Amazon Fine Food Reviews Analysis

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

EDA: [https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/)](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 [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")



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

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

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

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

from tqdm import tqdm
import os
```

[1]. Reading Data

In [2]:
```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 da
ta 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 500000""", con)
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a ne
gative rating.
def partition(x):
    if x < 3:
        return 0
    return 1

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

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:

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

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
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 [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

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

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

Out[6]: 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 [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inp
lace=False, kind='quicksort', na_position='last')
```

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

Out[9]: (4986, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 99.72

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

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulı
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(4986, 10)
```

Out[13]:
```
1    4178
0     808
Name: Score, dtype: int64
```

[3]. Text Preprocessing.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />h
ttp://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br />T
he Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pr
etty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these c
hips are.  The best thing was that there were a lot of "brown" chips in the b
sg (my favorite), so I bought some more through amazon and shared with family
and friends.  I am a little disappointed that there are not, so far, very man
y 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 crunchier 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; the other wants crispy cookies.  Hey, I'm sorry; but these reviews
do nobody any good beyond reminding us to look  before ordering.<br /><br />T
hese are chocolate-oatmeal cookies.  If you don't like that combination, do
n't order this type of cookie.  I find the combo quite nice, really.  The oat
meal 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 given my opinion.<br /><br />Then, these are soft, chewy cookies -- as a
dvertised.  They are not "crispy" cookies, or the blurb would say "crispy," r
ather than "chewy."  I happen to like raw cookie dough; however, I don't see
where these taste like raw cookie dough.  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 cost.  Oh yeah, choc
olate chip cookies tend to be somewhat sweet.<br /><br />So, if you want some
thing 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, gi
ve 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 cu
p is great coffee.  dcaf is very good as well
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />
/><br />The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-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. Prett
y stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these c
hips are.  The best thing was that there were a lot of "brown" chips in the b
sg (my favorite), so I bought some more through amazon and shared with family
and friends.  I am a little disappointed that there are not, so far, very man
y 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 crunchier 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; the other wants crispy cookies.  Hey, I'm sorry; but these reviews
do nobody any good beyond reminding us to look  before ordering.These are cho
colate-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 "c
alms" the rich chocolate flavor and gives the cookie sort of a coconut-type c
onsistency.  Now let's also remember that tastes differ; so, I've given my op
inion.Then, these are soft, chewy cookies -- as advertised.  They are not "cr
ispy" cookies, or the blurb would say "crispy," rather than "chewy."  I happe
n to like raw cookie dough; however, I don't see where these taste like raw c
ookie dough.  Both are soft, however, so is this the confusion?  And, yes, th
ey stick together.  Soft cookies tend to do that.  They aren't individually w
rapped, which would add to the cost.  Oh yeah, chocolate chip cookies tend to
be somewhat sweet.So, if you want something hard and crisp, I suggest Nabis
o'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.This k cup is g
reat coffee.  dcaf is very good as well
```

In [17]:
```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were
ordering; the other wants crispy cookies.  Hey, I am sorry; but these reviews
do nobody any good beyond reminding us to look  before ordering.<br /><br />T
hese are chocolate-oatmeal cookies.  If you do not like that combination, do
not order this type of cookie.  I find the combo quite nice, really.  The oat
meal 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.<br /><br />Then, these are soft, chewy cookies -- as a
dvertised.  They are not "crispy" cookies, or the blurb would say "crispy," r
ather than "chewy."  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 together.  Soft cookies tend to do that.
They are not individually wrapped, which would add to the cost.  Oh yeah, cho
colate chip cookies tend to be somewhat sweet.<br /><br />So, if you want som
ething hard and crisp, I suggest Nabiso is Ginger Snaps.  If you want a cooki
e that is soft, chewy and tastes like a combination of chocolate and oatmeal,
give these a try.  I am here to place my second order.
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/408403
9
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. P
retty stinky, but only right nearby.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were orde
ring the other wants crispy cookies Hey I am sorry but these reviews do nobod
y any good beyond reminding us to look before ordering br br These are chocol
ate oatmeal cookies If you do not like that combination do not order this typ
e 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 is also remember that tastes differ so I have given my opinion br br
Then these are soft chewy cookies as advertised They are not crispy cookies o
r the blurb would say crispy rather than chewy I happen to like raw cookie do
ugh however I do not see where these taste like raw cookie dough Both are sof
t however so is this the confusion And yes they stick together Soft cookies t
end to do that They are not individually wrapped which would add 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 cooki
e that is soft chewy and tastes like a combination of chocolate and oatmeal g
ive these a try I am here to place my second order
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st
step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'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', 'becau
se', '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', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d'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", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████
████| 4986/4986 [00:01<00:00, 3253.67it/s]
```

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

Out[23]:
```
'wow far two two star reviews one obviously no idea ordering wants crispy coo
kies hey sorry reviews nobody good beyond reminding us look ordering chocolat
e oatmeal cookies not like combination not order type cookie find combo quite
nice really oatmeal sort calms rich chocolate flavor gives cookie sort coconu
t type consistency let also remember tastes differ given opinion soft chewy c
ookies advertised not crispy cookies blurb would say crispy rather chewy happ
en like raw cookie dough however not see taste like raw cookie dough soft how
ever confusion yes stick together soft cookies tend not individually wrapped
would add cost oh yeah chocolate chip cookies tend somewhat sweet want someth
ing hard crisp suggest nabiso ginger snaps want cookie soft chewy tastes like
combination chocolate oatmeal give try place second order'
```

[3.2] Preprocess Summary

In [24]:
```
preprocessed_reviewsummary = []
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not i
n stopwords)
    preprocessed_reviewsummary.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████
████| 4986/4986 [00:01<00:00, 4703.67it/s]
```

In [25]:
```
len(preprocessed_reviewsummary)
```

Out[25]:
```
4986
```

[4] Featurization

[4.1] BAG OF WORDS

In [32]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

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

[4.2] Bi-Grams and n-Grams.

In [33]:
```
#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/sta
ble/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_counts.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
```

[4.3] TF-IDF

In [34]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_featu
re_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) ['ability', 'able', 'able fi
nd', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutel
y love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [43]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [36]:
```
# 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/#.W17SRFAzZP
Y
# 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-negati
ve300.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 train your own w2v ")
```

