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.

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 [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [0]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=94731898
9803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%
3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.tes
t%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2F
auth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readon
ly&response_type=code

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

In [0]:

```
!ls 'gdrive/My Drive/Machine Learning/Assignmnets/Assignment2/amazon-fine-food-reviews'
```

```
'Copy of 02 Amazon Fine Food Reviews Analysis_TSNE.ipynb'   Reviews.csv
 database.sqlite
```

In [0]:

```
con = sqlite3.connect('gdrive/My Drive/Machine Learning/Assignmnets/Assignment2/amazon-fi
ne-food-reviews/database.sqlite')
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000""
", con)
```

[1]. Reading Data

In [0]:

```
# 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 points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000
00""", con)
# for tsne assignment you can take 5k data points

#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000"
"", con)

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

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	1303862400
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	1346976000
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	1219017600

In [0]:

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

In [0]:

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

```
(80668, 7)
```

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering	2

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
	R115TNMSPFT9I7					the price...	
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [0]:

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

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [0]:

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

Out[0]:

```
393063
```

Exploratory Data Analysis

[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 [0]:

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

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600

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 [0]:

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

In [0]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='fi
rst', inplace=False)
final.shape
```

Out[0]:

```
(4986, 10)
```

In [0]:

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

Out[0]:

```
99.72
```

Observation:- **It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions**

In [0]:

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

```
display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200

In [0]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [0]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(4986, 10)
```

Out[0]:

```
Positive    4178
Negative     808
Name: Score, dtype: int64
```

[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 [0]:

```
# 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.am
azon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br />The Victor M380 and M502
traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  T
he best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I b
ought some more through amazon and shared with family and friends.  I am a little disappo
inted that there are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are thicker and cr
unchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I'm sorry; but these reviews do nobody any good beyo
nd reminding us to look  before ordering.<br /><br />These are chocolate-oatmeal cookies.
If you don't like that combination, don't order this type of cookie.  I find the combo qu
ite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the co
okie sort of a coconut-type consistency.  Now let's also remember that tastes differ; so,
I've given my opinion.<br /><br />Then, these are soft, chewy cookies -- as advertised.
They are not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."  I h
appen to like raw cookie dough; however, I don't see where these taste like raw cookie do
ugh.  Both are soft, however, so is this the confusion?  And, yes, they stick together.
Soft cookies tend to do that.  They aren't individually wrapped, which would add to the c
ost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you wa
nt something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie that'
s soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try.
I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great c
offee.  dcaf is very good as well
==================================================
```

In [0]:

```
# 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 M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, bu
t only right nearby.
```

In [0]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tag
s-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
```

```
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380
and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right
nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  T
he best thing was that there were a lot of "brown" chips in the bsg (my favorite), so I b
ought some more through amazon and shared with family and friends.  I am a little disappo
inted that there are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion flavor because they
do not seem to be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are thicker and cr
unchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I'm sorry; but these reviews do nobody any good beyo
nd reminding us to look  before ordering.These are chocolate-oatmeal cookies.  If you don
't like that combination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort
of a coconut-type consistency.  Now let's also remember that tastes differ; so, I've give
n my opinion.Then, these are soft, chewy cookies -- as advertised.  They are not "crispy"
cookies, or the blurb would say "crispy," rather than "chewy."  I happen to like raw cook
ie dough; however, I don't see where these taste like raw cookie dough.  Both are soft, h
owever, so is this the confusion?  And, yes, they stick together.  Soft cookies tend to d
o that.  They aren't individually wrapped, which would add to the cost.  Oh yeah, chocola
te chip cookies tend to be somewhat sweet.So, if you want something hard and crisp, I sug
gest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy and tastes like a co
mbination of chocolate and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.
dcaf is very good as well
```

In [0]:

```
# 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 [0]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; t
he other wants crispy cookies.  Hey, I am sorry; but these reviews do nobody any good bey
```

```
ond reminding us to look  before ordering.<br /><br />These are chocolate-oatmeal cookies
.  If you do not like that combination, do not order this type of cookie.  I find the com
bo quite nice, really.  The oatmeal sort of "calms" the rich chocolate flavor and gives t
he cookie sort of a coconut-type consistency.  Now let is also remember that tastes diffe
r; so, I have given my opinion.<br /><br />Then, these are soft, chewy cookies -- as adve
rtised.  They are not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where these taste like raw
cookie dough.  Both are soft, however, so is this the confusion?  And, yes, they stick to
gether.  Soft cookies tend to do that.  They are not individually wrapped, which would ad
d to the cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So,
if you want something hard and crisp, I suggest Nabiso is Ginger Snaps.  If you want a co
okie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give the
se a try.  I am here to place my second order.
==================================================
```

In [0]:

```python
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The
Victor  and  traps are unreal, of course -- total fly genocide. Pretty stinky, but only r
ight nearby.
```

In [0]:

```python
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ordering the oth
er wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond remind
ing us to look before ordering br br These are chocolate oatmeal cookies If you do not lik
e that combination do not order this type of cookie I find the combo quite nice really Th
e oatmeal sort of calms the rich chocolate flavor and gives the cookie sort of a coconut
type consistency Now let is also remember that tastes differ so I have given my opinion b
r br Then these are soft chewy cookies as advertised They are not crispy cookies or the b
lurb would say crispy rather than chewy I happen to like raw cookie dough however I do no
t see where these taste like raw cookie dough Both are soft however so is this the confus
ion And yes they stick together Soft cookies tend to do that They are not individually wr
apped which would add to the cost Oh yeah chocolate chip cookies tend to be somewhat swee
t br br So if you want something hard and crisp I suggest Nabiso is Ginger Snaps If you w
ant a cookie that is soft chewy and tastes like a combination of chocolate and oatmeal gi
ve these a try I am here to place my second order
```

In [0]:

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves'
, 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', '
his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'th
ey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had
', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any'
, 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 
'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now
', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'might
n', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
sn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:

```
# 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 stopword
s)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 4986/4986 [00:02<00:00, 2381.08it/s]
```

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sor
ry reviews nobody good beyond reminding us look ordering chocolate oatmeal cookies not li
ke combination not order type cookie find combo quite nice really oatmeal sort calms rich
chocolate flavor gives cookie sort coconut type consistency let also remember tastes diff
er given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy
rather chewy happen like raw cookie dough however not see taste like raw cookie dough sof
t however confusion yes stick together soft cookies tend not individually wrapped would a
dd cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp sugg
est nabiso ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal
give try place second order'
```

[3.2] Preprocess Summary

Inspecting Summary

In [0]:

```
for i in range(10):
    print("Before Data Cleaning of Summary #",i)
    print(final.Summary[i+10])

    print()
```

```
Before Data Cleaning of Summary # 0
The Best Hot Sauce in the World

Before Data Cleaning of Summary # 1
My cats LOVE this "diet" food better than their regular food

Before Data Cleaning of Summary # 2
My Cats Are Not Fans of the New Food
```

```
Before Data Cleaning of Summary # 3
fresh and greasy!

Before Data Cleaning of Summary # 4
Strawberry Twizzlers - Yummy

Before Data Cleaning of Summary # 5
Lots of twizzlers, just what you expect.

Before Data Cleaning of Summary # 6
poor taste

Before Data Cleaning of Summary # 7
Love it!

Before Data Cleaning of Summary # 8
GREAT SWEET CANDY!

Before Data Cleaning of Summary # 9
Home delivered twizlers
```

Observations

1. **One Line Sentences 2.Data Cleaning of Summary is easy compared to text**

Data Wrangling

In [0]:

```
# Reference for  list of contractions in English took from http://stackoverflow.com/quest
ions/19790188/expanding-english-language-contractions-in-python
contractions = {
"ain't": "am not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he would",
"he'd've": "he would have",
"he'll": "he will",
"he's": "he is",
"how'd": "how did",
"how'll": "how will",
"how's": "how is",
"i'd": "i would",
"i'll": "i will",
"i'm": "i am",
"i've": "i have",
"isn't": "is not",
"it'd": "it would",
"it'll": "it will",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"must've": "must have",
"mustn't": "must not",
"needn't": "need not",
"oughtn't": "ought not",
"shan't": "shall not",
"sha'n't": "shall not",
"she'd": "she would",
"she'll": "she will",
"she's": "she is",
"should've": "should have",
"shouldn't": "should not",
"that'd": "that would",
"that's": "that is",
"there'd": "there had",
"there's": "there is",
"they'd": "they would",
"they'll": "they will",
"they're": "they are",
"they've": "they have",
"wasn't": "was not",
"we'd": "we would",
"we'll": "we will",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"where'd": "where did",
"where's": "where is",
"who'll": "who will",
"who's": "who is",
"won't": "will not",
"wouldn't": "would not",
"you'd": "you would",
"you'll": "you will",
"you're": "you are"
}
```

In [0]:

```
def clean_Summary(sentance):
  # using contractions to replace short with their longer forms
    if True:
        sentance= sentance.split()
        new_text = []
        for word in sentance:
            if word in contractions:
                new_text.append(contractions[word])
            else:
                new_text.append(word)
        sentance = " ".join(new_text)
        return sentance
```

In [0]:

```
from tqdm import tqdm
preprocessed_Summary = []
# 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 = clean_Summary(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 stopword
s)
```

```
    preprocessed_Summary.append(sentance.strip())
```

```
100%|██████████| 4986/4986 [00:01<00:00, 3412.86it/s]
```

Inspecting random Data after Summary Wrangling

In [0]:

```
for i in range(15):
    print(preprocessed_Summary[i+9])
```

```
buy product unless looking shredded coconut
little flavor
staple house
favorite quick meal solution
best hot sauce taco sauce available america
pico pica best
stuff
everyone saying pico pica true
not edible
shining star
inexpensive alternative gold leaf
create exquisite cake decorations
gold dust awesome
perfect sons cake
really cute made great golf cake
```

In [0]:

```
len(preprocessed_Summary)
```

Out[0]:

```
4986
```

BAG OF WORDS

In [0]:

```
#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_Summary)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abd
ominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
```

[4] Featurization

Ways to convert text to vector

Bi-Grams and n-Grams.

