Amazon Fine Food Reviews Preprocessing

This IPython notebook consists code for preprocessing of text, conversion of text into vectors and saving that information for further use.

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

Public Information -

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

1. Number of reviews: 568,454
2. Number of users: 256,059
3. Number of products: 74,258
4. Timespan: Oct 1999 - Oct 2012
5. 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

Current Objective -

Go through the reviews and perform preprocessing, convert them into vectors and save them for future use.

Let's start with mounting paths and importing necessary libraries

In [0]:

```
#mounting the dataset from drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2
F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.co
m%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.re
adonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&respon
se_type=code (https://accounts.google.com/o/oauth2/auth?client_id=9473189898
03-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=
urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleap
is.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
#importing necessary libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import missingno as msno
```

In [0]:

```
#connecting to sqlite db
con = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/d

#filtering only positive and negative reviews
data = pd.read_sql_query("SELECT * FROM Reviews WHERE Score != 3", con)

print("Shape of data:", data.shape)

#scores < 3 are considered to be negative reviews and > 3 are considered to be positive rev
data.head()
```

```
Shape of data: (525814, 10)
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

Missing values? Yeah, it happens...

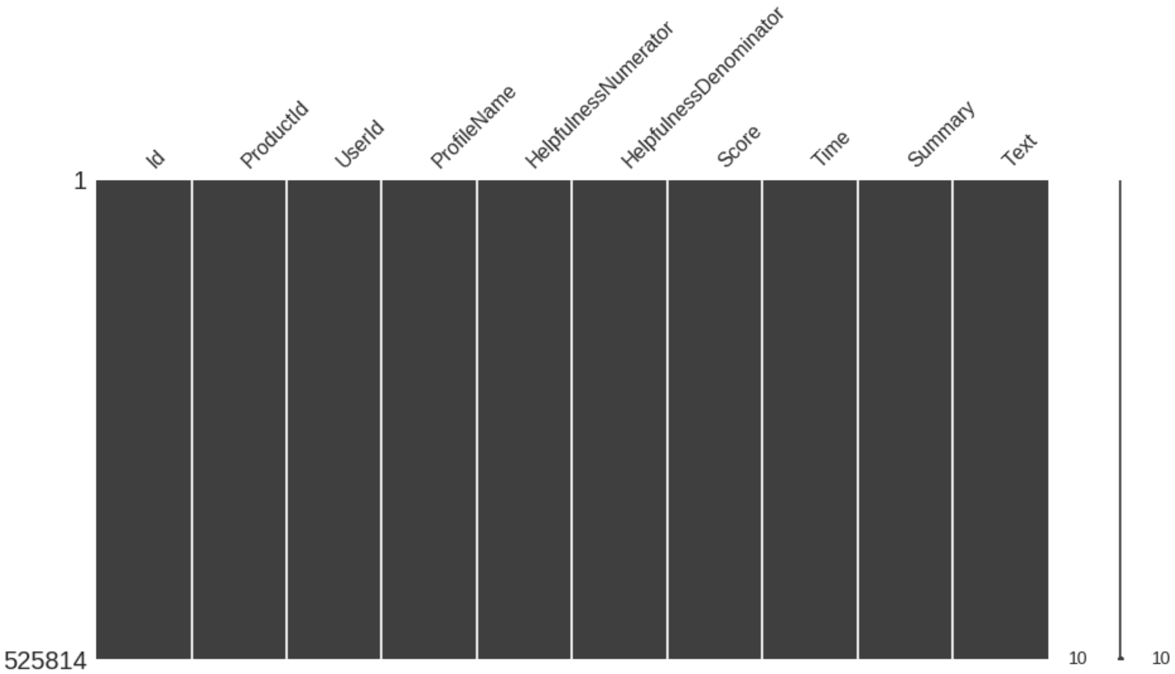
In [0]:

```
#let's just check, just in case if any
print("Missing values? Ans -", data.isnull().values.any())

#visualizing it
msno.matrix(data, figsize=(15,7))
```

```
Missing values? Ans - False
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f35bf4505f8>
```



One can write more than one review for the same product!

In [0]:

```
df = data.copy()
df['ProdUser'] = df['ProductId'] + df['UserId']
df[df['ProdUser'].duplicated(keep=False)]. \
    sort_values('ProdUser', axis=0, ascending=True, inplace=False, kind='quicksort', na_pos
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
157863	171174	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	0	
157871	171183	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	5	
157912	171228	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5	
157841	171152	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	

Obeservations

1. There are some instances where a user has written more than one review for the same product.
2. We can remove the one which has less Helpfulness but lets keep all and treat it as review from a different user.
3. Will definitely have to remove same reviews because it is just redundant data.

In [0]:

```
#Sorting data according to ProductId in ascending order
data = data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort
```

In [0]:

```
#Deduplication of entries
data=data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
data.shape
```

Out[9]:

```
(364173, 10)
```

In [0]:

```
data.head(2)
```

Out[10]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	

Same reviews on multiple products with different timestamps!!

In [0]:

```
data[data['Text'].duplicated(keep=False)]. \
    sort_values('Text', axis=0, ascending=True, inplace=False, kind='quicksort', na_positic
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenoı
67574	73444	B0046IISFG	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1	
287090	311004	B001EO6FPU	A3OXHLG6DIBRW8	C. F. Hill "CFH"	9	
302818	327982	B0000CEQ6H	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca	3	

In [0]:

```
#removing duplicate reviews
data=data.drop_duplicates(subset={"Text"}, keep='first', inplace=False)
data.shape
```

Out[12]:

```
(363836, 10)
```

Observations

1. There are reviews which are same on similar products (mostly different flavors).
2. These reviews were posted with different timestamps by the same person (weird).
3. Since we are interested in a review being positive or negative, having redundant reviews makes no sense, so removing them.

In [0]:

```
#also removing those reviews where HelpfulnessNumerator is greater than HelpfulnessDenomina
data=data[data['HelpfulnessNumerator']<=data['HelpfulnessDenominator']]
data.shape
```

Out[13]:

```
(363834, 10)
```

In [0]:

```
# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'
```

In [0]:

```
actualScore = data['Score']
positiveNegative = actualScore.map(partition)
data['Score'] = positiveNegative
print("Negatives shape:", data[data['Score']=='negative'].shape)
print("Positives shape:", data[data['Score']=='positive'].shape)
```

```
Negatives shape: (57070, 10)
Positives shape: (306764, 10)
```

Now, it's time for some Text Preprocessing

We will be doing the following in order.

1. Text cleaning - includes removal of special characters which are not required.
2. Check if the word is actually an English word.
3. 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)
4. Convert the word to lower case.
5. Remove stop words but let's keep words like 'not' which makes the sentence negative.

6. POS Tagging and WordNet Lemmatizing the word.

In [0]:

```
from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
```

In [0]:

```
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characte
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [0]:

```
!python -m nltk.downloader stopwords
```

```
/usr/lib/python3.6/runpy.py:125: RuntimeWarning: 'nltk.downloader' found in
sys.modules after import of package 'nltk', but prior to execution of 'nltk.
downloader'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [0]:

```
stop = list(set(stopwords.words('english'))) #set of stopwords
print(stop)

#removing words like 'not' that gives negative meaning to a sentence from stopwords
important_stopwords = ['hadn', 'weren', 'shouldn', "needn't", 'needn', 'doesn', "shan't", '
                       'mustn', "hadn't", "doesn't",
                      'wouldn', "weren't", "didn", "mustn't", "wasn't", "didn't", "don't",
pre_final_stops = [x for x in stop if x not in important_stopwords]

#removing punctuation from stop words
final_stops = list(set([cleanpunc(x) for x in pre_final_stops]))
print("Final stopwords:", final_stops)
```

```
["it's", 'more', 'just', 'couldn', 'and', 'our', "hasn't", 'this', 've', 'of
f', 'needn', 'themselves', 'on', 'now', 'own', 'before', 'yourself', 'i', 'd
id', 'didn', 'from', "weren't", 'out', "that'll", 'during', "hadn't", 'thes
e', 'but', 'nor', 'don', 'his', 'are', 'an', 'hadn', 'because', 'very', "wou
ldn't", "needn't", 'as', 'where', 'too', 'shan', 'them', 'she', 'was', 'th
e', 'can', 'who', 'y', 'weren', 're', "haven't", 'whom', 'been', "won't", 'y
ou', 'down', 'until', 't', 'd', 'my', 'won', 'through', 'that', "you'll", 'd
oes', 'both', "couldn't", 'himself', 'ours', 'being', 'what', 'for', 'when',
'once', 'were', "doesn't", 'again', 'am', 'then', 'so', "mightn't", "sha
n't", 'than', 'how', 'any', 'your', 'doing', 'here', 'ourselves', 'between',
"you're", "should've", 'there', 'myself', "you've", 'hers', 'which', 'unde
r', 'same', 'against', 'will', "shouldn't", 'not', 'each', "wasn't", 'over',
'why', 'those', 'further', 'about', 'me', 'yours', 'should', "you'd", 'shoul
dn', 'other', "she's", 'herself', "don't", "aren't", 'up', 'wouldn', 'in',
's', 'it', 'be', 'have', 'ma', 'has', 'is', 'her', 'few', 'all', 'such', 'ha
ven', 'we', 'theirs', 'having', 'only', 'do', "mustn't", 'had', 'yourselve
s', 'after', 'by', 'mightn', 'm', 'its', 'some', 'below', 'most', 'o', 'he',
'above', 'a', 'their', 'wasn', 'isn', 'itself', 'if', 'or', 'no', 'while',
'at', 'into', "didn't", 'll', 'with', 'to', "isn't", 'ain', 'of', 'doesn',
'him', 'hasn', 'aren', 'they', 'mustn']
Final stopwords: ['more', 'just', 'couldn', 'and', 'havent', 'our', 've', 't
his', 'off', 'themselves', 'on', 'now', 'own', 'before', 'yourself', 'did',
'i', 'from', 'out', 'during', 'these', 'but', 'nor', 'don', 'his', 'mightn
t', 'are', 'an', 'because', 'very', 'youd', 'isnt', 'where', 'as', 'arent',
'too', 'shan', 'them', 'she', 'was', 'the', 'can', 'shouldve', 'who', 'y',
're', 'youve', 'whom', 'been', 'you', 'down', 'until', 't', 'd', 'my', 'wo
n', 'through', 'that', 'does', 'both', 'himself', 'ours', 'being', 'what',
'wouldnt', 'for', 'when', 'once', 'wont', 'were', 'again', 'am', 'then', 's
o', 'than', 'how', 'any', 'your', 'doing', 'here', 'ourselves', 'between',
'there', 'myself', 'hers', 'which', 'under', 'same', 'against', 'will', 'eac
h', 'over', 'why', 'those', 'further', 'about', 'me', 'yours', 'should', 'ot
her', 'herself', 'up', 'in', 's', 'youre', 'it', 'shes', 'be', 'have', 'ma',
'has', 'is', 'her', 'few', 'all', 'such', 'haven', 'we', 'theirs', 'having',
'only', 'do', 'youll', 'had', 'yourselves', 'after', 'by', 'mightn', 'm', 'i
ts', 'some', 'below', 'most', 'o', 'he', 'above', 'a', 'their', 'isn', 'itse
lf', 'if', 'or', 'no', 'while', 'at', 'hasnt', 'into', 'll', 'with', 'to',
'ain', 'of', 'him', 'thatll', 'hasn', 'aren', 'they']
```

In [0]:

```
from nltk import pos_tag, word_tokenize
wnl = WordNetLemmatizer()
```

In [0]:

```
!python -m nltk.downloader punkt averaged_perceptron_tagger wordnet
```

```
/usr/lib/python3.6/runpy.py:125: RuntimeWarning: 'nltk.downloader' found in
sys.modules after import of package 'nltk', but prior to execution of 'nltk.
downloader'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

In [0]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
scores = data['Score'].values
for sent in data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    tokens = pos_tag(word_tokenize(sent))
    for w in tokens:
        for cleaned_words in cleanpunc(w[0]).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    #s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    # lemmatization works better with POS tagging
                    tag = w[1][0].lower()
                    tag = tag if tag in ['a', 'n', 'v'] else None
                    if not tag:
                        s = cleaned_words.lower().encode('utf8')
                    else:
                        s = wnl.lemmatize(cleaned_words.lower(), tag).lower().encode("utf8"
                    filtered_sentence.append(s)
                    if scores[i] == "positive":
                        all_positive_words.append(s) #list of all words used to describe po
                    if scores[i] == "negative":
                        all_negative_words.append(s) #list of all words used to describe ne
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
print("Done!")
```

Done!

In [0]:

```
data['CleanedText']=final_string #adding a column of CleanedText which displays the data af
data['CleanedText']=data['CleanedText'].str.decode("utf-8")
```

In [0]:

```
# store final table into an SQLLite table for future.
conn = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/
c=conn.cursor()
conn.text_factory = str
data.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, index_label=Nor
```

TSNE Assignment

Objective

Apply TSNE on Bow, TFIDF, W2V, TFIDF W2v

In [0]:

```
con = sqlite3.connect('/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/f
data = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
del data['index']
data.shape
```

Out[4]:

```
(363834, 11)
```

In [0]:

```
data.head()
```

Out[5]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [0]:

```
# positive reviews
pos_df = data[data['Score'] == 'positive'].copy()
pos_df['Time'] = pos_df['Time'].astype('int')
# sorting it based on time so that we can split based on time
pos_df.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na_posi
pos_df.shape
```

Out[6]:

```
(306764, 11)
```

In [0]:

```
#negative reviews
neg_df = data[data['Score'] == 'negative'].copy()
neg_df['Time'] = neg_df['Time'].astype('int')
neg_df.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na_posi
neg_df.shape
```

Out[7]:

```
(57070, 11)
```

In [0]:

```
pos_50k = pos_df.head(50000).copy()
neg_50k = neg_df.head(50000).copy()
```

In [0]:

```
# training data 60%
pos_train = pos_50k.head(30000).copy()
neg_train = neg_50k.head(30000).copy()

# cross validation data 20%
pos_cv = pos_50k[30000:40000].copy()
neg_cv = neg_50k[30000:40000].copy()

# test data 20%
pos_test = pos_50k[40000:].copy()
neg_test = neg_50k[40000:].copy()
```

In [0]:

```
train_df = pos_train.append(neg_train, ignore_index=True).copy()
cv_df = pos_cv.append(neg_cv, ignore_index=True).copy()
test_df = pos_test.append(neg_test, ignore_index=True).copy()
```

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
```

In [0]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
train_final_counts = count_vect.fit_transform(train_df['CleanedText'].values)
cv_final_counts = count_vect.transform(cv_df['CleanedText'].values)
test_final_counts = count_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_final_counts))
print("the shape of out text BOW vectorizer ",train_final_counts.get_shape())
print("the number of unique words ", train_final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 40286)
the number of unique words  40286
```

In [0]:

```
import nltk
```

In [0]:

```
freq_dist_positive=nltk.FreqDist(all_positive_words)
freq_dist_negative=nltk.FreqDist(all_negative_words)
print("Most Common Positive Words : ",freq_dist_positive.most_common(20))
print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive Words :  [(b'like', 137224), (b'taste', 122045), (b'goo
d', 111390), (b'love', 104938), (b'great', 103358), (b'use', 101467), (b'mak
e', 100401), (b'flavor', 99414), (b'one', 96800), (b'get', 93100), (b'produc
t', 90812), (b'try', 86221), (b'tea', 82860), (b'coffee', 78957), (b'find',
78423), (b'buy', 75962), (b'food', 64946), (b'would', 59996), (b'eat', 5743
8), (b'time', 54081)]
Most Common Negative Words :  [(b'taste', 33523), (b'like', 31734), (b'produ
ct', 28122), (b'buy', 20800), (b'one', 20593), (b'would', 20028), (b'get', 2
0000), (b'flavor', 18123), (b'try', 17575), (b'make', 16240), (b'use', 1491
5), (b'good', 14894), (b'coffee', 14764), (b'order', 12792), (b'food', 1275
6), (b'think', 11931), (b'tea', 11633), (b'eat', 11013), (b'even', 10947),
(b'box', 10812)]
```

In [0]:

```
import pickle
```

In [0]:

```
#saving BoW unigrams
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_t
    pickle.dump(train_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_lab.pkl
    pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_c
    pickle.dump(cv_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/cv_lab.pkl",
    pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_uni_vec_t
    pickle.dump(test_final_counts, bow)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/test_lab.pkl"
    pickle.dump(test_df['Score'].values, bow)
```

In [0]:

```
#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) ) #in scikit-learn
train_bigram_counts = count_vect.fit_transform(train_df['CleanedText'].values)
cv_bigram_counts = count_vect.transform(cv_df['CleanedText'].values)
test_bigram_counts = count_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_bigram_counts))
print("the shape of out text BOW vectorizer ",train_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", train_bigram_count
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 977013)
the number of unique words including both unigrams and bigrams  977013
```

In [0]:

```
#saving BoW bigrams
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_tr
    pickle.dump(train_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_cv
    pickle.dump(cv_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_te
    pickle.dump(test_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(test_df['Score'].values, bow)
```

In [0]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [0]:

```
#tf-idf
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
train_tf_idf = tf_idf_vect.fit_transform(train_df['CleanedText'].values)
cv_tfidf = tf_idf_vect.transform(cv_df['CleanedText'].values)
test_tfidf = tf_idf_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_tf_idf))
print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", train_tf_idf.get_s
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (60000, 977013)
the number of unique words including both unigrams and bigrams  977013
```

In [0]:

```
features = tf_idf_vect.get_feature_names()
print("some sample features(unique words in the corpus)",features[100000:100010])
```

```
some sample features(unique words in the corpus) ['bpa originally', 'bpa pac
kaging', 'bpa person', 'bpa personally', 'bpa plastic', 'bpa prove', 'bpa re
ally', 'bpa recently', 'bpa rest', 'bpa safe']
```

In [0]:

```
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature name
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(train_tf_idf[1,:].toarray()[0],features,25)
```

In [0]:

```
top_tfidf
```

Out[47]:

	feature	tfidf
0	paperback seem	0.182072
1	rosie movie	0.182072
2	incorporate love	0.182072
3	version paperback	0.182072
4	cover version	0.182072
5	page open	0.182072
6	keep page	0.182072
7	read sendak	0.182072
8	movie incorporate	0.182072
9	hard cover	0.175544
10	miss hard	0.175544
11	sendak book	0.175544
12	grow read	0.175544
13	kind flimsy	0.175544
14	really rosie	0.175544
15	watch really	0.175544
16	flimsy take	0.175544
17	however miss	0.175544
18	book watch	0.175544
19	two hand	0.175544
20	love son	0.167320
21	rosie	0.164385
22	paperback	0.164385
23	seem kind	0.161903
24	hand keep	0.157857

In [0]:

```
#saving tfidf
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_tra
    pickle.dump(train_tf_idf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_t
#     pickle.dump(train_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_cv.
    pickle.dump(cv_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_c
#     pickle.dump(cv_df['Score'].values, bow)

with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_tes
    pickle.dump(test_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_t
#     pickle.dump(test_df['Score'].values, bow)
```

In [0]:

```
from gensim.models import Word2Vec
```

In [0]:

```
list_of_sent=[]
for sent in train_df['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [0]:

```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=7)
```

In [0]:

```
#saving w2v model
w2v_model.save("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/amzn_w2v
```

In [0]:

```
#loading model
w2v_model = Word2Vec.load("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_revie
```

In [0]:

```
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  12979
sample words  ['witty', 'little', 'book', 'make', 'son', 'laugh', 'loud', 'r
ecite', 'car', 'drive', 'along', 'always', 'sing', 'refrain', 'learn', 'whal
e', 'india', 'droop', 'rose', 'love', 'new', 'word', 'classic', 'willing',
'bet', 'still', 'able', 'memory', 'college', 'grow', 'read', 'sendak', 'watc
h', 'really', 'rosie', 'movie', 'incorporate', 'however', 'miss', 'hard', 'c
over', 'version', 'paperback', 'seem', 'kind', 'flimsy', 'take', 'two', 'han
d', 'keep']
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2vec(list_of_sent):
    sent_vectors = [] # the avg-w2v for each sentence/review is stored in this list
    for sent in list_of_sent: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        cnt_words =0 # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
    return sent_vectors
```

In [0]:

```
avg_w2v_train = avg_w2vec([sent.split() for sent in train_df['CleanedText'].values])
avg_w2v_cv = avg_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
avg_w2v_test = avg_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

```
60000
50
20000
50
20000
50
```

In [0]:

```
#saving word2vec
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_train
    pickle.dump(avg_w2v_train, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_cv.pk
    pickle.dump(avg_w2v_cv, w2v_pickle)
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/avg_w2v_test.
    pickle.dump(avg_w2v_test, w2v_pickle)
```

In [0]:

```
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor
from concurrent import futures
```

In [0]:

```
!pip install numba
```

```
Requirement already satisfied: numba in /usr/local/lib/python3.6/dist-packag
es (0.40.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packag
es (from numba) (1.14.6)
Requirement already satisfied: llvmlite>=0.25.0dev0 in /usr/local/lib/python
3.6/dist-packages (from numba) (0.26.0)
```

In [0]:

```
from numba import jit
```

In [0]:

```
def helper(list_of_sent, final_tf_idf):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this lis
    row=0;
    for sent in list_of_sent: # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                # obtain the tf_idfidf of a word in a sentence/review
                tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
                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
        print(row, end=" ")
    return tfidf_sent_vectors
```

In [0]:

```
helper_numba = jit()(helper)
```

In [0]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

#tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
#row=0;
#for sent in list_of_sent: # for each review/sentence
#this was taking a lot of time

# with ThreadPoolExecutor(max_workers=10000) as executor:
#     result_futures = [executor.submit(helper_numba, sent=x, row=y) for y, x in enumerate(
#     for f in futures.as_completed(result_futures):
#         i = f.result()
#         print(i)
# print("Threading done!")

# for y, x in enumerate(list_of_sent):
#     i = helper_numba(sent=x, row=y)
#     print(i)
list_of_sent = [sent.split() for sent in train_df['CleanedText'].values]
train_tfidf_w2v = helper(list_of_sent, train_tf_idf)

print("Done!")
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
9

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-67-e5652b938fbd> in <module>()
     18 #     print(i)
     19 list_of_sent = [sent.split() for sent in train_df['CleanedText'].val
ues]
---> 20 train_tfidf_w2v = helper(list_of_sent, train_tf_idf)
     21 
     22 print("Done!")

<ipython-input-64-410d0381b5e3> in helper(list_of_sent, final_tf_idf)
      9                 vec = w2v_model.wv[word]
     10                 # obtain the tf_idfidf of a word in a sentence/revie
w
---> 11                 tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
     12                 sent_vec += (vec * tf_idf)
     13                 weight_sum += tf_idf

KeyboardInterrupt: 
```