```
[('excellent', 0.9947681427001953), ('looking', 0.9937182068824768), ('thickn
ess', 0.9934208393096924), ('terrific', 0.9929349422454834), ('overall', 0.99
29120540618896), ('alternative', 0.9928317666053772), ('value', 0.99274325370
78857), ('care', 0.9926172494888306), ('inexpensive', 0.9924926161766052),
('microphone', 0.9924678206443787)]
==================================================
[('simply', 0.999352216720581), ('gain', 0.9993471503257751), ('bar', 0.99933
74347686768), ('somewhat', 0.9993264675140381), ('wife', 0.9993264079093933),
('awful', 0.9993148446083069), ('fairly', 0.9993109703063965), ('watchers',
0.999306321144104), ('wow', 0.9992793202400208), ('grew', 0.999275088310241
7)]
```

In [37]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky',
'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'shipmen
t', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'removed',
'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows', 'beauti
fully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'like', 'tv',
'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'window', 'ever
ybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [38]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
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 change 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 [00:04<00:00, 1145.32it/s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [39]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [40]:
```
# 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 [00:24<00:00, 204.46it/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 [41]:
```
# https://github.com/pavlin-policar/fastTSNE you can try this also, this versi
on is little 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']
k = iris['target']
print(type(k))

""""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_transf
orm(x.toarray()) , .toarray() will convert the sparse matrix into dense matrix
print(type(X_embedding),X_embedding.shape)
for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
z=y.reshape(-1,1)
print(z.shape)
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()"""
```

```
<class 'numpy.ndarray'>
```

Out[41]:
```
'"tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)\n\nX_embeddin
g = tsne.fit_transform(x)\n# if x is a sparse matrix you need to pass it as X
_embedding = tsne.fit_transform(x.toarray()) , .toarray() will convert the sp
arse matrix into dense matrix\nprint(type(X_embedding),X_embedding.shape)\nfo
r_tsne = np.hstack((X_embedding, y.reshape(-1,1)))\nz=y.reshape(-1,1)\nprint
(z.shape)\nfor_tsne_df = pd.DataFrame(data=for_tsne, columns=[\'Dimension_x
\',\'Dimension_y\',\'Score\'])\ncolors = {0:\'red\', 1:\'blue\', 2:\'green\'}
\nplt.scatter(for_tsne_df[\'Dimension_x\'], for_tsne_df[\'Dimension_y\'], c=f
or_tsne_df[\'Score\'].apply(lambda x: colors[x]))\nplt.show()'
```

[5.1] Applying TNSE on Text BOW vectors

In [0]:
```
# please write all the code with proper documentation, and proper titles for e
ach subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the rea
der
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

considering the same cleaned dataset which has 4986 reviews

caluclating BOW using the same code used in 4.1 with the already preprosessed reviews in preprocessed_reviews list

In [43]:
```
#considering the same cleaned dataset which has 4986 reviews
#caluclating BOW using the same code used in 4.1 with the already preprosessed
reviews in preprocessed_reviews list
count_vect = CountVectorizer()
count_vect.fit(preprocessed_reviews)
final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
```

converting scipy sparse matrix to dense matrix as tsne cannot be applied to sparce matrix

In [44]:
```
dense_counts=final_counts.todense()
type(dense_counts)
print(dense_counts.shape)
```

```
(4986, 12997)
```

applying tsne to dense matrix with default values of perplexity and iterations

In [87]:
```
#applying tsne to dense matrix
x=dense_counts
y=final["Score"]
tsne=TSNE(n_components=2, random_state=0)
xt=tsne.fit_transform(x)
print(xt.shape)
```

```
(4986, 12997)
```

converting pandas series into numpy array and stacking scores to this array

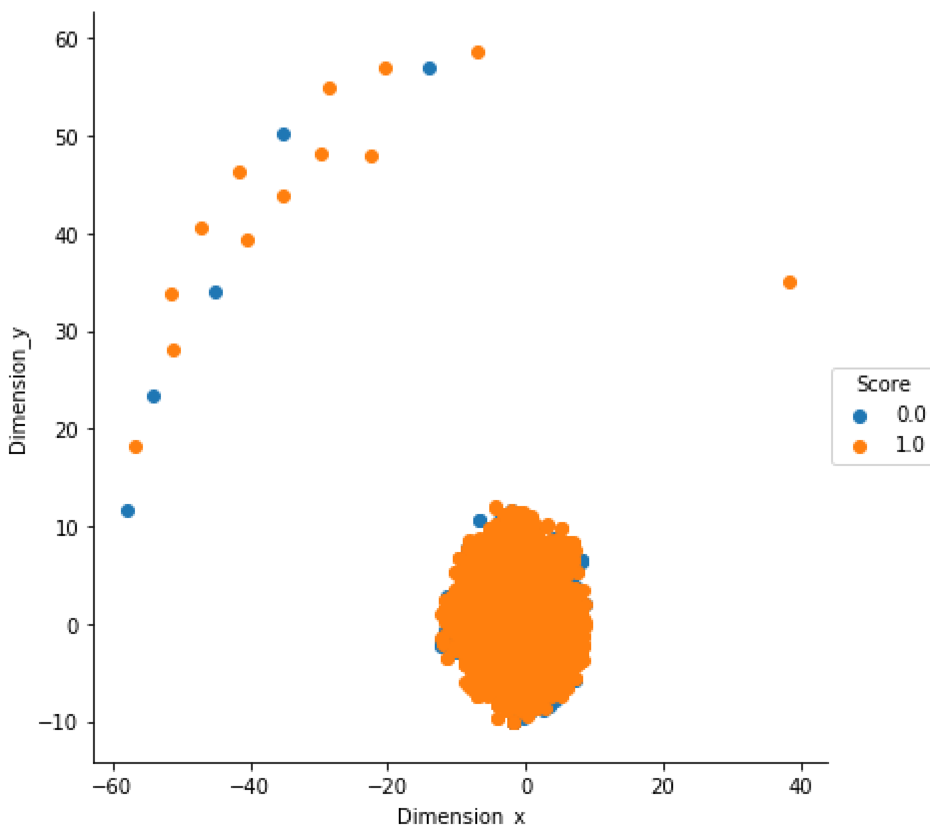
In [113]:
```
#converting pandas series into numpy array
e=y.as_matrix()
print(e.shape)
e1=e.reshape(-1,1)
completeframe=np.hstack((xt,e1))
print(completeframe.shape)