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))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules
/generated/sklearn.feature_extraction.text.CountVectorizer.html
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_cou
nts.get_shape()[1])
```

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

In [0]:

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_Summary)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[
0:10])
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.get
_shape()[1])
```

```
some sample features(unique words in the corpus) ['absolutely', 'almost', 'alternative',
'amazing', 'amazon', 'another', 'available', 'awesome', 'awful', 'baby']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 286)
the number of unique words including both unigrams and bigrams  286
```

[4.4] Word2Vec

In [0]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_Summary:
    list_of_sentance.append(sentance.split())
```

In [0]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin'
, binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to t
rain your own w2v ")
```

```
WARNING: Logging before flag parsing goes to stderr.
W0614 04:27:48.969915 140642358486912 base_any2vec.py:723] consider setting layer size to
a multiple of 4 for greater performance
W0614 04:27:49.075359 140642358486912 base_any2vec.py:1386] under 10 jobs per worker: con
sider setting a smaller `batch_words' for smoother alpha decay
```

```
[('best', 0.9772293567657471), ('good', 0.9760785102844238), ('not', 0.9742113351821899),
('taste', 0.9728367328643799), ('chips', 0.9695990085601807), ('free', 0.9652099013328552
), ('tea', 0.9613224864006042), ('flavor', 0.9591480493545532), ('love', 0.95810878276824
95), ('better', 0.9580333232879639)]
==================================================
[('excellent', 0.771611750125885), ('high', 0.7410725951194763), ('alternative', 0.735545
27759552), ('find', 0.7355020642280579), ('cups', 0.7351651191711426), ('green', 0.734540
8201217651), ('cup', 0.7321181893348694), ('better', 0.729761004447937), ('wonderful', 0.
7251964211463928), ('tastes', 0.7229461669921875)]
```

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  519
sample words  ['wow', 'make', 'great', 'product', 'good', 'stuff', 'premium', 'quality',
'dog', 'food', 'cats', 'love', 'nice', 'big', 'flavor', 'summer', 'treat', 'fat', 'free',
'guilt', 'buy', 'looking', 'coconut', 'little', 'house', 'favorite', 'quick', 'meal', 'so
lution', 'best', 'hot', 'sauce', 'available', 'everyone', 'true', 'not', 'edible', 'inexp
ensive', 'alternative', 'leaf', 'cake', 'awesome', 'perfect', 'really', 'cute', 'made', '
fantastic', 'beans', 'yum', 'yummy']
```

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

[4.4.1.1] Avg W2v

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
```

```
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to ch
ange this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 4986/4986 [00:00<00:00, 32937.74it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_Summary)
# 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 [0]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#              tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 4986/4986 [00:00<00:00, 9213.86it/s]
```

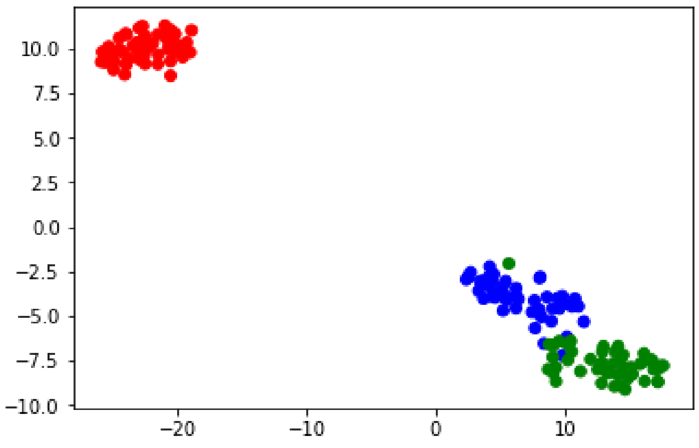
[5] Applying TSNE

1. **you need to plot 4 tsne plots with each of these feature set**
    A. **Review text, preprocessed one converted into vectors using (BOW)**
    B. **Review text, preprocessed one converted into vectors using (TFIDF)**
    C. **Review text, preprocessed one converted into vectors using (AVG W2v)**
    D. **Review text, preprocessed one converted into vectors using (TFIDF W2v)**
2. **Note 1: The TSNE accepts only dense matrices**

3. **Note 2: Consider only 5k to 6k data points**

In [0]:

```
# https://github.com/pavlin-policar/fastTSNE you can try this also, this version is littl
e faster than sklearn
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt

iris = datasets.load_iris()
x = iris['data']
y = iris['target']

tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)

X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit_transform(x.toarr
ay()) , .toarray() will convert the sparse matrix into dense matrix

for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score'])
colors = {0:'red', 1:'blue', 2:'green'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Score
'].apply(lambda x: colors[x]))
plt.show()
```



