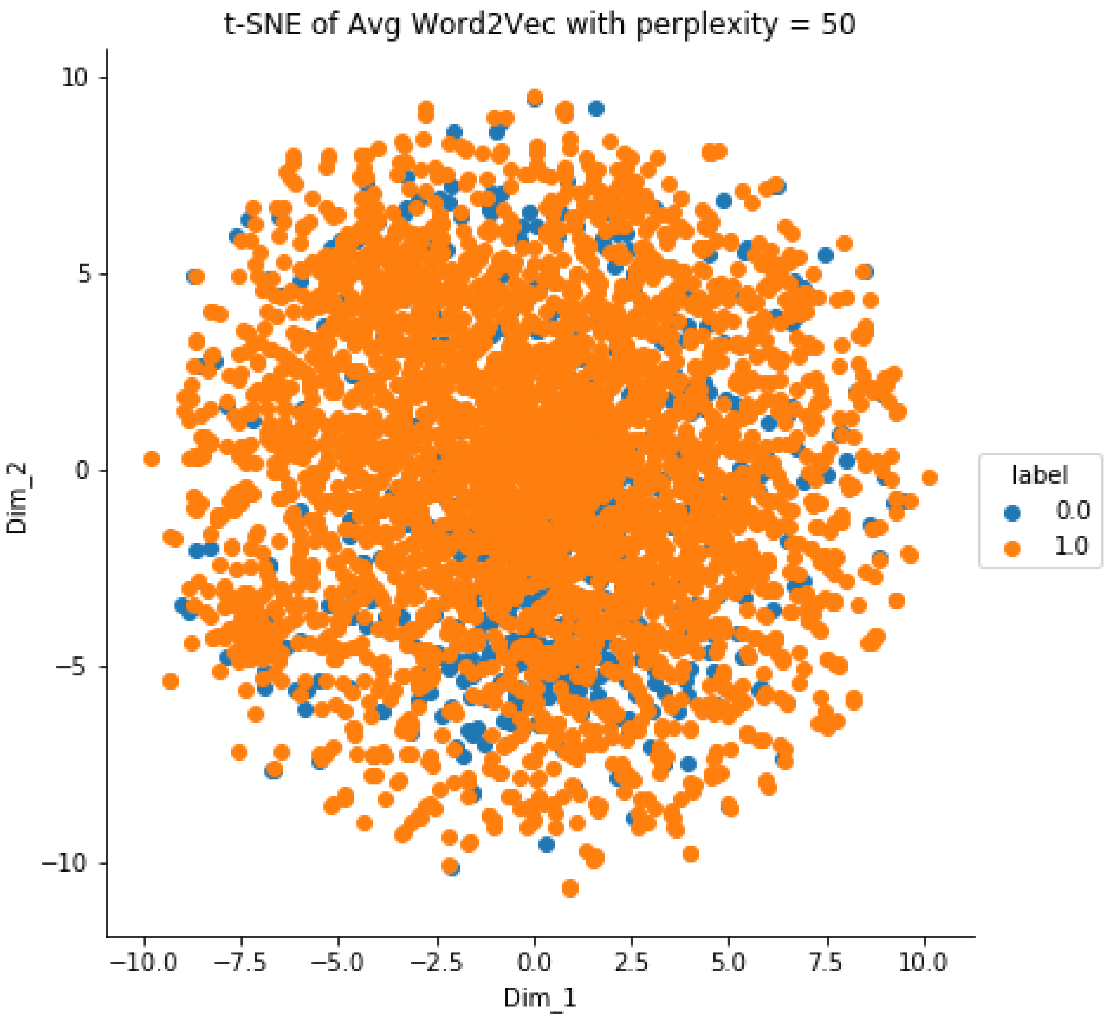
In [44]:
```
# please write all the code with proper documentation, and proper titles for each sub
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label



data_aw2v = sent_vectors
labels = final["Score"]
```