for_tsne_df = pd.DataFrame(data=completeframe, columns=['Dimension_x','Dimensi
on_y','Score'])
#colors = {0:'red', 1:'blue', 2:'green'}
#plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsn
e_df['Score'].apply(lambda x: colors[x]))
#plt.show()
```

```
(4986,)
(4986, 3)
```

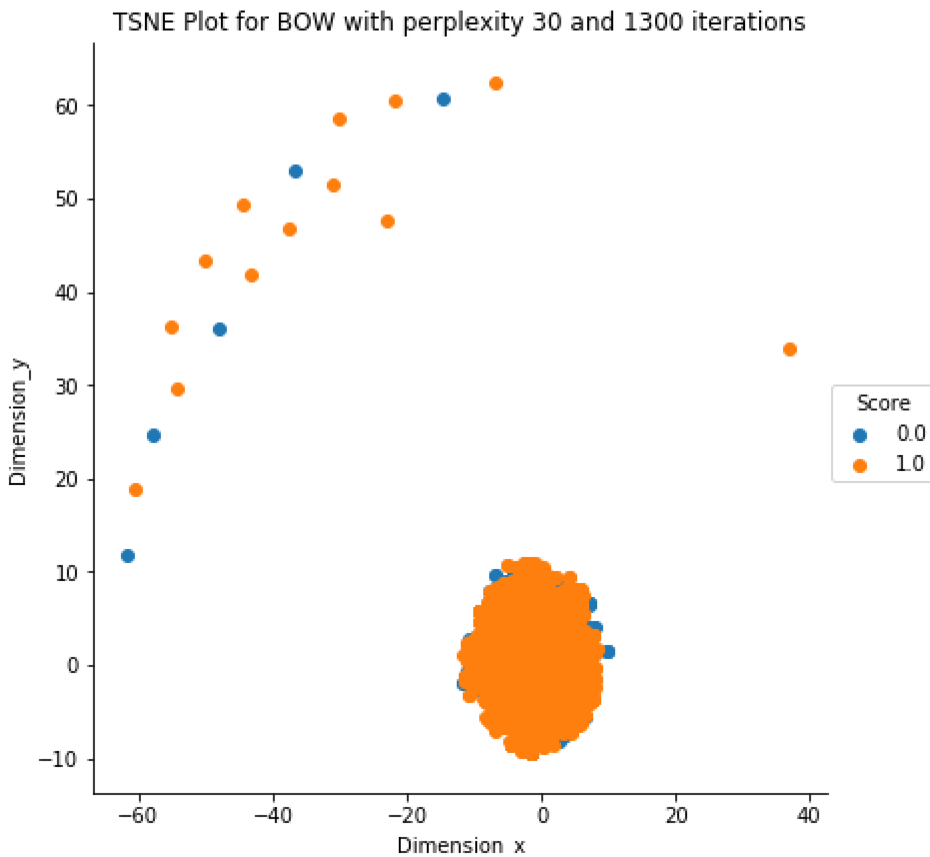
TSNE PLOT BOW for perplexity of 30 and 1000 iterations

In [116]:
```
import seaborn as sn
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.show()
```



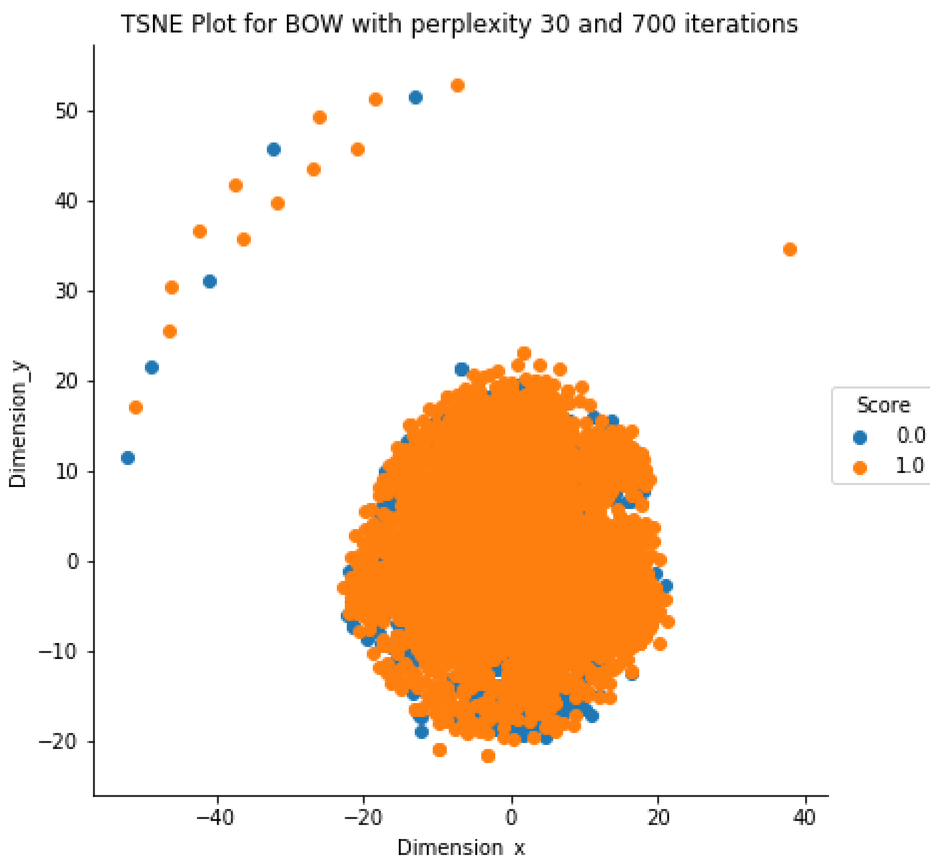
TSNE Plot for BOW with perplexity 30 and 1300 iterations

In [45]:
```
x2=dense_counts
y2=final["Score"]
tsne=TSNE(n_components=2, random_state=0, n_iter=1300)
xt2=tsne.fit_transform(x2)
e2=y2.as_matrix()
e21=e2.reshape(-1,1)
completeframe2=np.hstack((xt2,e21))
for_tsne_df = pd.DataFrame(data=completeframe2, columns=['Dimension_x','Dimens
ion_y','Score'])
#colors = {0:'red', 1:'blue', 2:'green'}
#plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsn
e_df['Score'].apply(lambda x: colors[x]))
#plt.show()
import seaborn as sn
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for BOW with perplexity 30 and 1300 iterations')
plt.show()
```



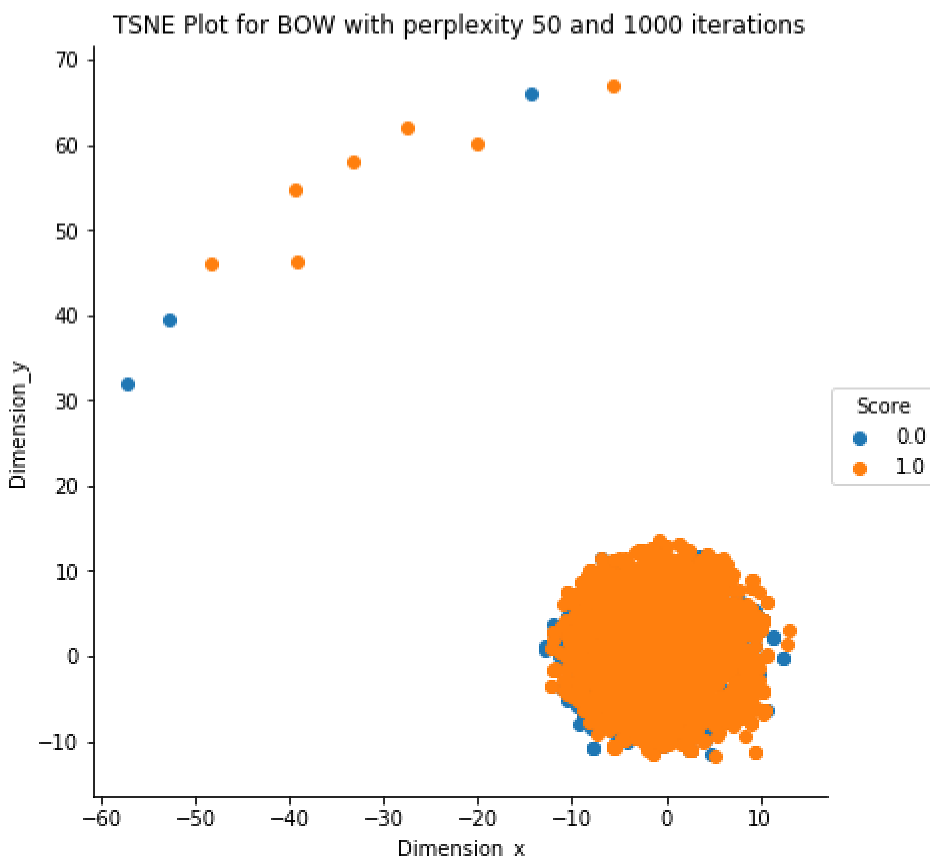
TSNE Plot for BOW with perplexity 30 and 700 iterations

In [46]:
```
x3=dense_counts
y3=final["Score"]
tsne=TSNE(n_components=2, random_state=0, n_iter=700)
xt3=tsne.fit_transform(x3)
e3=y3.as_matrix()
e31=e3.reshape(-1,1)
completeframe3=np.hstack((xt3,e31))
for_tsne_df = pd.DataFrame(data=completeframe3, 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]))
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for BOW with perplexity 30 and 700 iterations')
plt.show()
```



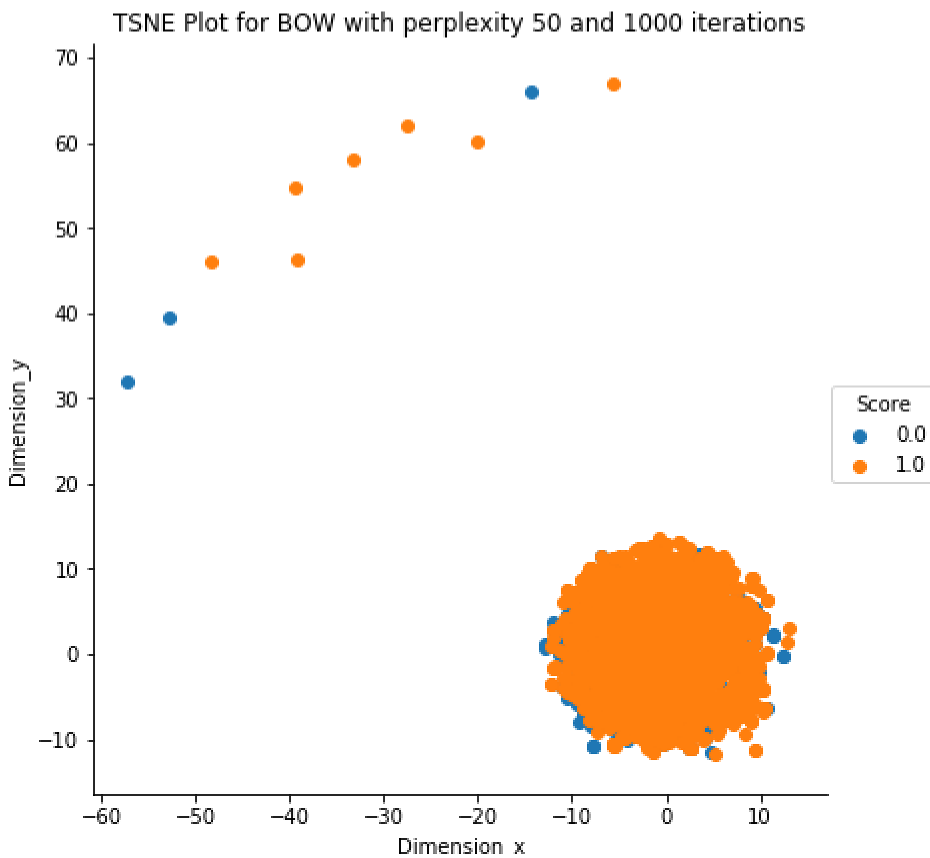
TSNE Plot for BOW with perplexity 50 and 1000 iterations