[5.1] Applying TNSE on Text BOW vectors

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsect
ion
# 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
```

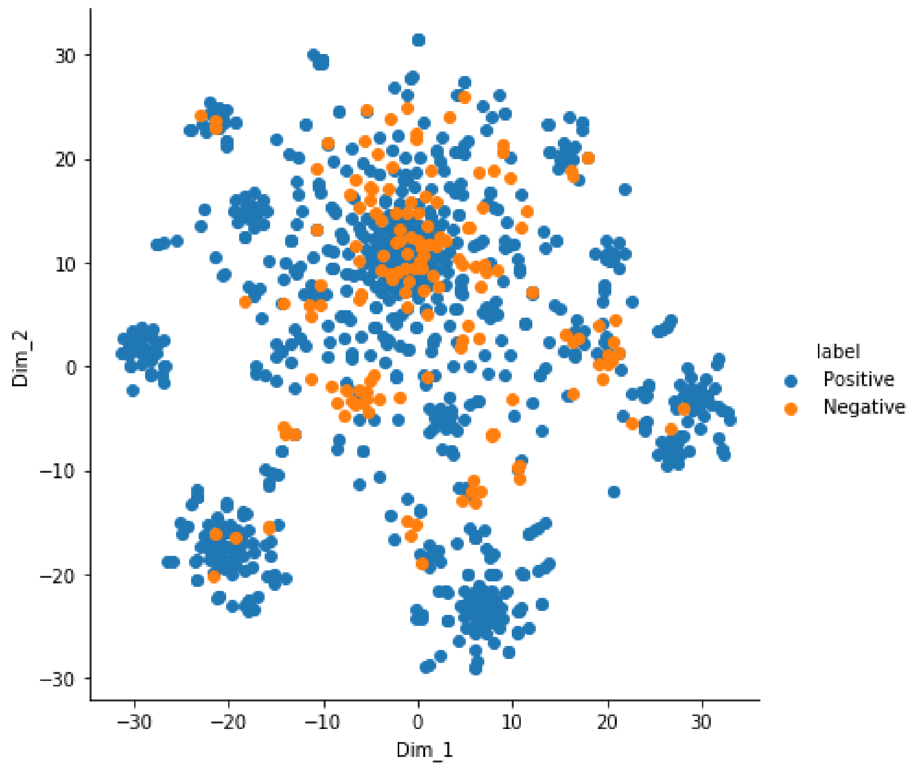
In [0]:

```
final_counts=final_counts.todense()
```

In [0]:

```
# TSNE for some Datapoints from whole dataset
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings
```

```
warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = final_counts[:1000]
score=final['Score']
score_1000 = score[:1000]

model = TSNE(n_components=2, random_state=0,perplexity=20,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()
plt.show()
```

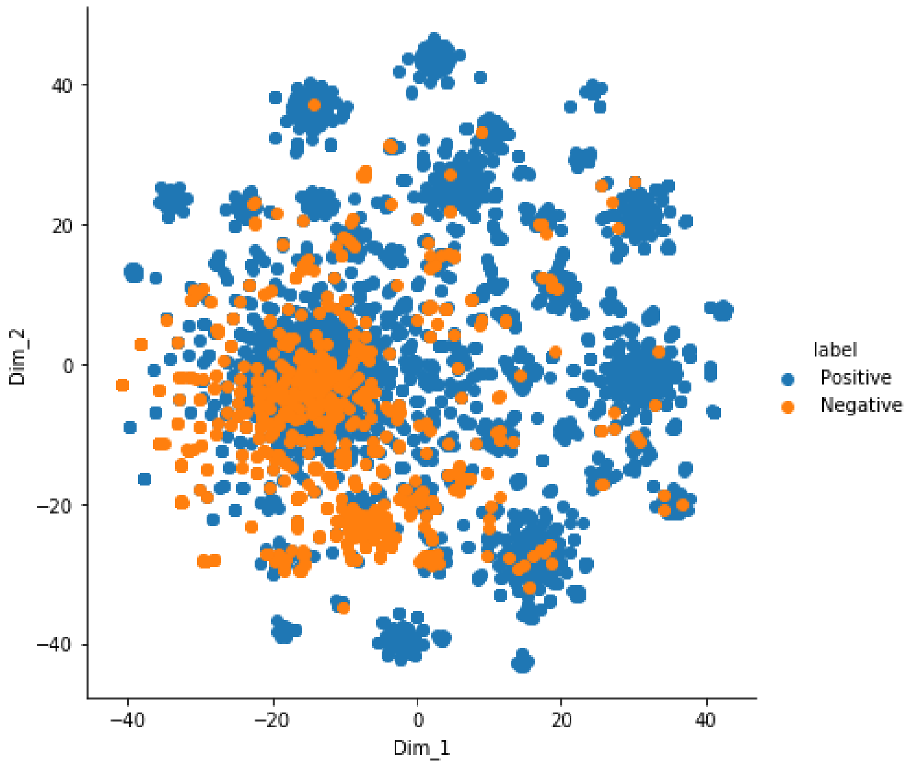


```
CPU times: user 30.2 s, sys: 409 ms, total: 30.7 s
Wall time: 30.2 s
```

In [0]:

```
# TSNE with perplexity=20
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = final_counts[1:10000:,]
score=final['Score']
score_1000 = score[1:10000]
```

```
model = TSNE(n_components=2, random_state=0,perplexity=20,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()
plt.show()
```



```
CPU times: user 10min 25s, sys: 738 ms, total: 10min 26s
Wall time: 10min 26s
```

[5.1] Applying TNSE on Text TFIDF vectors

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsect
ion
# 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
```