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

tsne_data = model.fit_transform(data_aw2v)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of Avg Word2Vec with perplexity = 30")
plt.show()
```



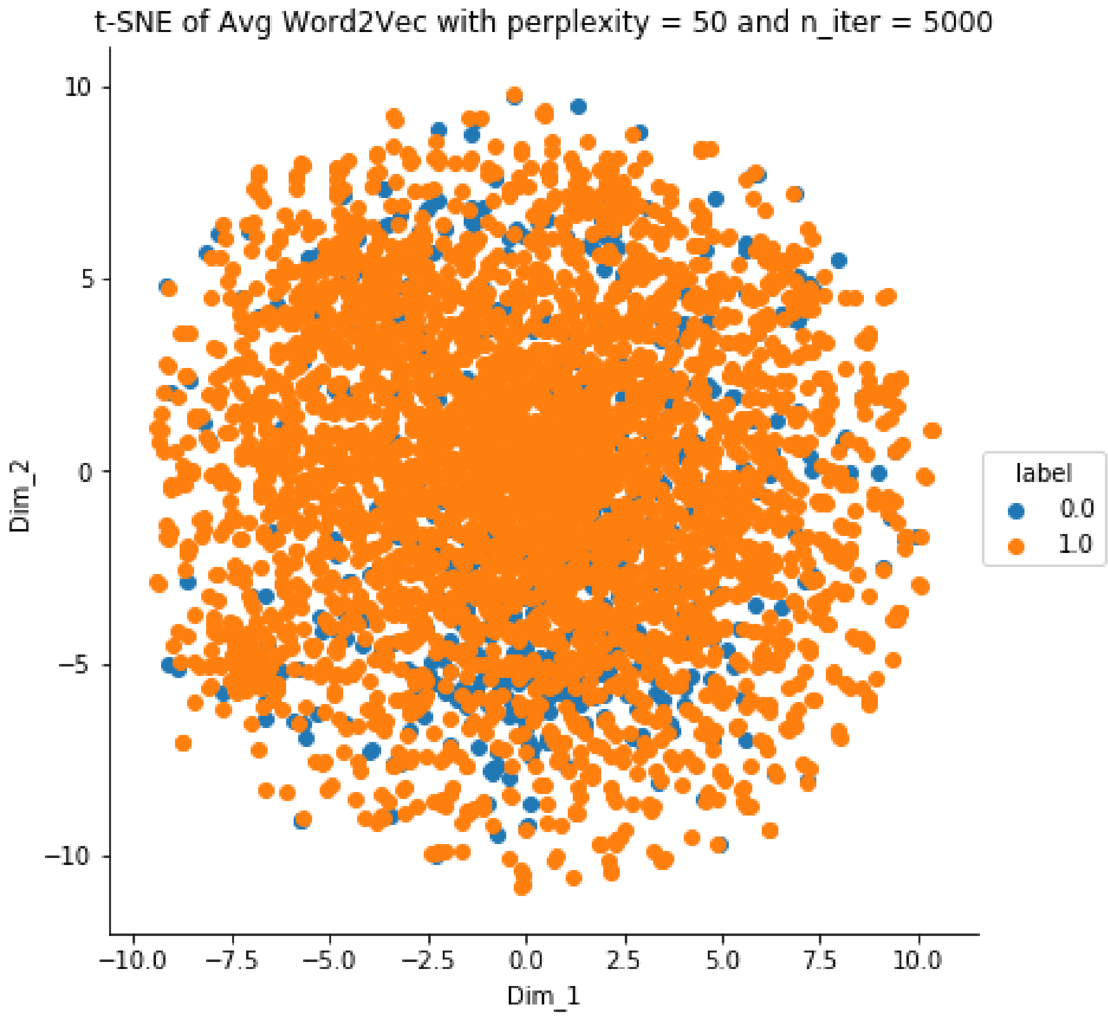
In [45]:
```
data_aw2v = sent_vectors
labels = final["Score"]
```

In [46]:
```
# with perplexity 50