In [47]:
```
#for iterations  tested between 700-1300 the shapes seems  stable
x5=dense_counts
y5=final["Score"]
tsne=TSNE(n_components=2, random_state=0, perplexity=50)
xt5=tsne.fit_transform(x5)
e5=y5.as_matrix()
e51=e5.reshape(-1,1)
completeframe5=np.hstack((xt5,e51))
for_tsne_df = pd.DataFrame(data=completeframe5, columns=['Dimension_x','Dimens
ion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for BOW with perplexity 50 and 1000 iterations')
plt.show()
```



TSNE Plot for BOW with perplexity 50 and 1000 iterations

In [48]:
```
#for iterations  tested between 700-1300 the shapes seems  stable
x5=dense_counts
y5=final["Score"]
tsne=TSNE(n_components=2, random_state=0, perplexity=50)
xt5=tsne.fit_transform(x5)
e5=y5.as_matrix()
e51=e5.reshape(-1,1)
completeframe5=np.hstack((xt5,e51))
for_tsne_df = pd.DataFrame(data=completeframe5, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for BOW with perplexity 50 and 1000 iterations')
plt.show()
```



Conclusion TSNE on BOW:-

Both 0 and 1 datapoints seems to be highly overlaping suggesting there may be highly cluttered data in higher dimensions too

And Not able to classify the Score using BOW vector representation on reviews and applying TSNE on the same as there is no linear seperation as data ovelaps

The shape seems to be stable in 700-1300 iterations

Not much of a diffrence when there is a change in perplexity(Expected Behaviour as SKlearn TSNE document mentions the same that tsne is quite insenstive to this parameter)

[5.1] Applying TNSE on Text TFIDF vectors

In [0]:
```
# please write all the code with proper documentation, and proper titles for e
ach subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the rea
der
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