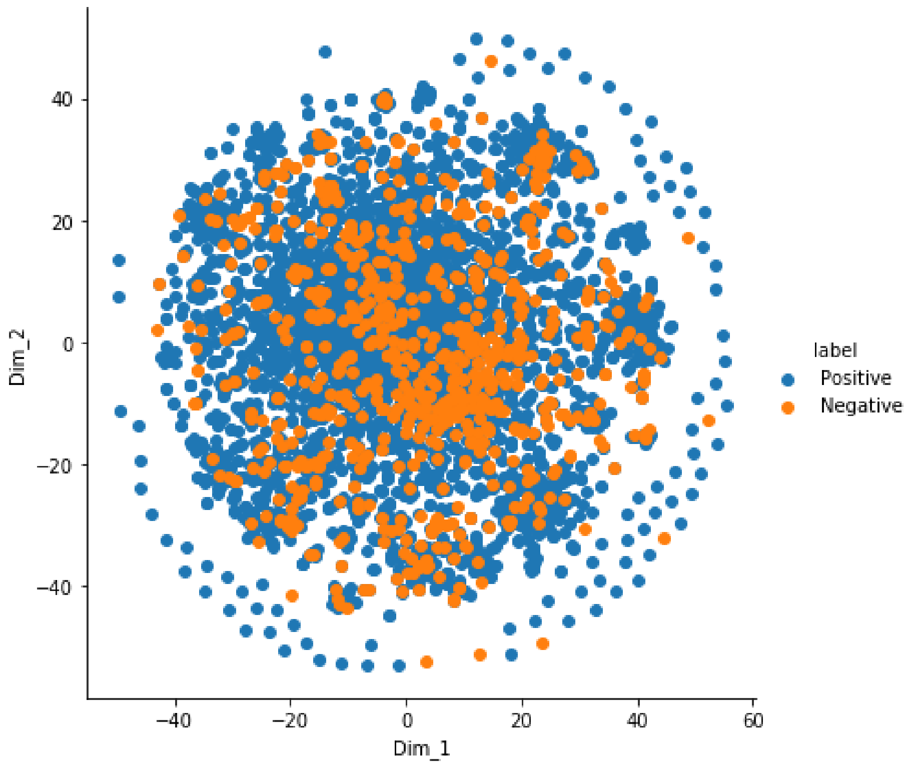
In [0]:

```
final_tf_idf=final_tf_idf.todense()
```

In [0]:

```
# TSNE with perplexity=20
%%time
from sklearn.manifold import TSNE
```

```
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = final_tf_idf[1:10000:,]
score=final['Score']
score_1000 = score[1:10000]

model = TSNE(n_components=2, random_state=0,perplexity=20,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()
plt.show()
```



```
CPU times: user 42.1 s, sys: 431 ms, total: 42.5 s
Wall time: 42.1 s
```

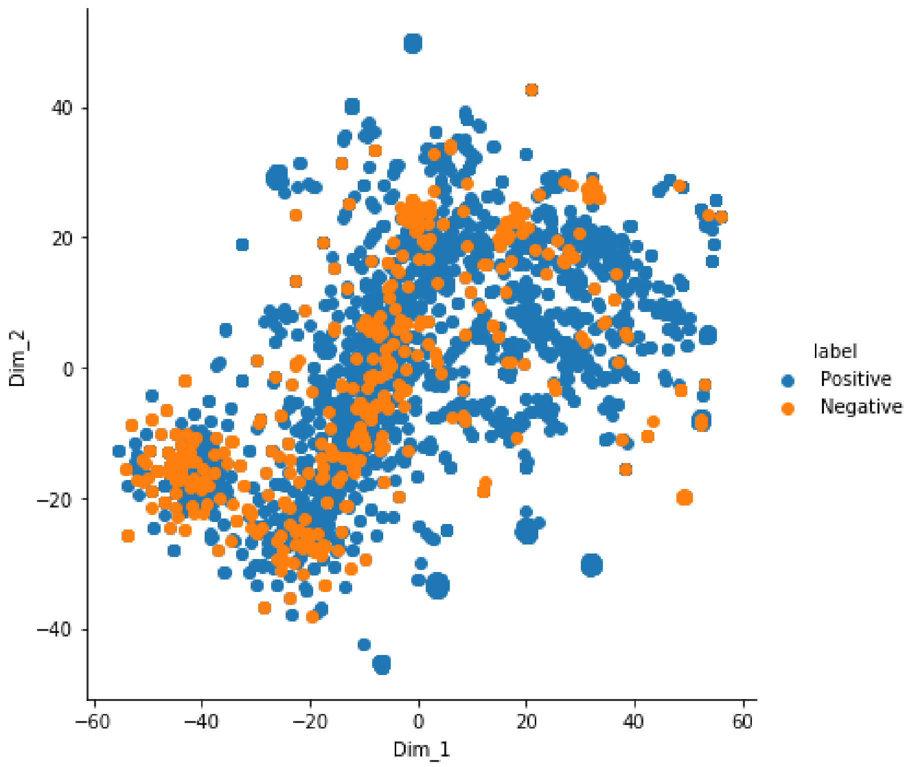
[5.3] Applying TNSE on Text Avg W2V vectors