model = TSNE(n_components=2, random_state=0, perplexity=50)
tsne_data = model.fit_transform(data_aw2v)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title('t-SNE of Avg Word2Vec with perplexity = 50')
plt.show()
```



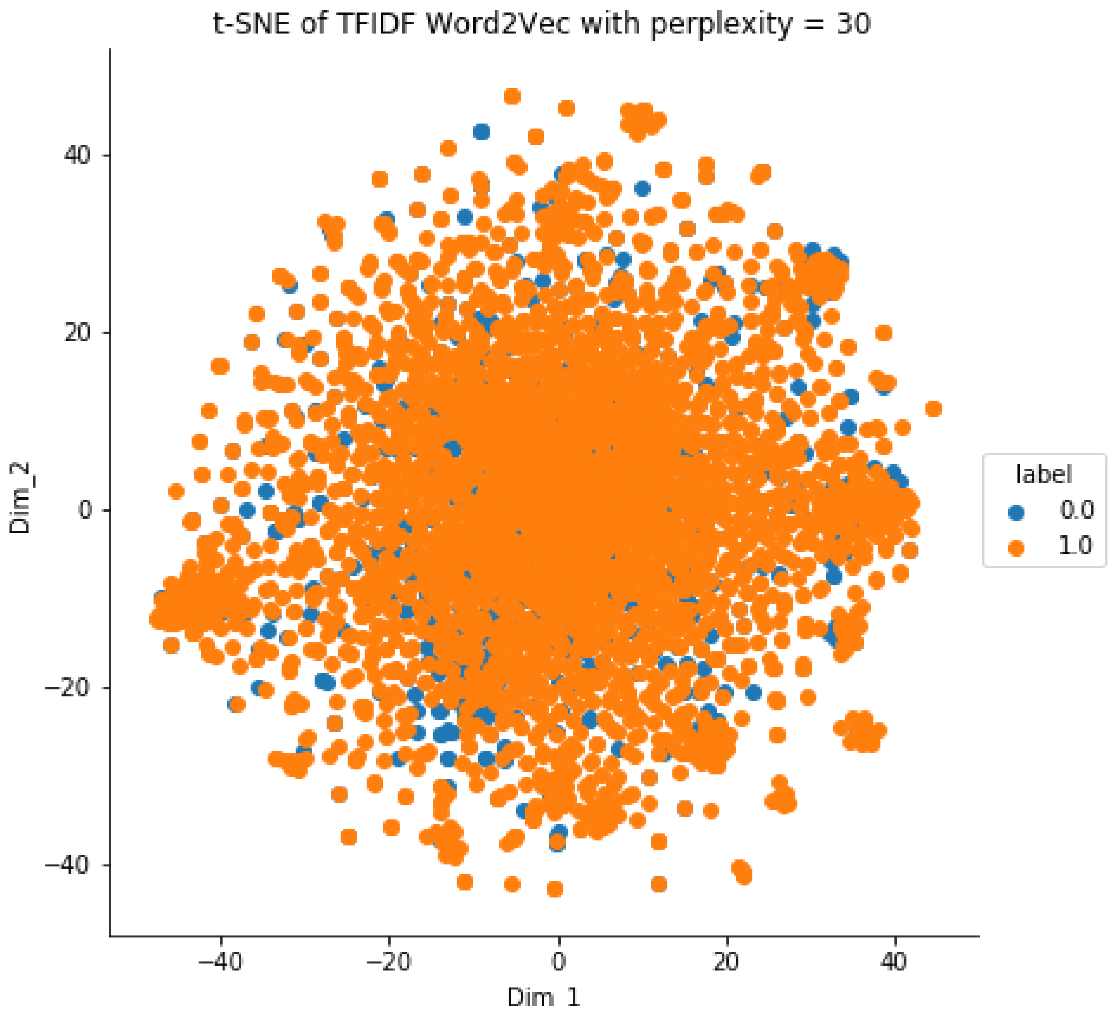
In [47]: # with perplexity 5o and 5000 number of iterations