considering the same cleaned dataset which has 4986 reviews

caluclating TFIDF using the same code used in 4.3 with the already preprosessed reviews in preprocessed_reviews list

In [28]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
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])
```

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

converting scipy sparse matrix to dense matrix as tsne cannot be applied to sparce matrix

In [29]:
```
dense_matrix=final_tf_idf.todense()
print(type(dense_matrix))
print(dense_matrix.shape)
```

```
<class 'numpy.matrixlib.defmatrix.matrix'>
(4986, 3144)
```

applying TSNE with perplexity 30 and 1000 iterations

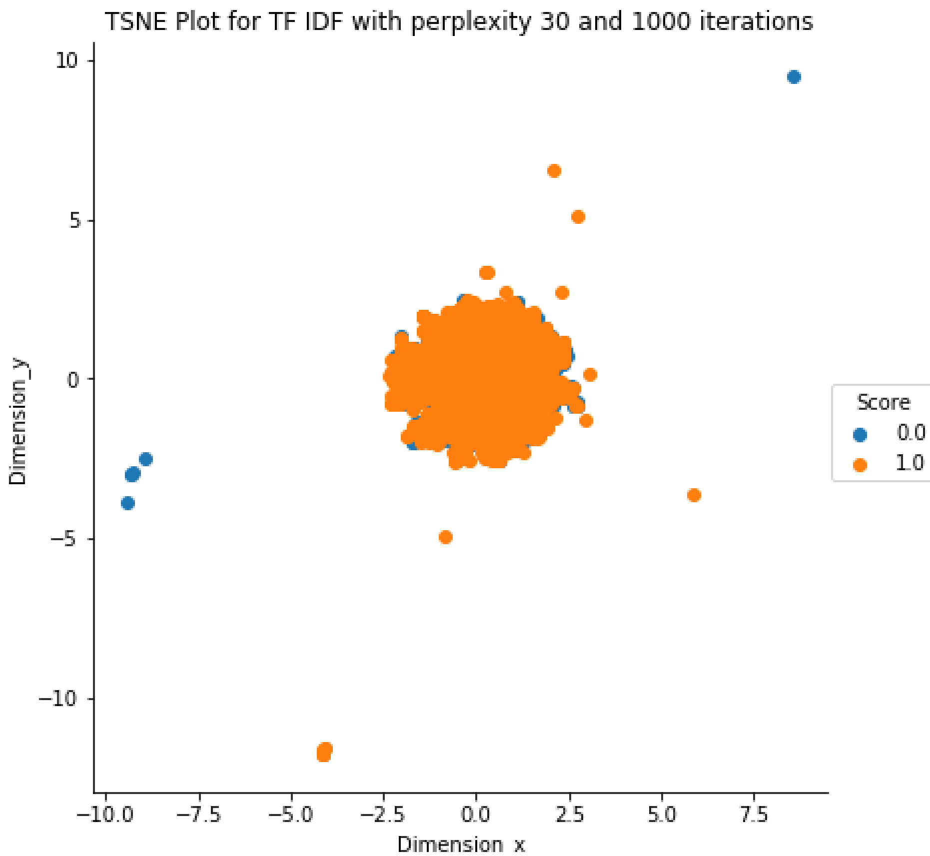
In [31]:
```
from sklearn.manifold import TSNE
tsne=TSNE(n_components=2, random_state=0)
tf_idf_tsne=tsne.fit_transform(dense_matrix)
print(tf_idf_tsne.shape)
y=final["Score"]
# Converting series to matrix
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
```