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsect
ion
# 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
```

In [0]:

```
# TSNE with perplexity=20
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]

model = TSNE(n_components=2, random_state=0,perplexity=20,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()
plt.show()
```

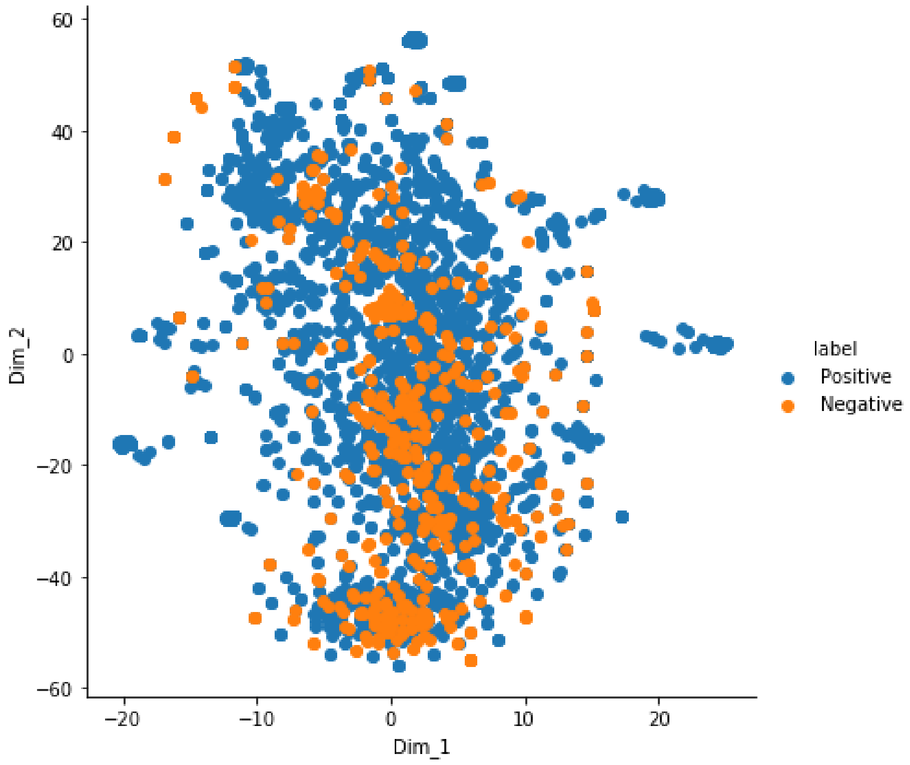


```
CPU times: user 13.7 s, sys: 444 ms, total: 14.1 s
Wall time: 13.7 s
```

In [0]:

```
# TSNE with perplexity=30
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')
```

```
# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 =sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]

model = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend
()
plt.show()
```



```
CPU times: user 17 s, sys: 378 ms, total: 17.4 s
Wall time: 17 s
```

[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

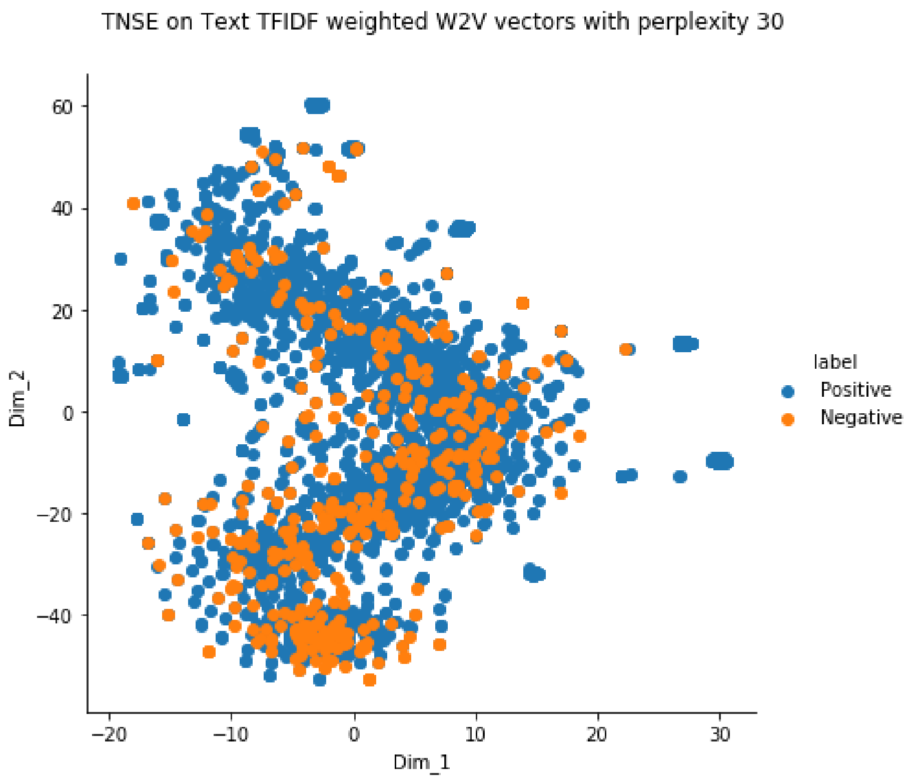
In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsect
ion
# 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
```