In [0]:

```
# TF-IDF weighted Word2Vec was taking a lot of time!! So for TSNE, will run it on 2k data p
#saving tfidf weighted w2v
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_weighte
    pickle.dump(train_tfidf_w2v, tfidf_w2v_pickle)

print("Done!")
```

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import pandas as pd

from sklearn.manifold import TSNE
```

In [0]:

```
#loading sparse matrices of BoW
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_tr
    bigram_counts = pickle.load(bow)
#loading labels
with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/train_lab.pkl
    labels = pickle.load(bow)
bigram_counts.shape[0]
```

Out[3]:

```
60000
```

In [0]:

```
from scipy.sparse import vstack
```

In [0]:

```
bigram_counts.shape
```

Out[5]:

```
(60000, 977013)
```

In [0]:

```
final = bigram_counts[0:1000,:]
final.shape
```

Out[6]:

```
(1000, 977013)
```

In [0]:

```
#picking 1k positive and 1k negative review vectors
final = bigram_counts[0:1000,:]
final = vstack((final, bigram_counts[30000:31000,:]))
final_labels = labels[:1000].tolist()
final_labels.extend(labels[30000:31000].tolist())
print(final.shape, len(final_labels))
```

```
(2000, 977013) 2000
```

BoW

In [0]:

```
# tried with above dimensions - BoW unique word dict is based on 60k reviews, was getting '
# hence will create BOW using 2000 reviews

pos_1k = pos_df.sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False,
neg_1k = neg_df.sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False,
pos_1k.head(5)
```

In [0]:

```
reviews = pos_1k.append(neg_1k, ignore_index=True)
```

In [0]:

```
reviews.shape
```

Out[9]:

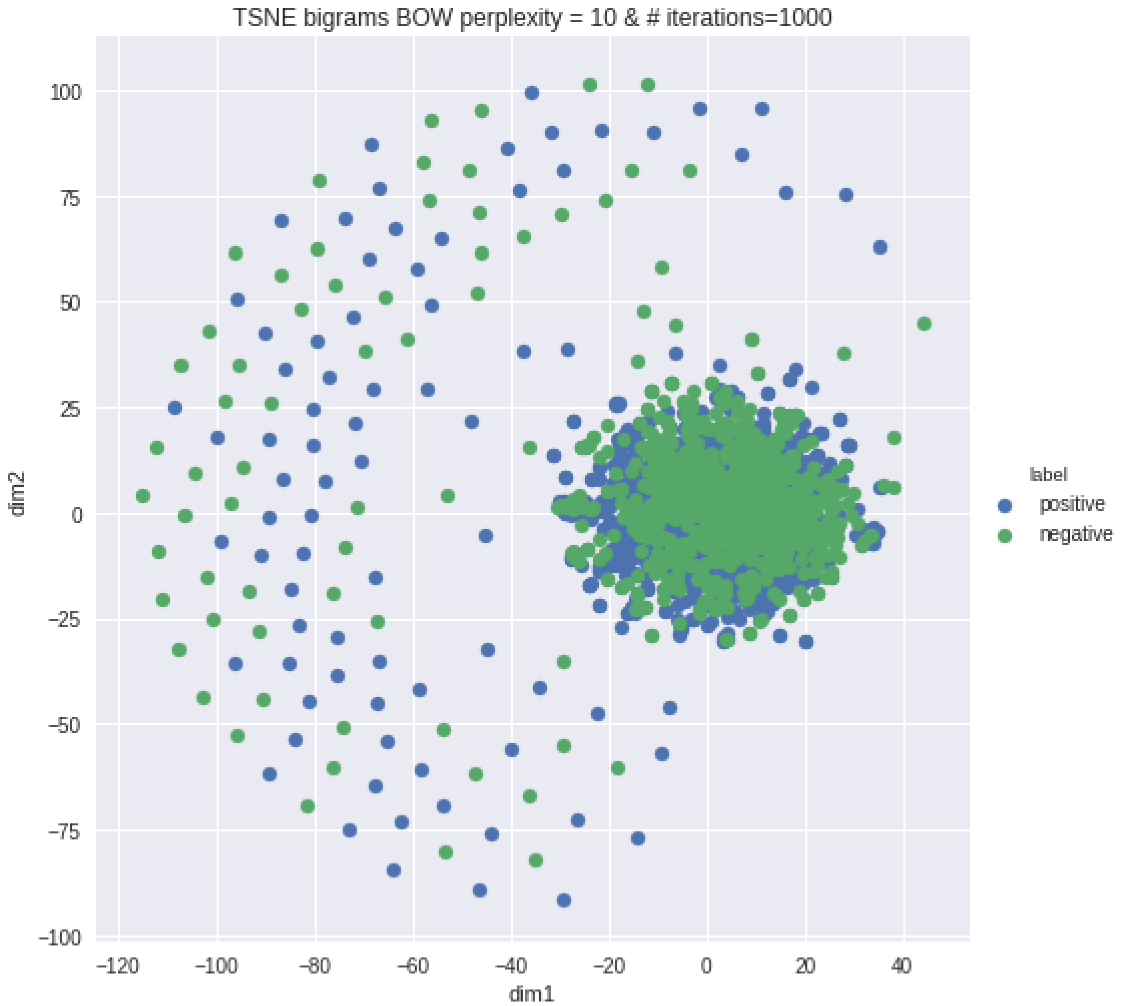
```
(2000, 11)
```

In [0]:

```
count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
final = count_vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

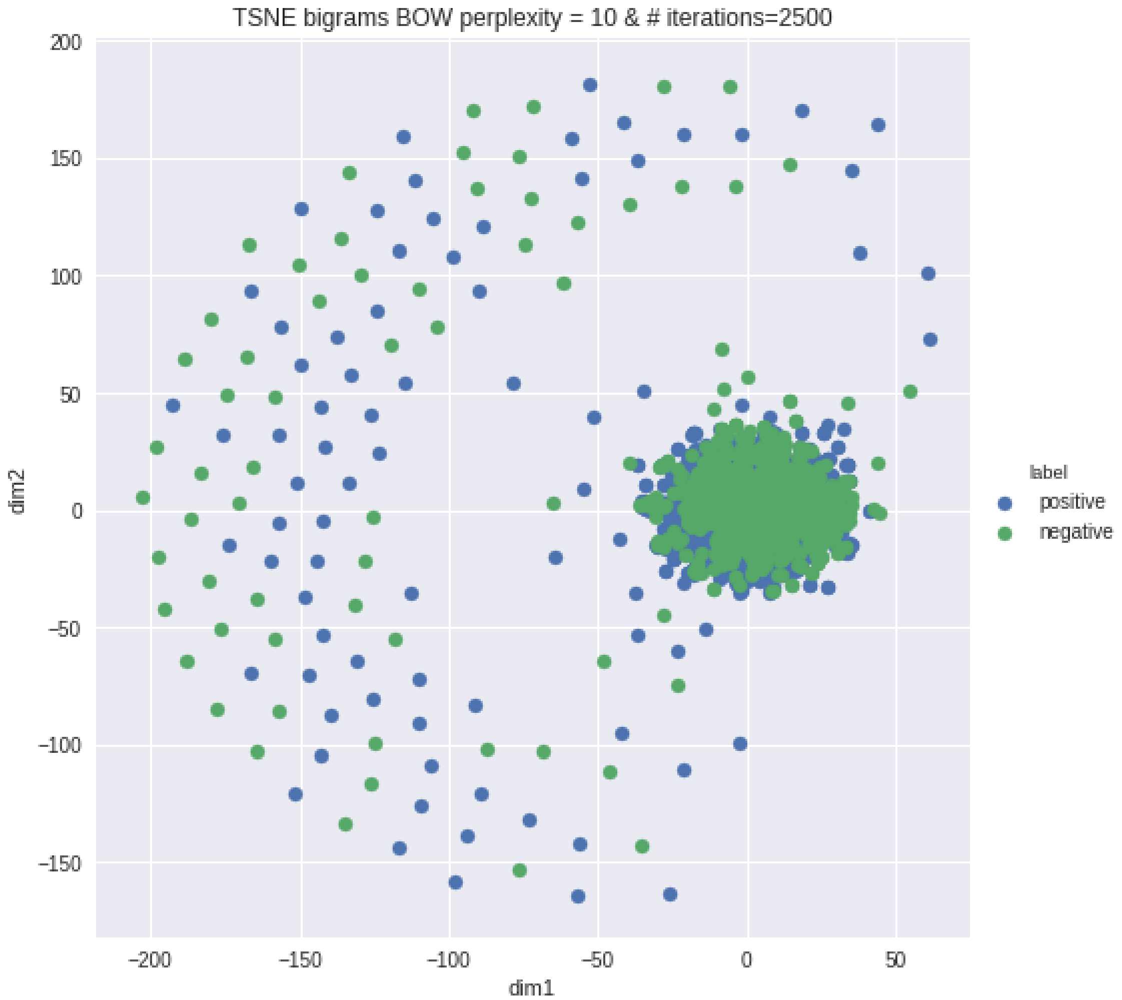
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=10 and #iterations=1000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=1000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 10 & # iterations=1000')
plt.show()
```



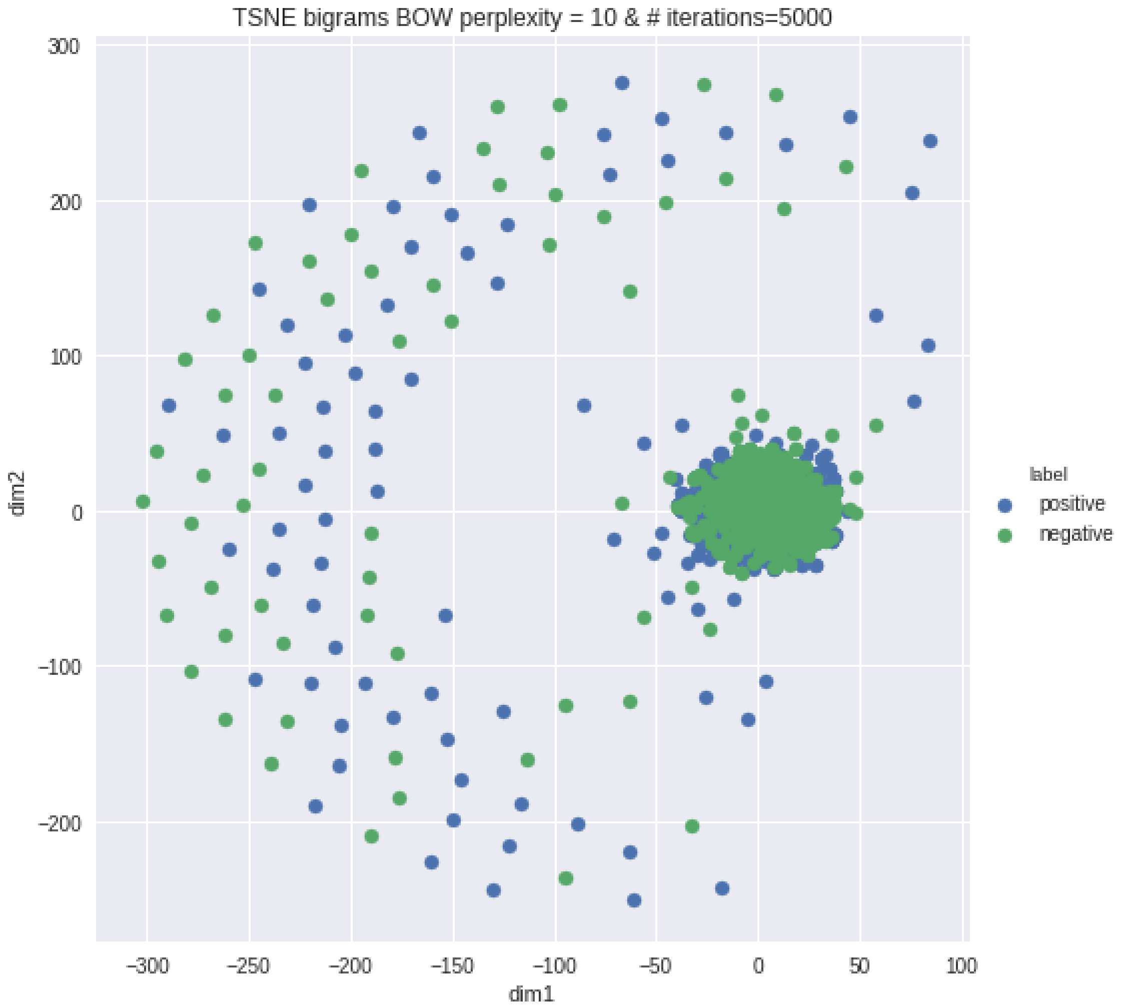
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 10 & # iterations=2500')
plt.show()
```