```
(4986, 2)
(4986, 1)
```

In [32]:
```
complte_tfidf=np.hstack((tf_idf_tsne,y))
print(complte_tfidf.shape)
print(type(complte_tfidf))
```

```
(4986, 3)
<class 'numpy.ndarray'>
```

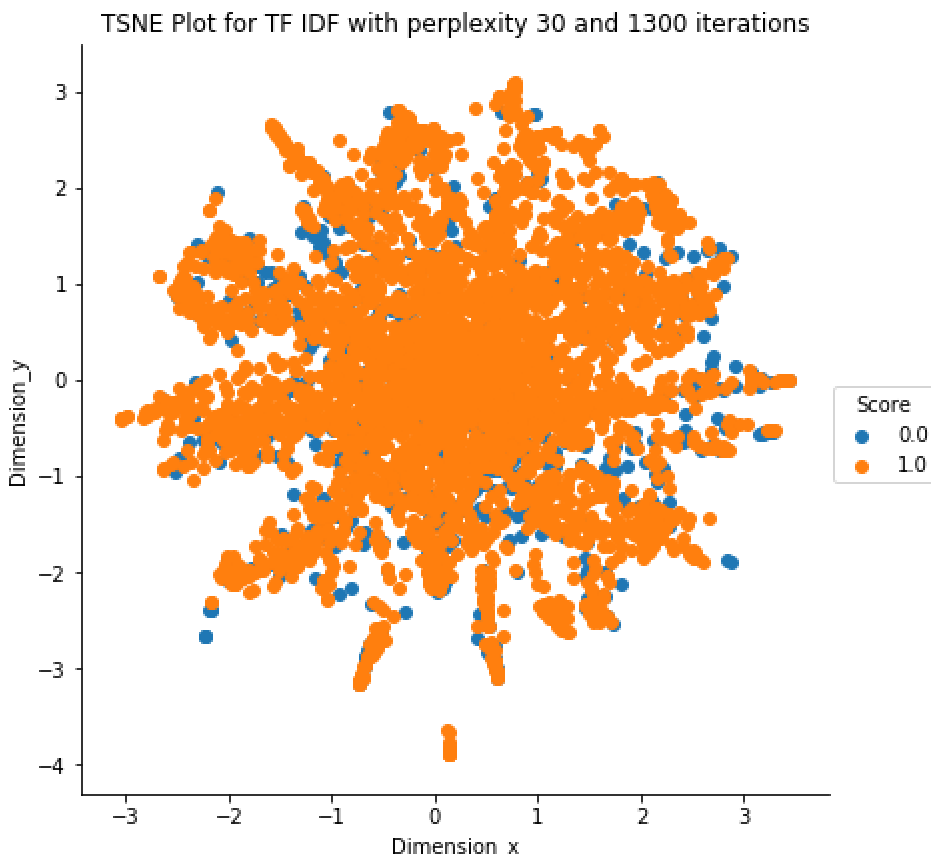
In [34]:
```
import seaborn as sn
for_tsne_df = pd.DataFrame(data=complte_tfidf, columns=['Dimension_x','Dimensi
on_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TF IDF with perplexity 30 and 1000 iterations')
plt.show()
```



TSNE Plot for TF IDF with perplexity 30 and 1300 iterations

In [35]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1300)
tf_idf_tsne=tsne.fit_transform(dense_matrix)
print(tf_idf_tsne.shape)
y=final["Score"]
# Converting series to matrix
y=y.as_matrix()
y=y.reshape(-1,1)
complte_tfidf=np.hstack((tf_idf_tsne,y))
for_tsne_df = pd.DataFrame(data=complte_tfidf, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for TF IDF with perplexity 30 and 1300 iterations')
plt.show()
```