In [0]:

```
# TSNE with perplexity=30
%%time
```

```
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = tfidf_sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]

model = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
# Ploting the result of tsne

g=sn.FacetGrid(tsne_df,hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legen
d()
plt.subplots_adjust(top=0.9)
g.fig.suptitle('TNSE on Text TFIDF weighted W2V vectors with perplexity 30') # can also g
et the figure from plt.gcf()
plt.show()
```

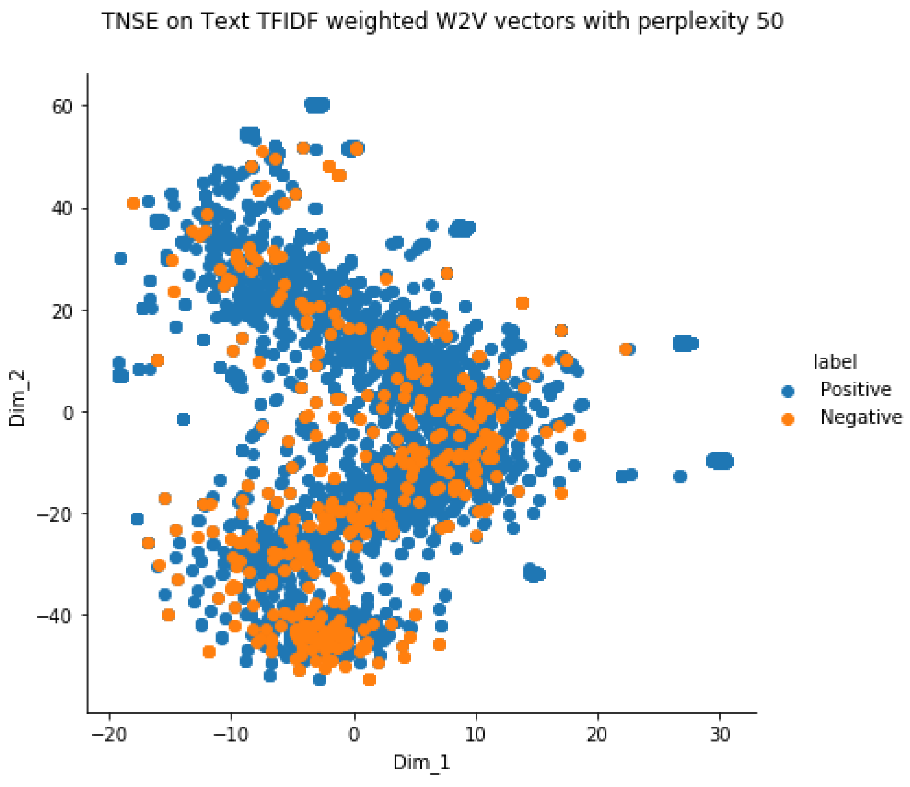


```
CPU times: user 17.6 s, sys: 508 ms, total: 18.1 s
Wall time: 17.7 s
```

In [0]:

```
# TSNE with perplexity=50
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')
```

```
data_1000 = tfidf_sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]

model = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=500)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
# Ploting the result of tsne

g=sn.FacetGrid(tsne_df,hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legen
d()
plt.subplots_adjust(top=0.9)
g.fig.suptitle('TNSE on Text TFIDF weighted W2V vectors with perplexity 50') # can also g
et the figure from plt.gcf()
plt.show()
```



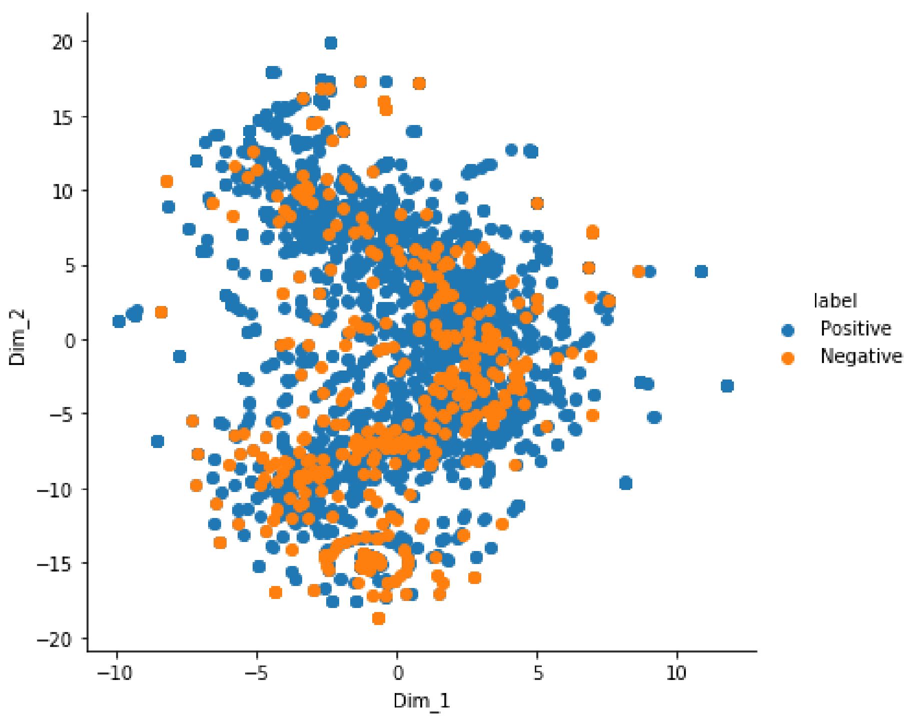
```
CPU times: user 17.5 s, sys: 503 ms, total: 18 s
Wall time: 17.6 s
```