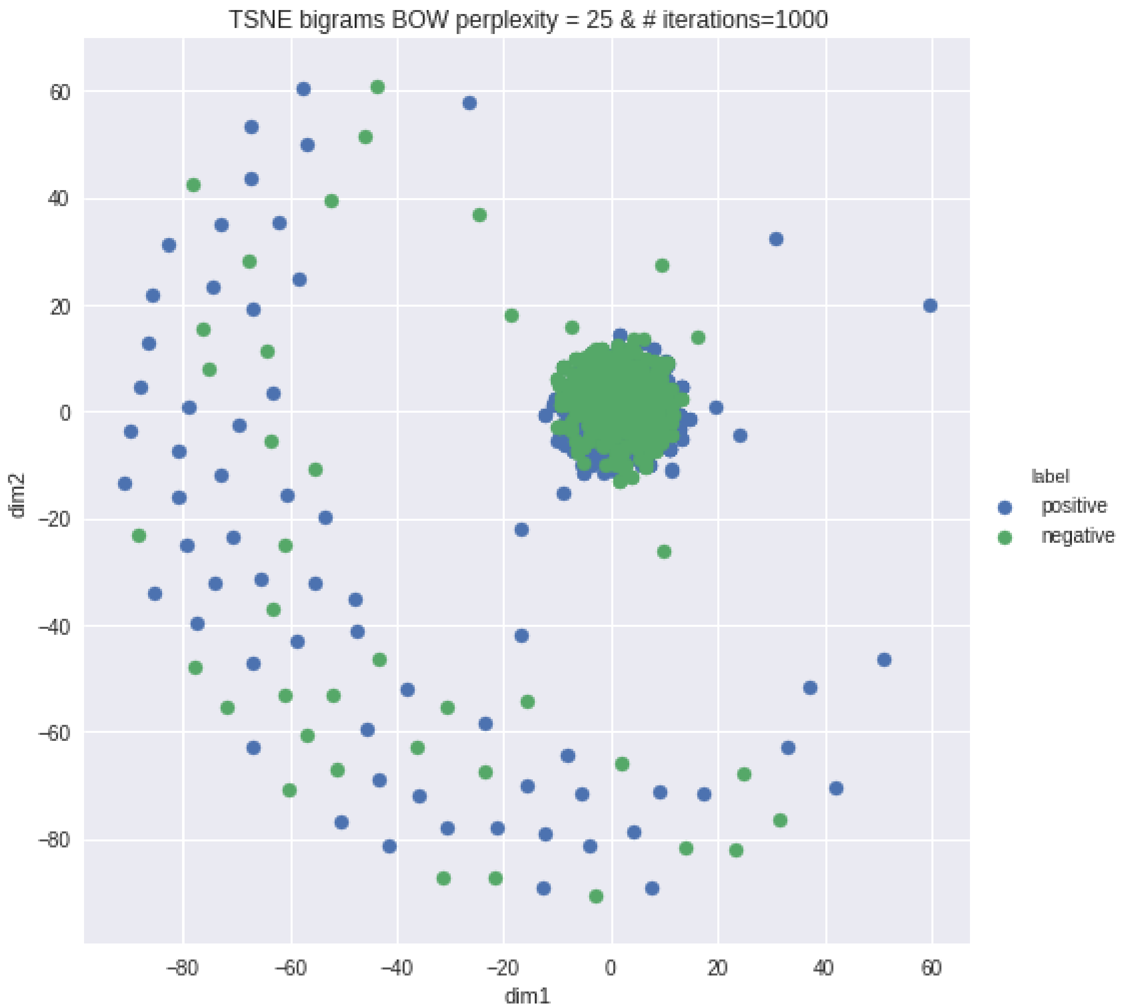
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=10 and #iterations=5000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 10 & # iterations=5000')
plt.show()
```



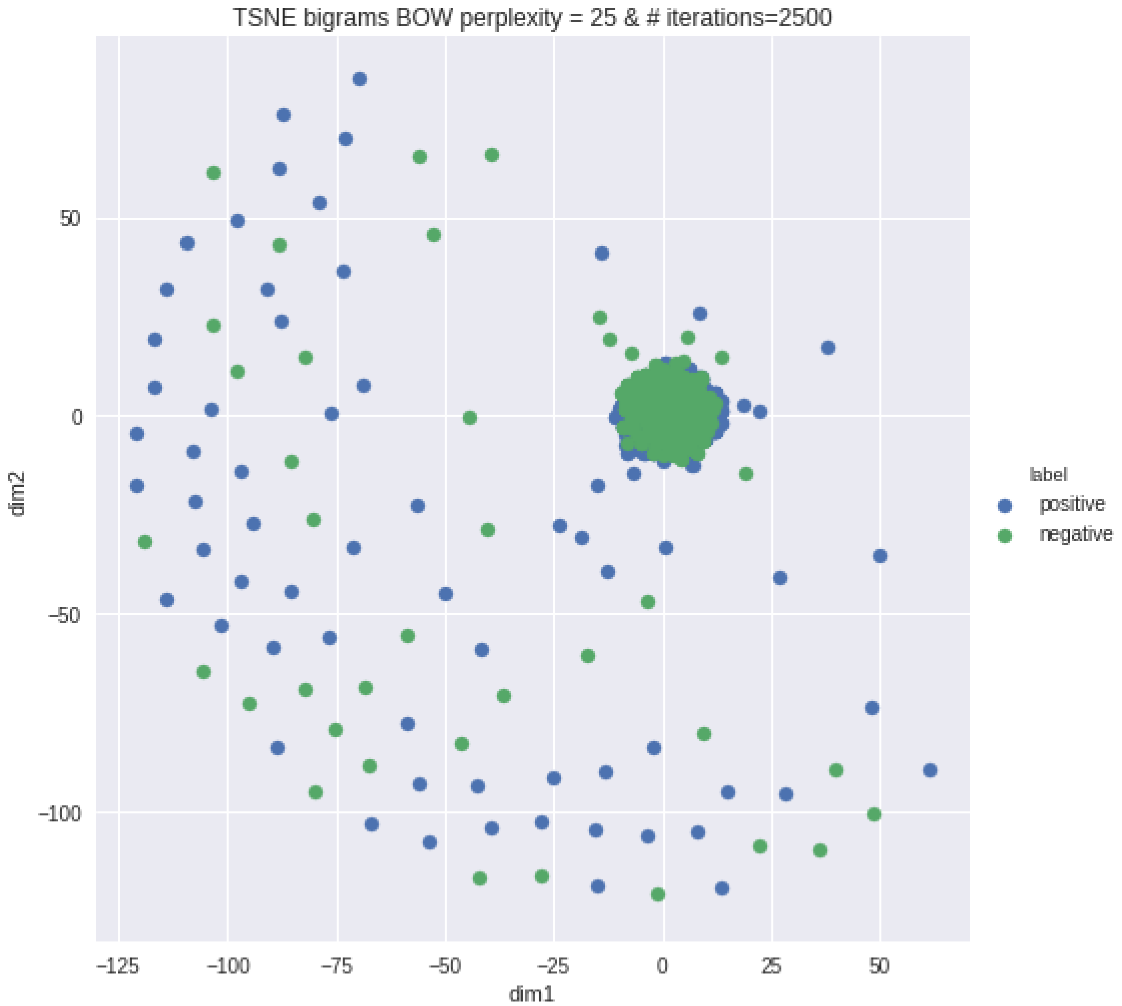
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=25 and #iterations=1000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=1000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 25 & # iterations=1000')
plt.show()
```



In [0]:

```
#TSNE for bi-grams BOW
#perplexity=25 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 25 & # iterations=2500')
plt.show()
```

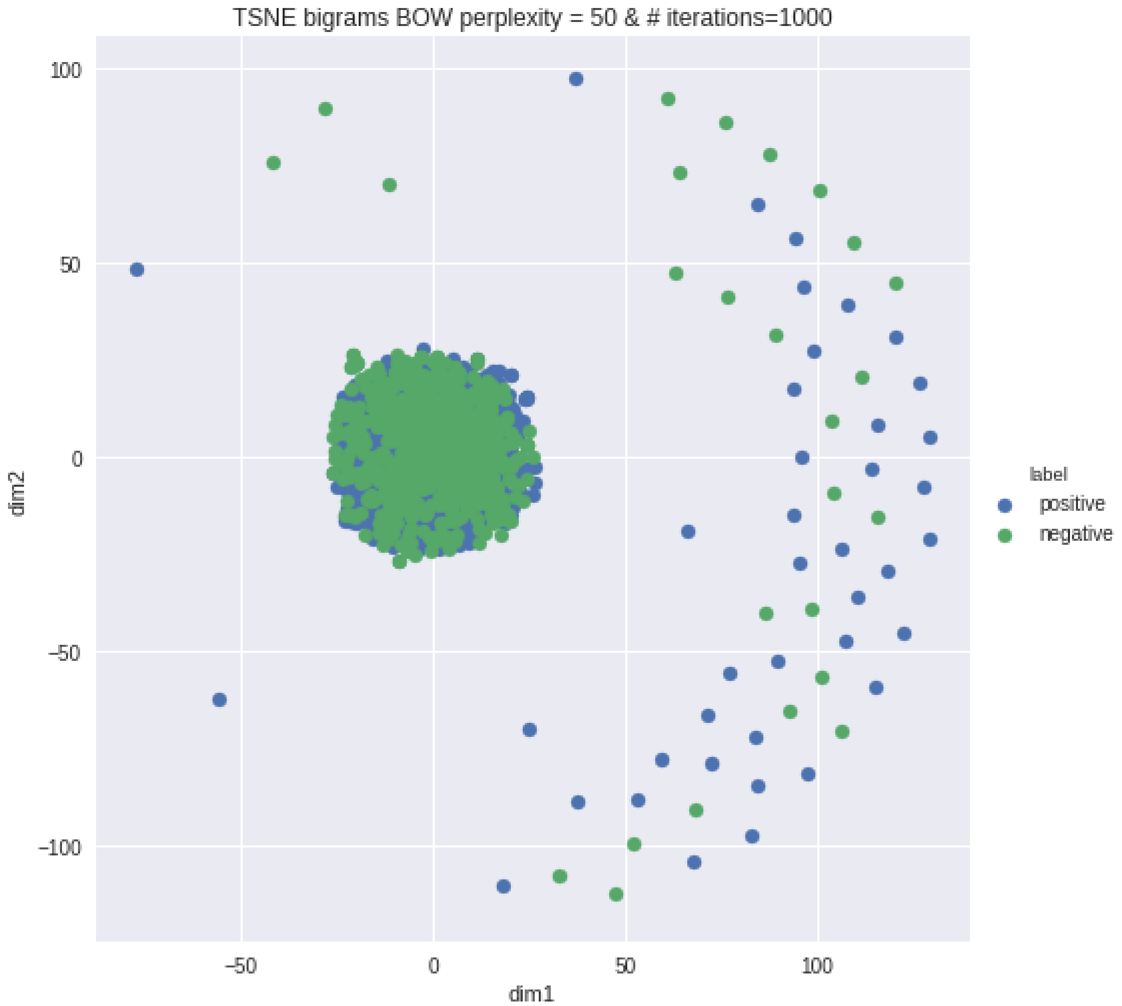


In [0]:

```
#TSNE for bi-grams BOW
#perplexity=25 and #iterations=5000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 25 & # iterations=5000')
plt.show()
```