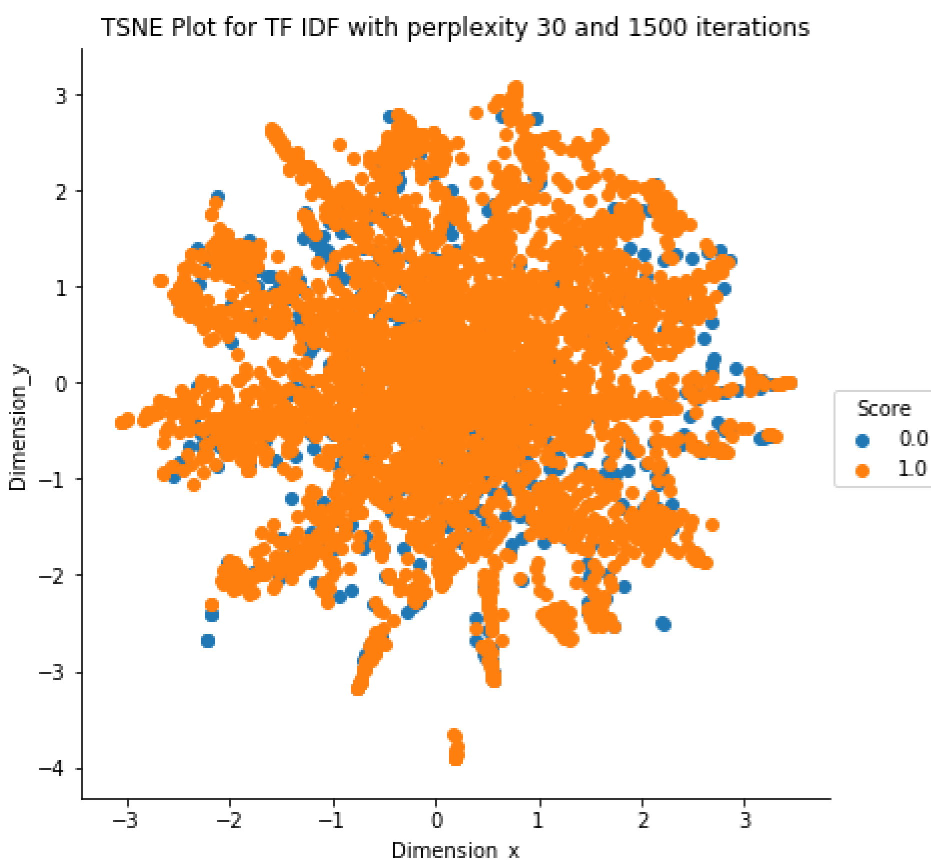
```
(4986, 2)
```



TSNE Plot for TF IDF with perplexity 30 and 1500 iterations

In [39]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1500)
tf_idf_tsne=tsne.fit_transform(dense_matrix)
print(tf_idf_tsne.shape)
y=final["Score"]
# Converting series to matrix
y=y.as_matrix()
y=y.reshape(-1,1)
complte_tfidf=np.hstack((tf_idf_tsne,y))
for_tsne_df = pd.DataFrame(data=complte_tfidf, columns=['Dimension_x','Dimensi
on_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TF IDF with perplexity 30 and 1500 iterations')
plt.show()
```

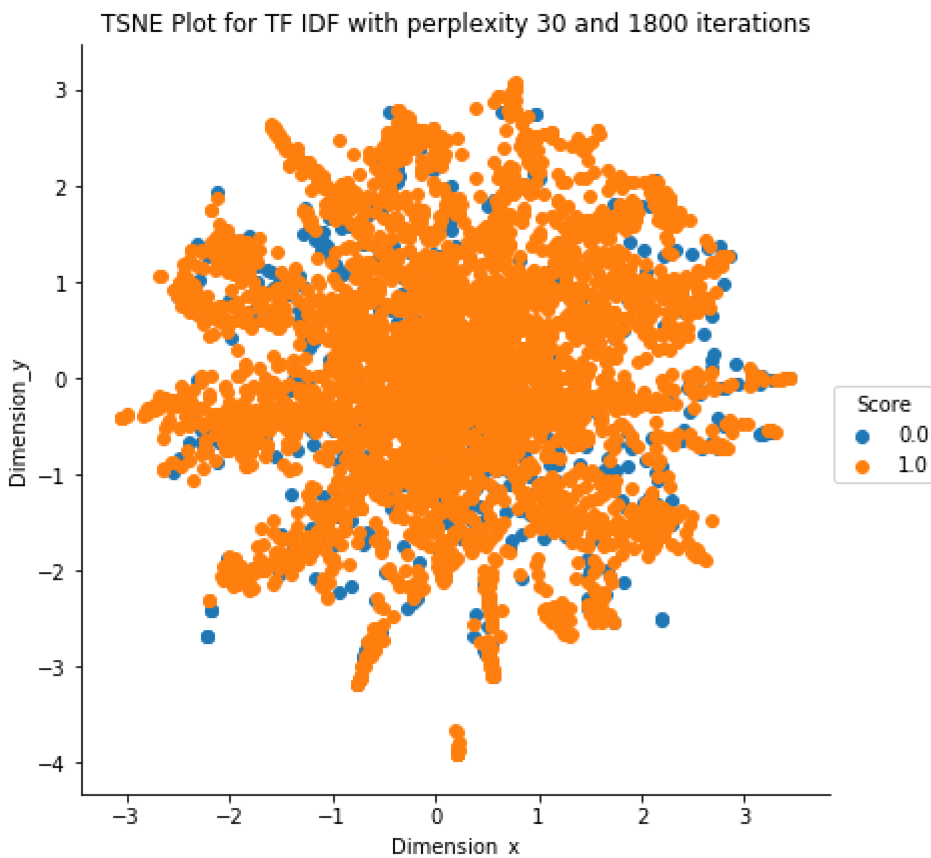
```
(4986, 2)
```



TSNE Plot for TF IDF with perplexity 30 and 1800 iterations

In [40]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1800)
tf_idf_tsne=tsne.fit_transform(dense_matrix)
y=final["Score"]
# Converting series to matrix
y=y.as_matrix()
y=y.reshape(-1,1)
complte_tfidf=np.hstack((tf_idf_tsne,y))
print(complte_tfidf.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidf, columns=['Dimension_x','Dimensi
on_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TF IDF with perplexity 30 and 1800 iterations')
plt.show()
```