```
model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
tsne_data = model.fit_transform(data_aw2v)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title('t-SNE of Avg Word2Vec with perplexity = 50 and n_iter = 5000')
plt.show()
```

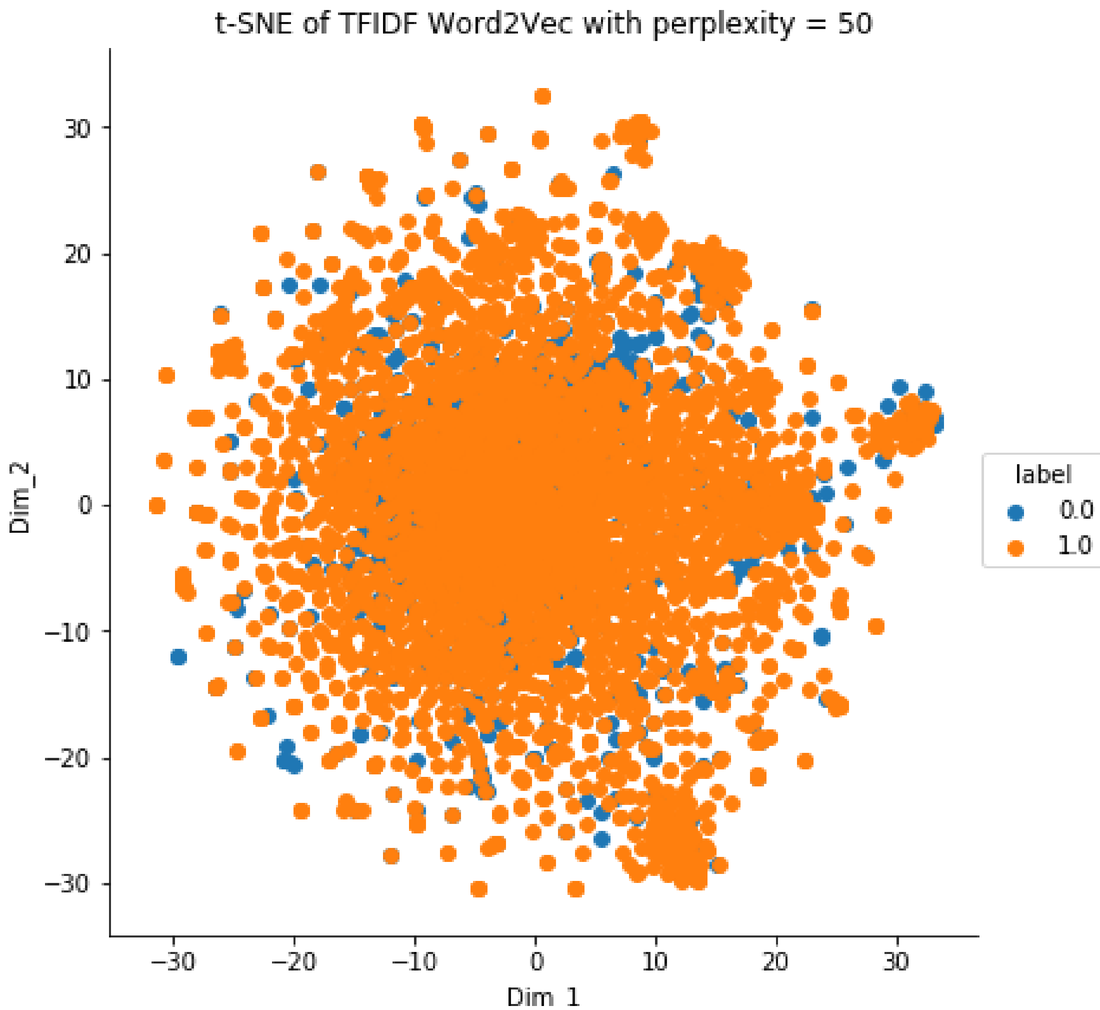


6.4 [5.4] Applying TNSE on Text TFIDF weighted W2V vectors

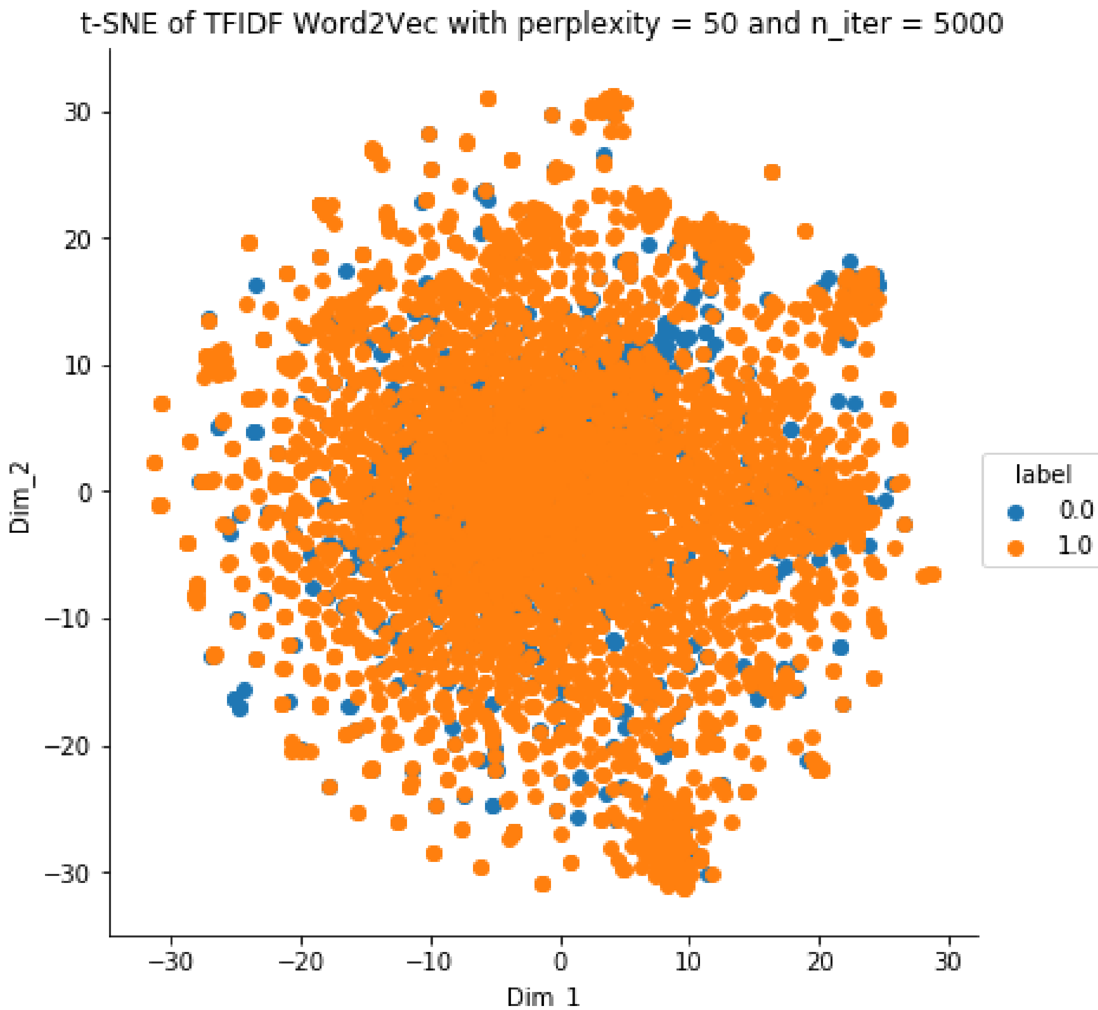
In [48]:
```
# please write all the code with proper documentation, and proper titles for each sub
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label