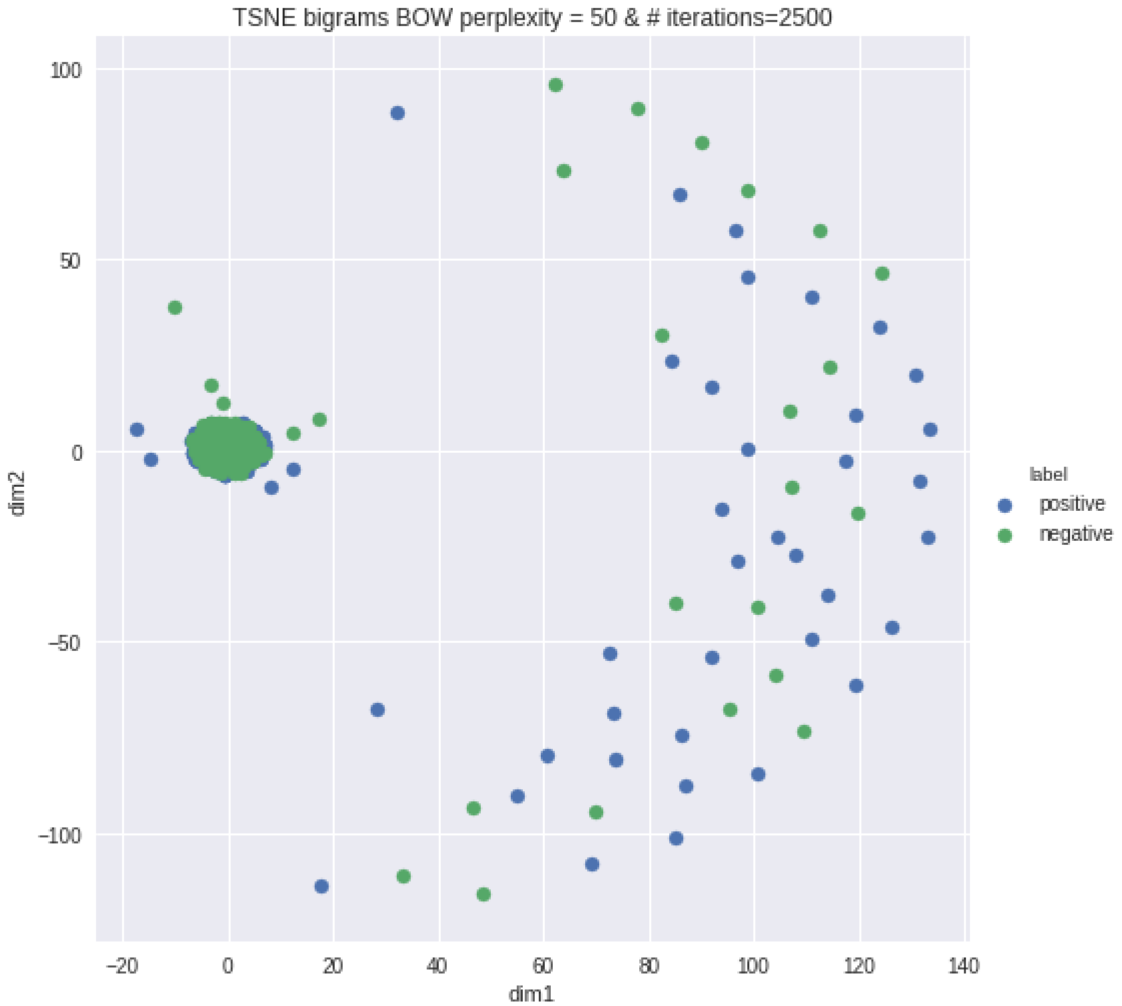
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=50 and #iterations=1000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=1000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 50 & # iterations=1000')
plt.show()
```



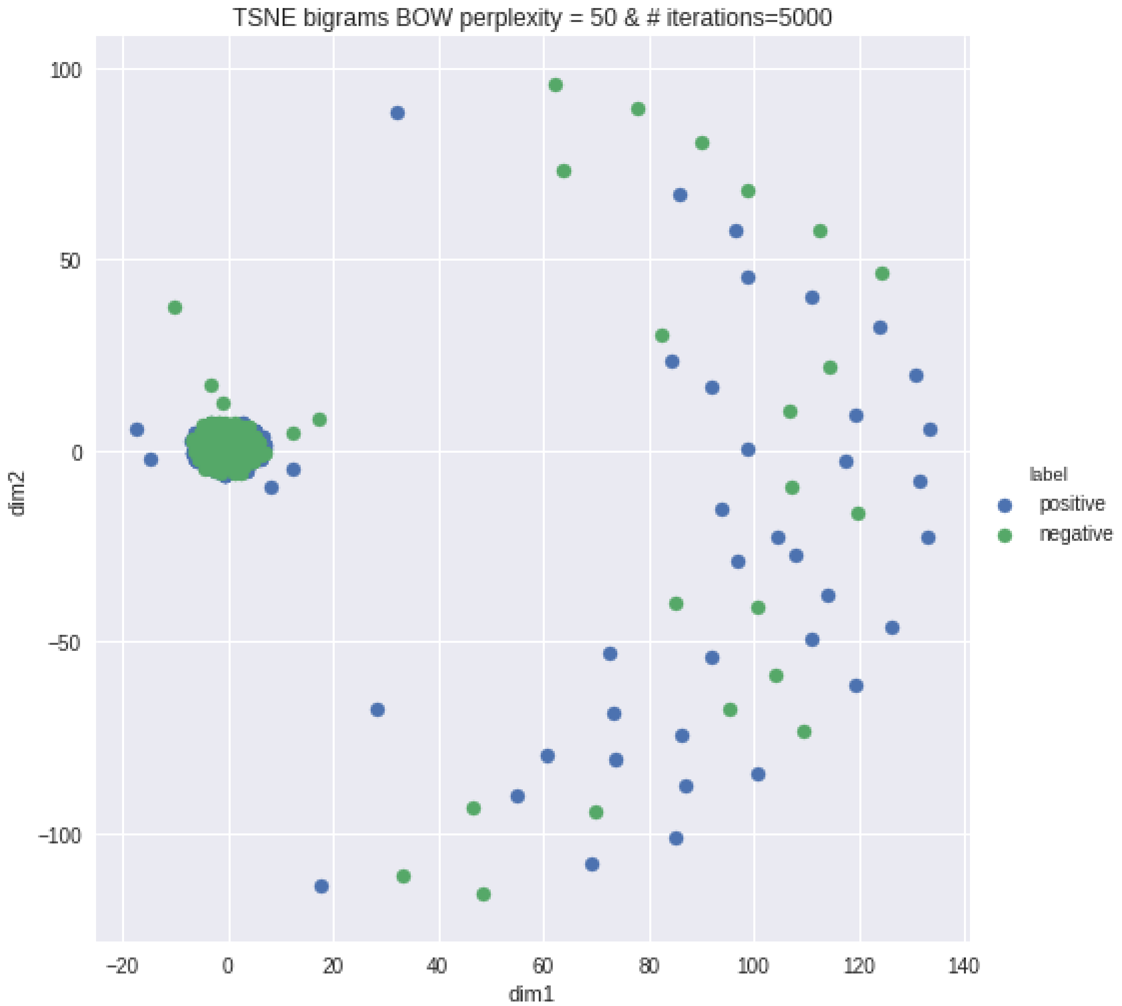
In [0]:

```
#TSNE for bi-grams BOW
#perplexity=50 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 50 & # iterations=2500')
plt.show()
```



In [0]:

```
#TSNE for bi-grams BOW
#perplexity=50 and #iterations=5000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 50 & # iterations=5000')
plt.show()
```



In [0]:

```
pos_1k = pos_df. \
    sort_values('Time', axis=0, ascending=False, inplace=False, kind='quicksort', na_positi
neg_1k = neg_df. \
    sort_values('Time', axis=0, ascending=False, inplace=False, kind='quicksort', na_positi
pos_1k.head(5)
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulı
261071	227839	B0033HPPIO	A2U3RRCA2URS56	ashley	0	
117303	300676	B000RQMQAO	A3KFOZ5D1KQLIU	Phyllis Brown	0	
343811	194172	B005UGSR72	A382RNG86OC7N0	Jennifer	0	
304007	276219	B004AA2MUM	A1YZ4FP8H9AV6I	Marburg	0	
362720	320388	B008JA73RG	AFJFXN42RZ3G2	R. DelParto "Rose2"	0	

In [0]:

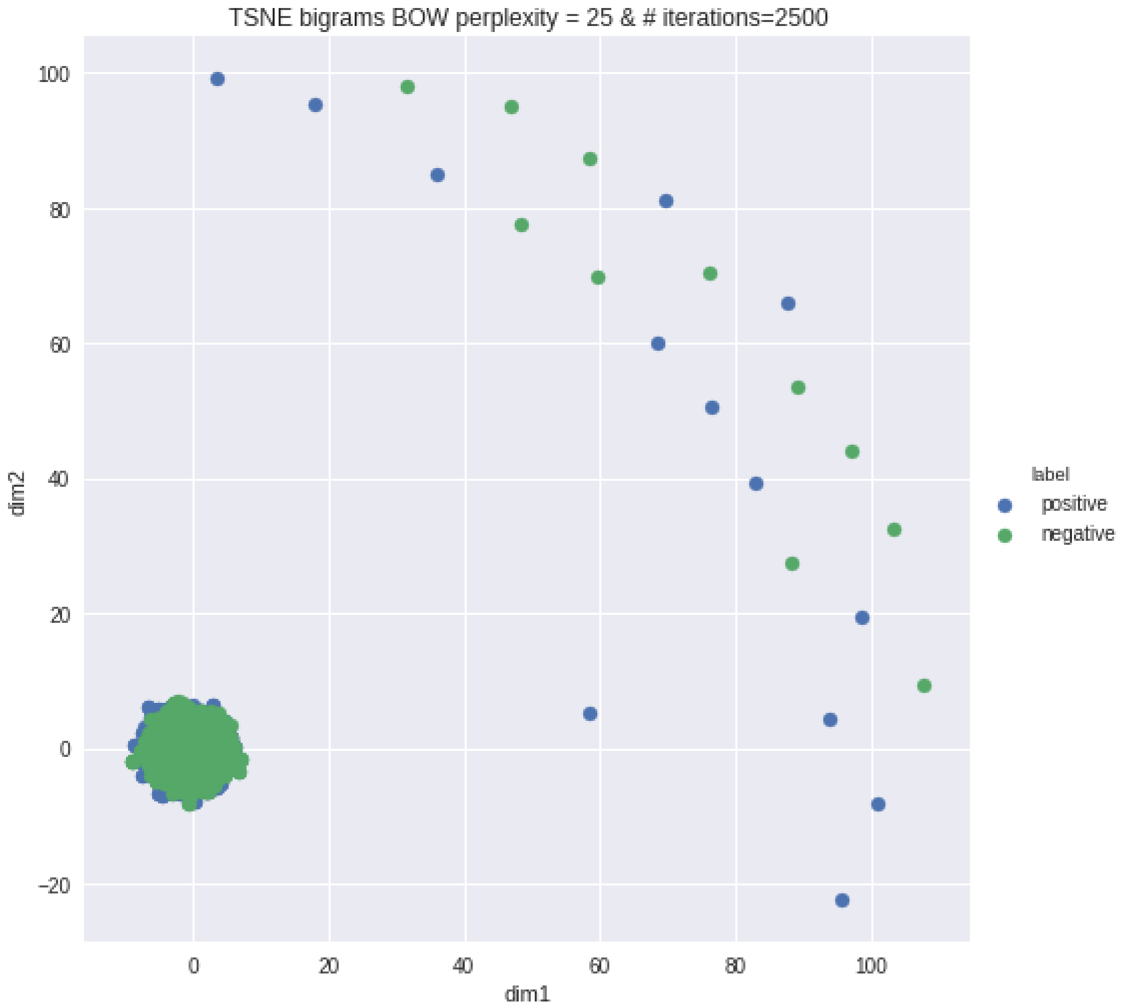
```
reviews = pos_1k.append(neg_1k, ignore_index=True)
```

In [0]:

```
count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
final = count_vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

In [0]:

```
#TSNE for bi-grams BOW
#perplexity=25 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 25 & # iterations=2500')
plt.show()
```



In [0]:

```
pos_1k = pos_df.sample(n=1000).copy()
neg_1k = neg_df.sample(n=1000).copy()
reviews = pos_1k.append(neg_1k, ignore_index=True)
```

In [0]:

```
count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
final = count_vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

In [0]:

```
#TSNE for bi-grams BOW
#perplexity=25 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE bigrams BOW perplexity = 25 & # iterations=2500')
plt.show()
```

I guess no change in TSNE however data you feed (Sorted based on Helpfulness/Time, Random). Why?

TFIDF

In [0]:

```
# TSNE for TFIDF
pos_1k = pos_df. \
    sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False, kind='quick
        head(1000).copy()
neg_1k = neg_df. \
    sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False, kind='quick
        head(1000).copy()
reviews = pos_1k.append(neg_1k, ignore_index=True)
reviews.shape
```

Out[23]:

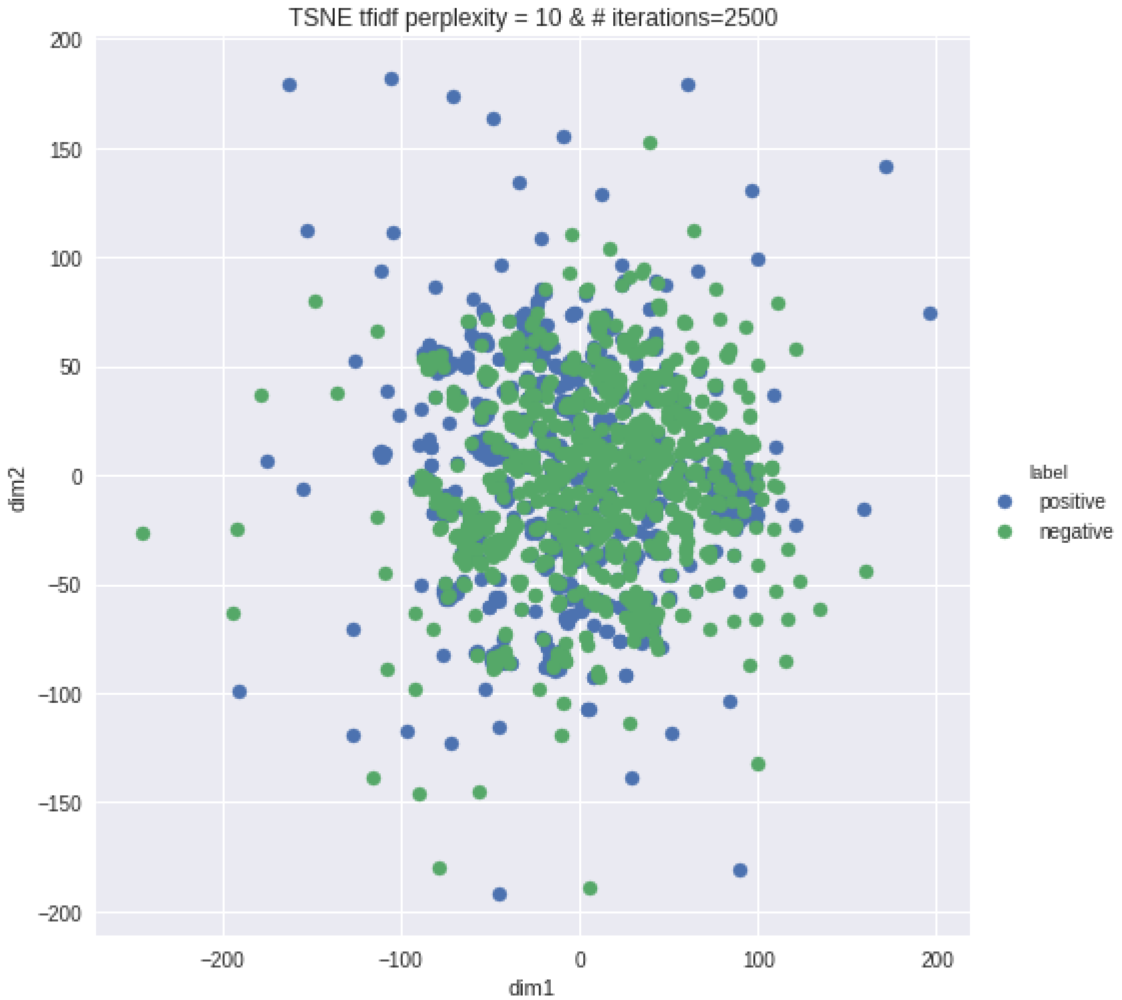
```
(2000, 11)
```

In [0]:

```
vect = TfidfVectorizer(ngram_range=(1,2))
final = vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

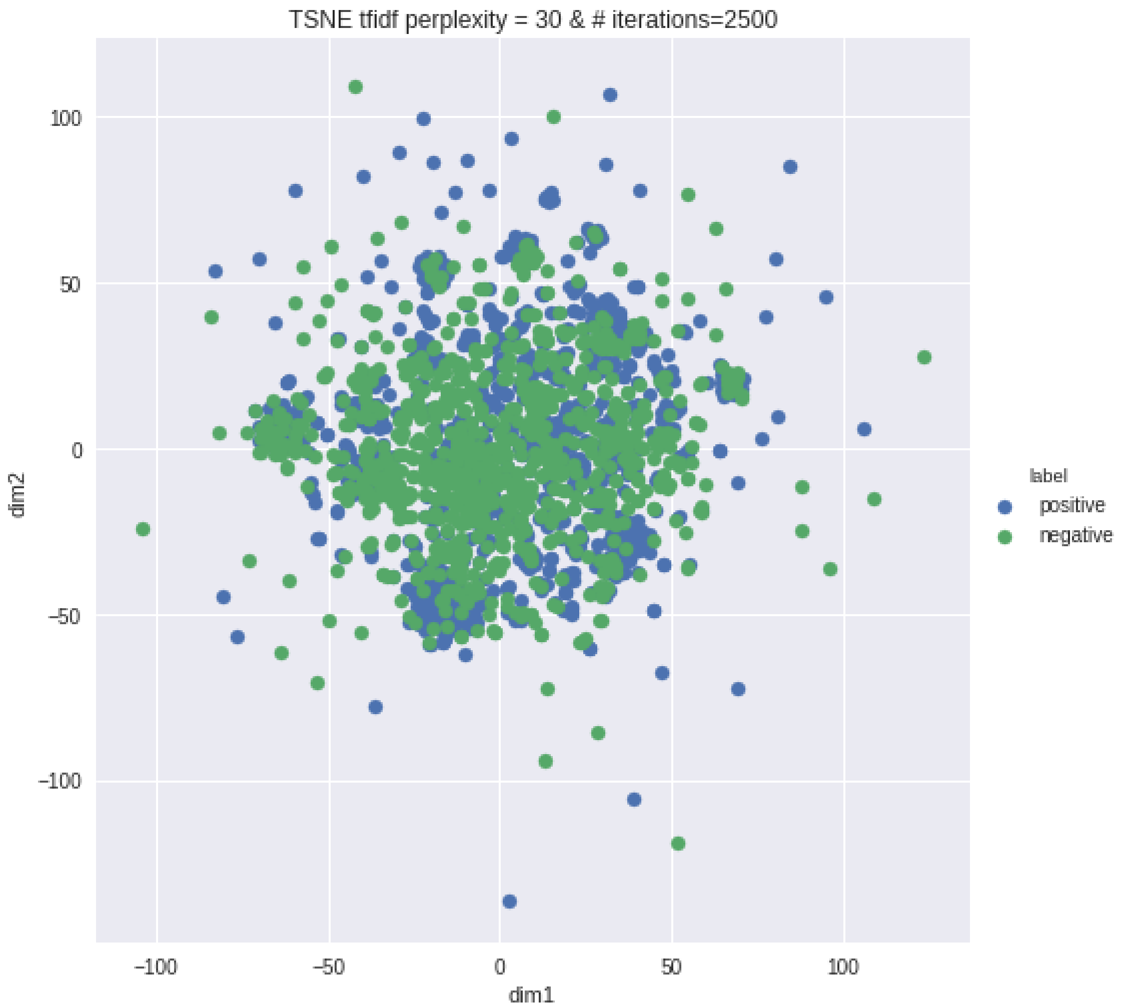
In [0]:

```
#TSNE for tfidf
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf perplexity = 10 & # iterations=2500')
plt.show()
```



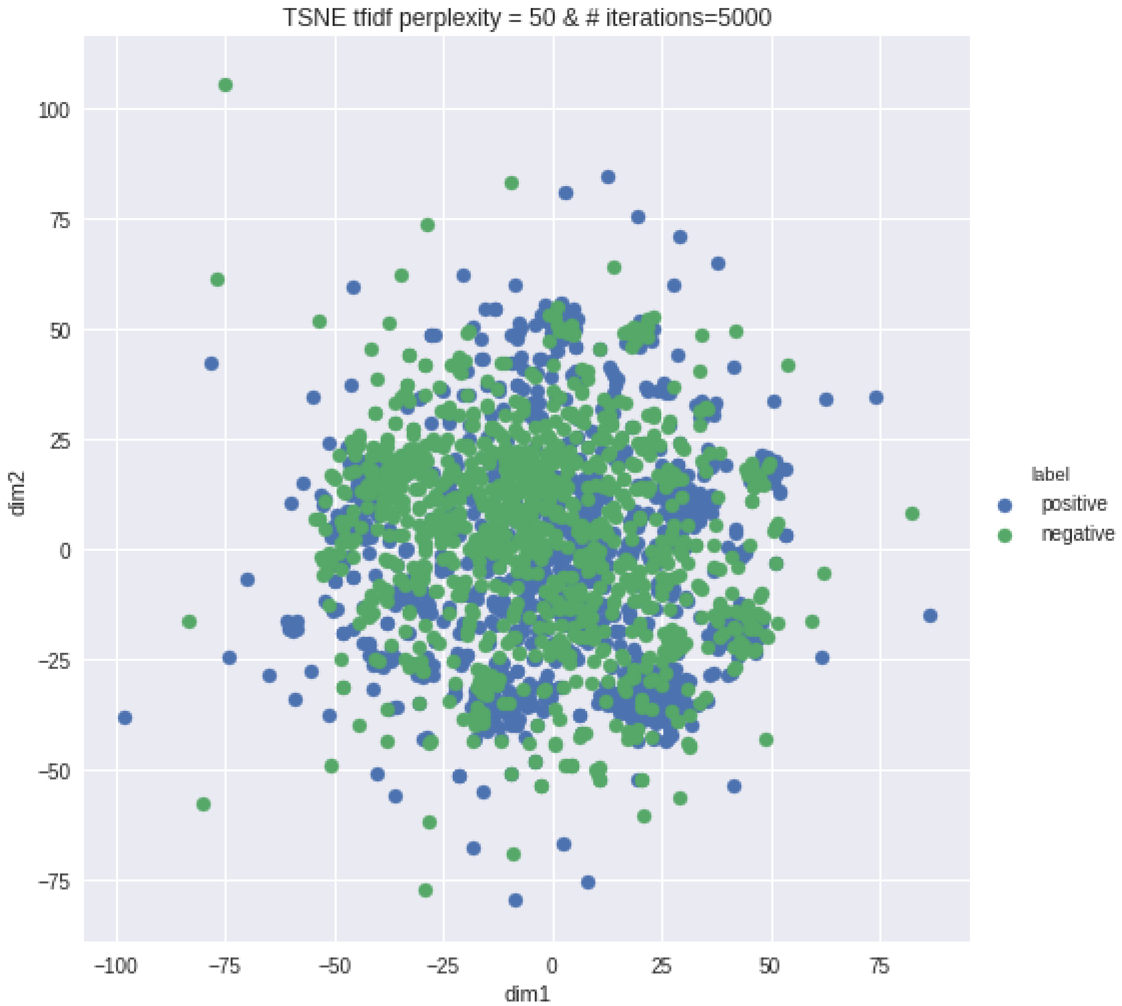
In [0]:

```
#TSNE for tfidf
#perplexity=30 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=30, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf perplexity = 30 & # iterations=2500')
plt.show()
```



In [0]:

```
#TSNE for tfidf
#perplexity=50 and #iterations=5000
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf perplexity = 50 & # iterations=5000')
plt.show()
```



In [0]:

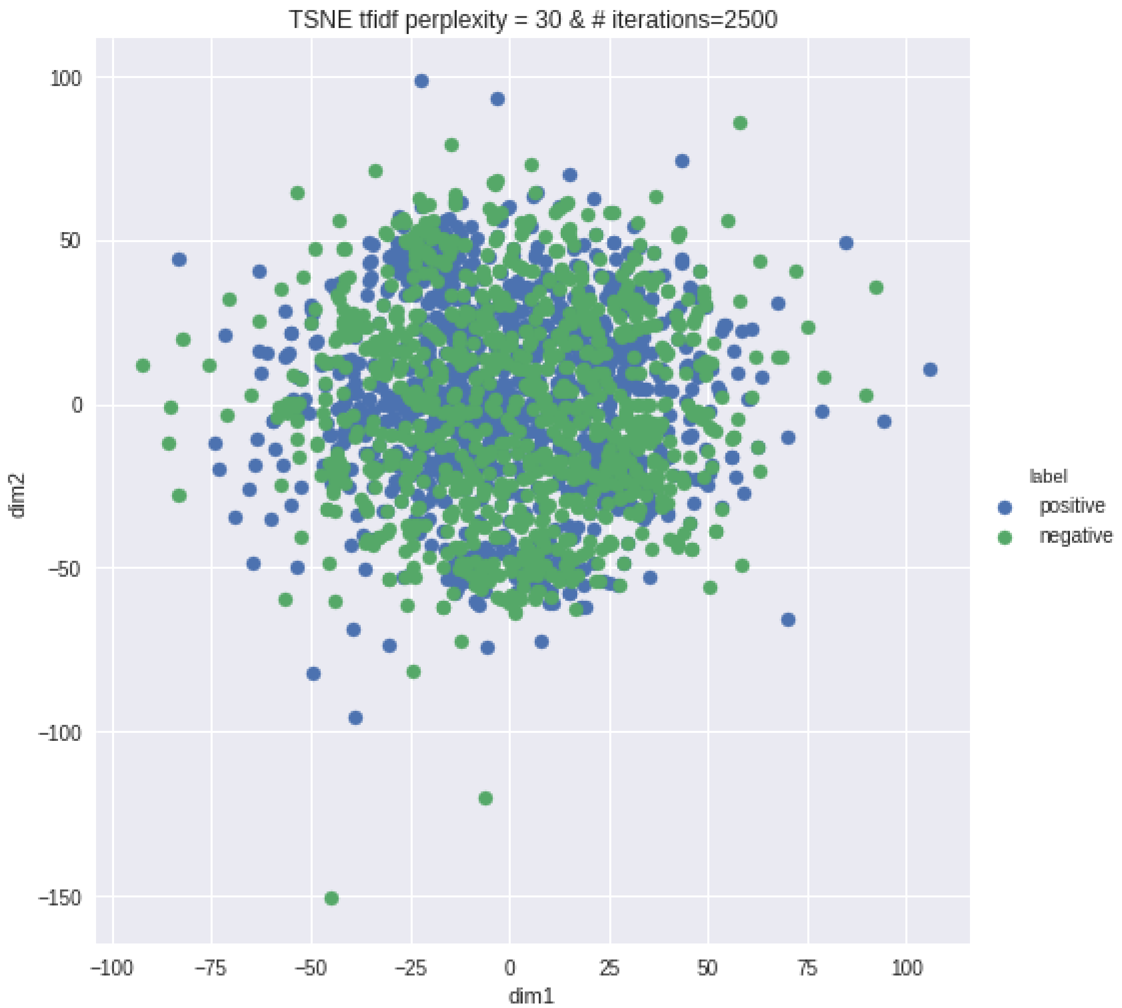
```
pos_1k = pos_df.sample(n=1000).copy()
neg_1k = neg_df.sample(n=1000).copy()
reviews = pos_1k.append(neg_1k, ignore_index=True)
```

In [0]:

```
vect = TfidfVectorizer(ngram_range=(1,2))
final = vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

In [0]:

```
#TSNE for tfidf
#perplexity=30 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=30, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final.todense())
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf perplexity = 30 & # iterations=2500')
plt.show()
```



W2V

In [0]:

```
# TSNE for W2V
pos_1k = pos_df. \
    sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False, kind='quick
        head(1000).copy()
neg_1k = neg_df. \
    sort_values('HelpfulnessNumerator', axis=0, ascending=False, inplace=False, kind='quick
        head(1000).copy()
reviews = pos_1k.append(neg_1k, ignore_index=True)
reviews.shape
```

Out[32]:

```
(2000, 11)
```

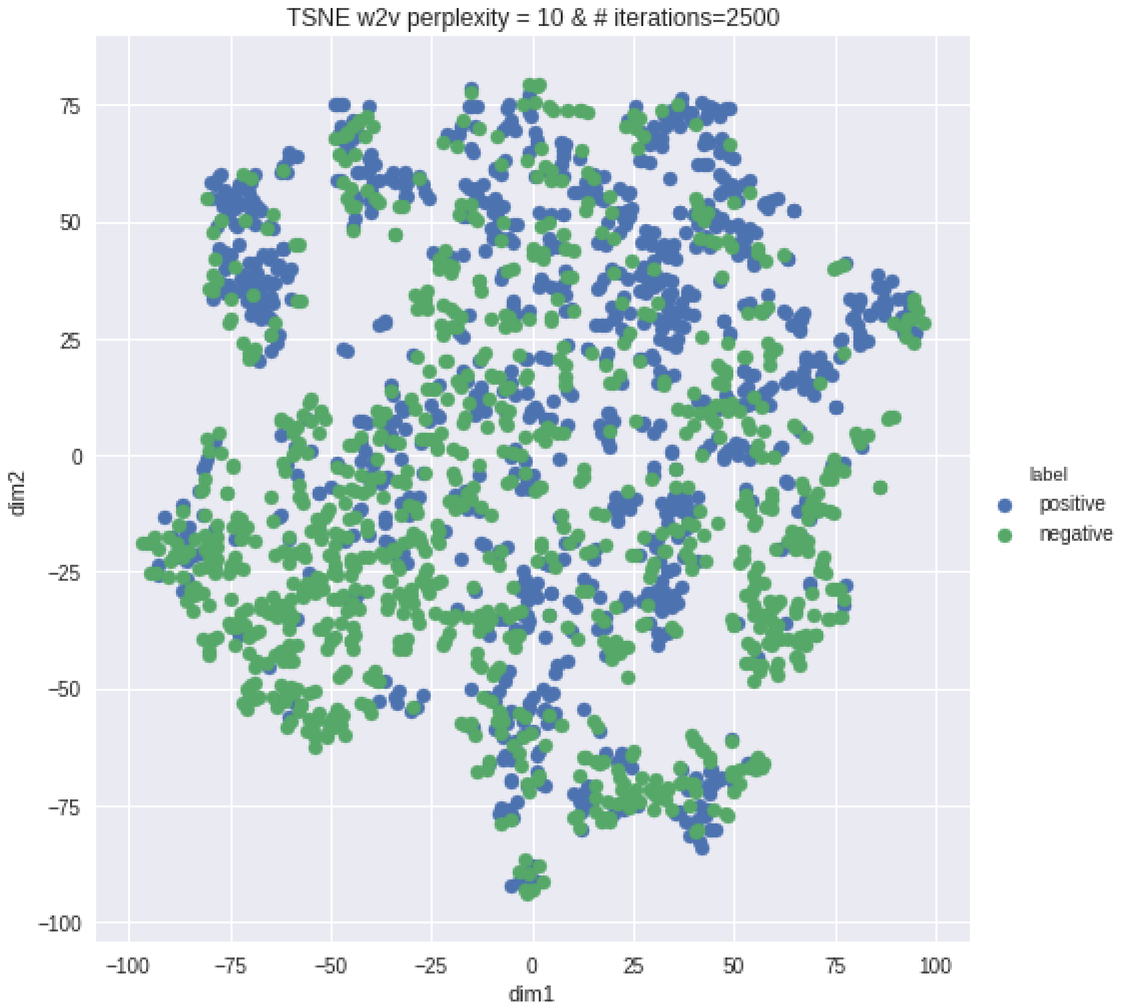
In [0]:

```
final = avg_w2vec([sent.split() for sent in reviews['CleanedText'].values])
final_labels = reviews['Score'].values
```

```
2000
50
```