In [0]:

```
# TSNE with perplexity=50 and iterations=300
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = tfidf_sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]
```

```
model = TSNE(n_components=2, random_state=0,perplexity=50,n_iter=300)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
# Ploting the result of tsne

g=sn.FacetGrid(tsne_df,hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legen
d()
plt.subplots_adjust(top=0.9)
g.fig.suptitle('TNSE on Text TFIDF weighted W2V vectors with perplexity 50 and n_iter=300
') # can also get the figure from plt.gcf()
plt.show()
```

TNSE on Text TFIDF weighted W2V vectors with perplexity 50 and n_iter=300

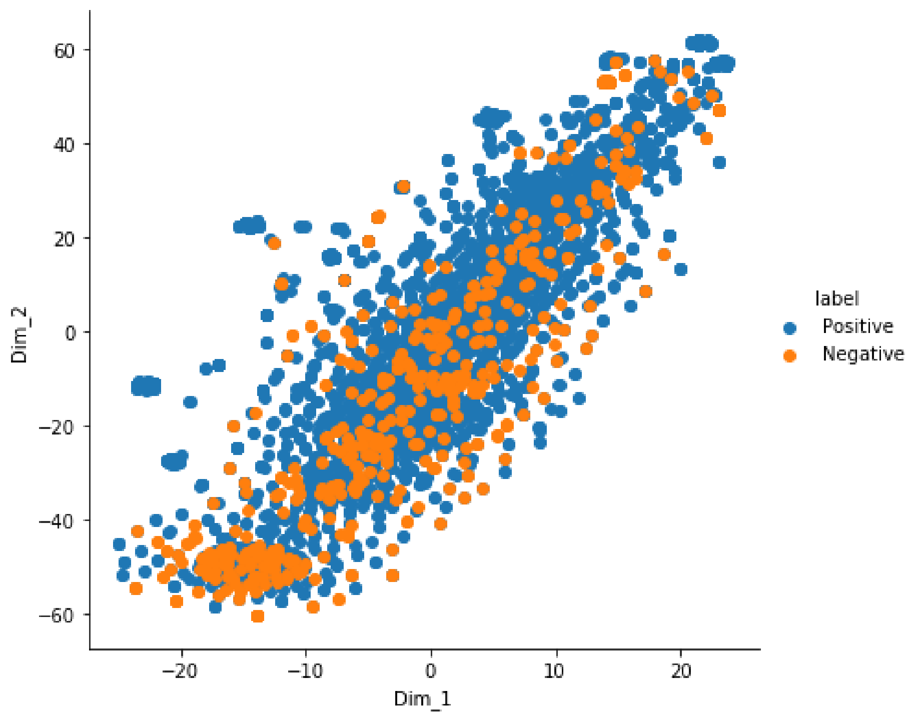


```
CPU times: user 10.5 s, sys: 474 ms, total: 11 s
Wall time: 10.6 s
```

In [0]:

```
# TSNE with perplexity=100 and iter=1000
%%time
from sklearn.manifold import TSNE
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = tfidf_sent_vectors[:3000]
score=final['Score']
score_1000 = score[:3000]

model = TSNE(n_components=2, random_state=0,perplexity=100,n_iter=1000)
# 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)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, score_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))
# Ploting the result of tsne

g=sn.FacetGrid(tsne_df,hue="label", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legen
d()
plt.subplots_adjust(top=0.9)
g.fig.suptitle('TNSE on Text TFIDF weighted W2V vectors with perplexity 100 and n_iter is
1000') # can also get the figure from plt.gcf()
plt.show()
```

TNSE on Text TFIDF weighted W2V vectors with perplexity 100 and n_iter is 1000



```
CPU times: user 42.1 s, sys: 515 ms, total: 42.6 s
Wall time: 42.2 s
```

[6] Conclusions

1. TSNE with Bag of Words :There is no linear division between Positive and negative.Its like overalapping and scattered one.There are some dense points of data.Tried different preplexity and no of iterations only change of shape is observed.

2. TSNE with TF-IDF too doesn't have linear division between Positive and negative.Its like overalapping and scattered one.Faster than BOW.There are some dense points of positive label compared to negative .

3. TSNE with Avg Word2Vec, TF-IDF Weighted Word2Vec too doesn't have linear division between approved and not approved projects.But tsne modeling is faster than BOW and the plot is like scattered one.Tried different preplexity and no of iterations got shape as linear in TFIDF weighted W2V vectors but still overlapping of data points are observed.

4. Imbalanced dataset .Dense of positive reviews are more compared to negative reviews.