data_tfidfw2v = tfidf_sent_vectors
labels = final["Score"]


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

#tsne_data = model.fit_transform(data_1000)
tsne_data = model.fit_transform(data_tfidfw2v)


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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title("t-SNE of TFIDF Word2Vec with perplexity = 30")
plt.show()
```




```
In [49]: data_tfidfw2v = tfidf_sent_vectors
         labels = final["Score"]

In [50]: # with perplexity 50

         model = TSNE(n_components=2, random_state=0, perplexity=50)
         tsne_data = model.fit_transform(data_tfidfw2v)

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

         # Ploting the result of tsne
         sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
         plt.title('t-SNE of TFIDF Word2Vec with perplexity = 50')
         plt.show()
```



In [52]:
```
# with perplexity 5o and 5000 number of iterations

model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
tsne_data = model.fit_transform(data_tfidfw2v)

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

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_leg
plt.title('t-SNE of TFIDF Word2Vec with perplexity = 50 and n_iter = 5000')
plt.show()
```



7 [6] Conclusions

1. We have taken 4k points here, I have also tried 5k points earlier. But to speed up the process later reduced the points to 4k.
2. We have tried t-sne plots with various perplexity values and different number of iterations.
3. But even with different perplexity values and different number of iterations we are unable to completely seprate the positive and negative data points.
4. Positive and negative data points almost overlapping in each and every t-sne plots for each vectorizations.
5. Avg W2V and TF-IDF W2V are better vectorizations because they tend to remove outliers better than as compared to BOW and TF-IDF.