In [0]:

```
#TSNE for W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE w2v perplexity = 10 & # iterations=2500')
plt.show()
```

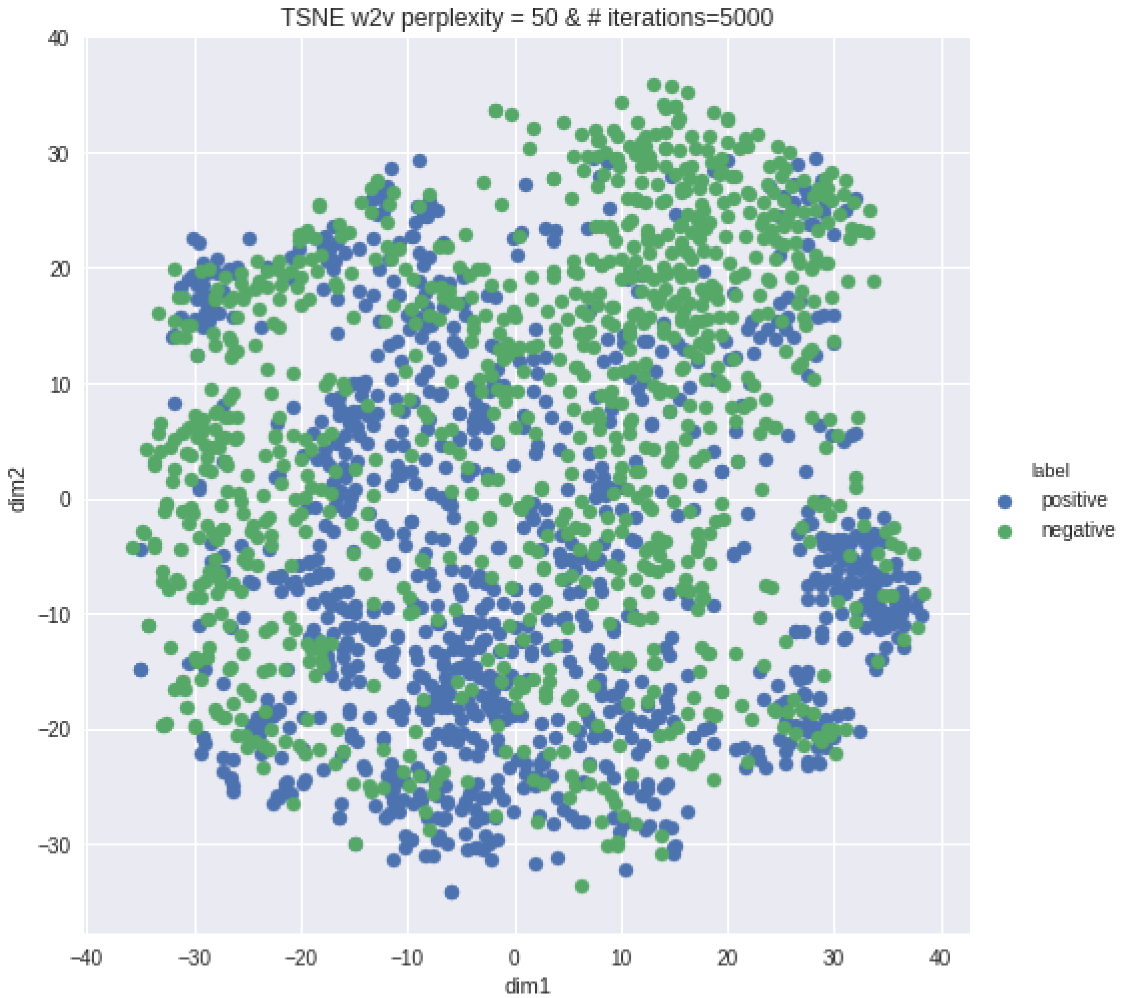


In [0]:

```
#TSNE for W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=30, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE w2v perplexity = 30 & # iterations=2500')
plt.show()
```

In [0]:

```
#TSNE for W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE w2v perplexity = 50 & # iterations=5000')
plt.show()
```



In [0]:

```
#TSNE for W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=10000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE w2v perplexity = 50 & # iterations=10000')
plt.show()
```

TFIDF W2V

In [0]:

```
vect = TfidfVectorizer(ngram_range=(1,2))
final_tfidf = vect.fit_transform(reviews['CleanedText'].values)
final_labels = reviews['Score'].values
```

In [0]:

```
tfidf_feat = vect.get_feature_names()
# def top_tfidf_feats(row, features, top_n=25):
#     ''' Get top n tfidf values in row and return them with their corresponding feature na
#     topn_ids = np.argsort(row)[::-1][:top_n]
#     top_feats = [(features[i], row[i]) for i in topn_ids]
#     df = pd.DataFrame(top_feats)
#     df.columns = ['feature', 'tfidf']
#     return df

# top_tfidf = top_tfidf_feats(final_tfidf[1,:].toarray()[0],features,25)
```

In [0]:

```
list_of_sent = [sent.split() for sent in reviews['CleanedText'].values]
final = helper(list_of_sent, final_tfidf)
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
9 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 8
0 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 1
04 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 1
23 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 1
42 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 1
61 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 1
80 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 1
99 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 2
18 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 2
37 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 2
56 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 2
75 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 2
94 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 3
13 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 3
32 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 3
51 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 3
70 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 3
89 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 4
08 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 4
27 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 4
46 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 4
65 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 4
84 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 5
03 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 5
22 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 5
41 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 5
60 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 5
79 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 5
98 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 6
17 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 6
36 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 6
55 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 6
74 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 6
93 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 7
12 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 7
31 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 7
50 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 7
69 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 7
88 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 8
07 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 8
26 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 8
45 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 8
64 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 8
83 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 9
02 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 9
21 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 9
40 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 9
59 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 9
78 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 9
97 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012
1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1
028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 10
43 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 105
8 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073
```

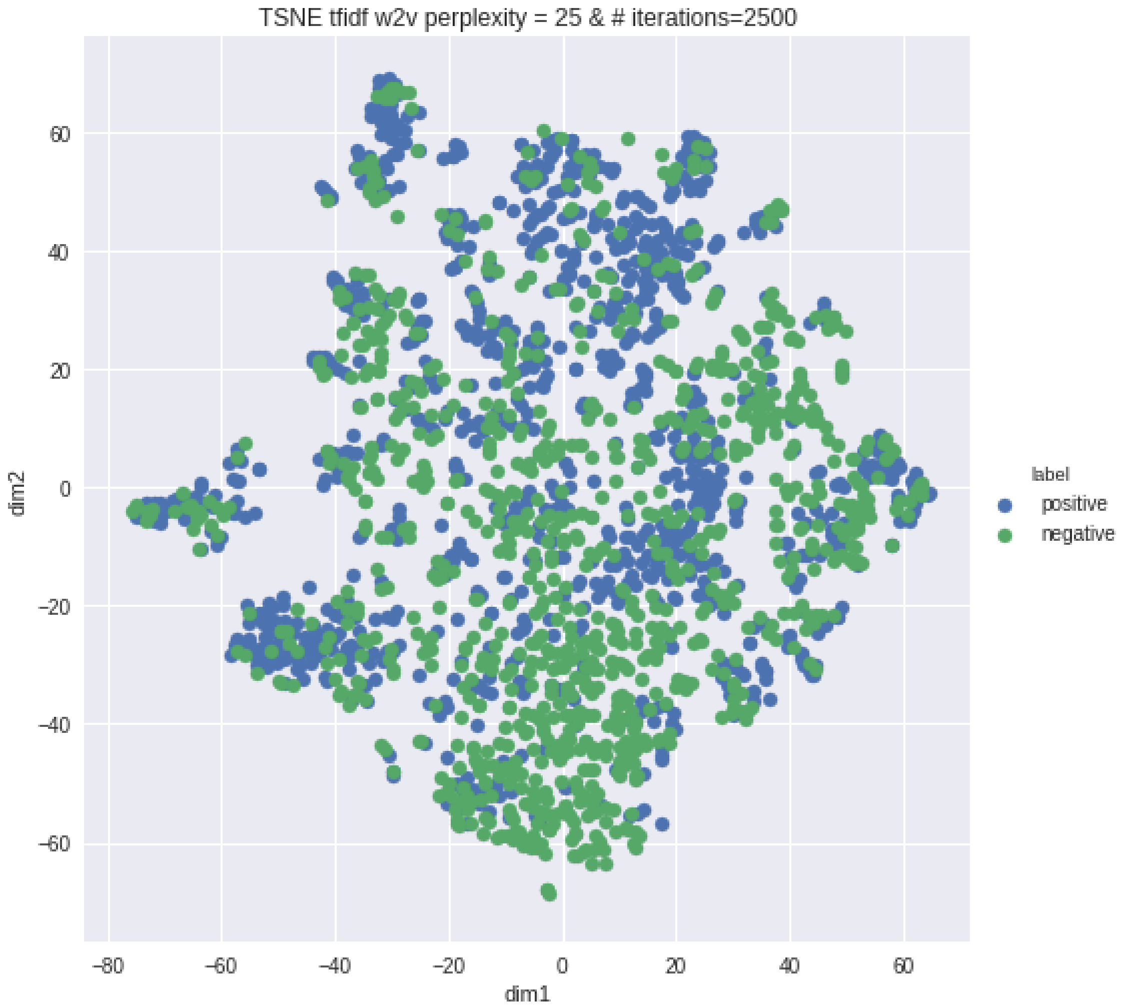
```
1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1
089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 11
04 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 111
9 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134
1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1
150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 11
65 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 118
0 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195
1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1
211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 12
26 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 124
1 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256
1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1
272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 12
87 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 130
2 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317
1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1
333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 13
48 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 136
3 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378
1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1
394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 14
09 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 142
4 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439
1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1
455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 14
70 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 148
5 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500
1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1
516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 15
31 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 154
6 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561
1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1
577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 15
92 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 160
7 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622
1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1
638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 16
53 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 166
8 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683
1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1
699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 17
14 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 172
9 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744
1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1
760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 17
75 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 179
0 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805
1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1
821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 18
36 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 185
1 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866
1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1
882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 18
97 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 191
2 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927
1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1
943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 19
58 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 197
3 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
```

In [0]:

```
#TSNE for tfidf W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=10, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf w2v perplexity = 10 & # iterations=2500')
plt.show()
```

In [0]:

```
#TSNE for tfidf W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=25, n_iter=2500)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf w2v perplexity = 25 & # iterations=2500')
plt.show()
```



In [0]:

```
#TSNE for tfidf W2V
#perplexity=10 and #iterations=2500
tsne_model = TSNE(n_components=2, random_state=0, perplexity=50, n_iter=5000)
#converting sparse to dense matrix
data = tsne_model.fit_transform(final)
tsne = np.vstack((data.T, final_labels)).T
df = pd.DataFrame(data=tsne, columns=("dim1", "dim2", "label"))

sns.FacetGrid(df, hue="label", size=7).map(plt.scatter, 'dim1', 'dim2').add_legend()
plt.title('TSNE tfidf w2v perplexity = 50 & # iterations=5000')
plt.show()
```

Observations and Questions

1. TSNE indicates that positive and negative reviews overlap a lot in 2D.
2. The patterns between BoW TSNE and TFIDF/W2V TSNE are completely different. Would like to know what causes this.
3. Does TSNE works better with large data sets? I understand algorithms require large data, but internally what actually happens with algorithms when we feed large data?