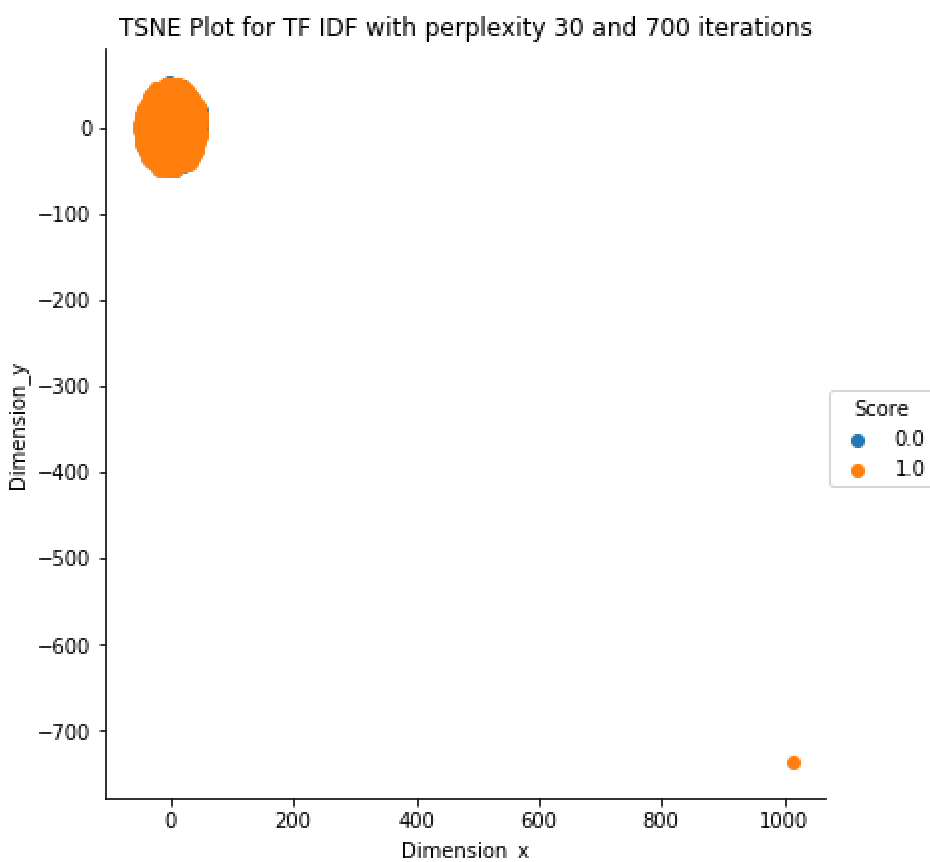
```
(4986, 3)
```



TSNE Plot for TF IDF with perplexity 30 and 700 iterations

In [41]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =700)
tf_idf_tsne=tsne.fit_transform(dense_matrix)
y=final["Score"]
# Converting series to matrix
y=y.as_matrix()
y=y.reshape(-1,1)
complte_tfidf=np.hstack((tf_idf_tsne,y))
print(complte_tfidf.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidf, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for TF IDF with perplexity 30 and 700 iterations')
plt.show()
```

```
(4986, 3)
```



Conclusion on applying TSNE on TFIDF Vectors

Both 0 and 1 (+ve and -ve review) datapoints seems to be highly overlaping suggesting there may be highly cluttered data in higher dimensions too

And Not able to classify the Score using TFIDF vector representation on reviews and applying TSNE on the same as there is no linear seperation as data ovelaps

The shape seems to be stable in 1300-1800 iterations

[5.3] Applying TNSE on Text Avg W2V vectors

In [0]:
```
# please write all the code with proper documentation, and proper titles for e
ach subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the rea
der
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

considering the same cleaned dataset which has 4986 reviews

caluclating W2V using the same code used in 4.4.1 with the already preprosessed reviews in preprocessed_reviews list

training w2v model using the processed amazon fine foods review data

In [48]:
```
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
```

Calculating average Word2Vec and storing it in a list sent_vectors

In [53]:
```
w2v_words = list(w2v_model.wv.vocab)
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
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 change 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))
```

```
100%|████████████████████████████████████████████████████████████████████████
████| 4986/4986 [00:03<00:00, 1417.81it/s]

4986
```

In [63]:
```
#converting the list to array
y=np.asarray(sent_vectors)
```

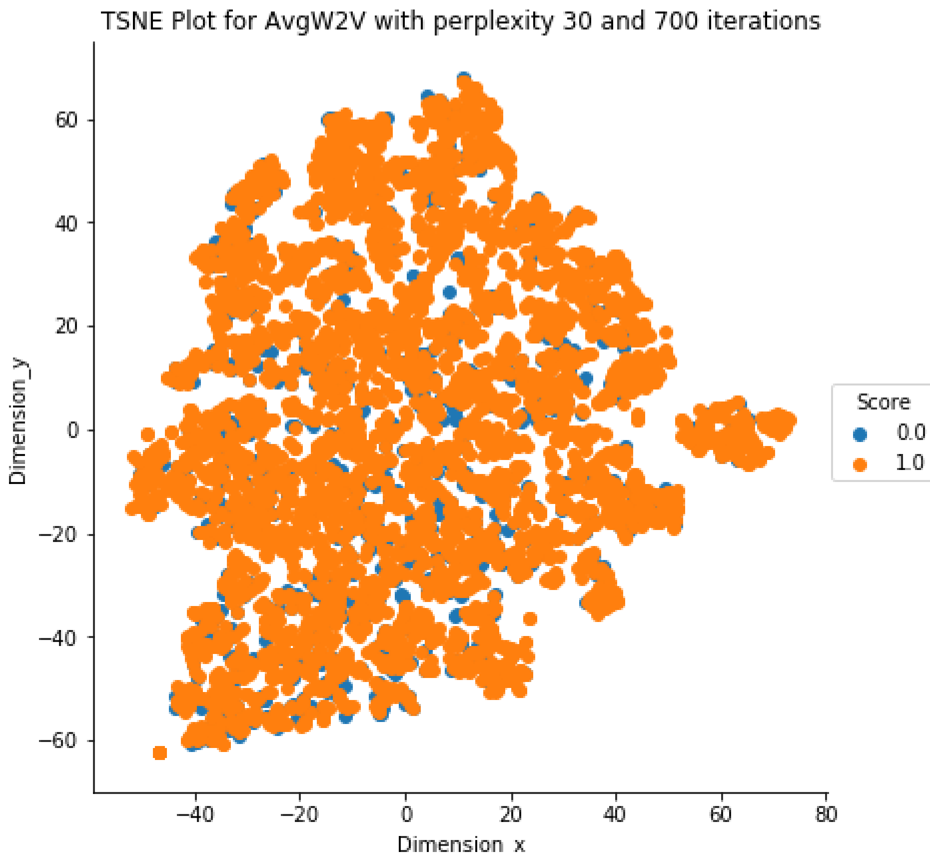
TSNE Plot for AvgW2V with perplexity 30 and 700 iterations

In [64]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =700)
avgw2v_tsne=tsne.fit_transform(y)
```

In [69]:
```
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_avgw2v=np.hstack((avgw2v_tsne,y))
print(complte_avgw2v.shape)
```

```
(4986, 1)
(4986, 3)
```

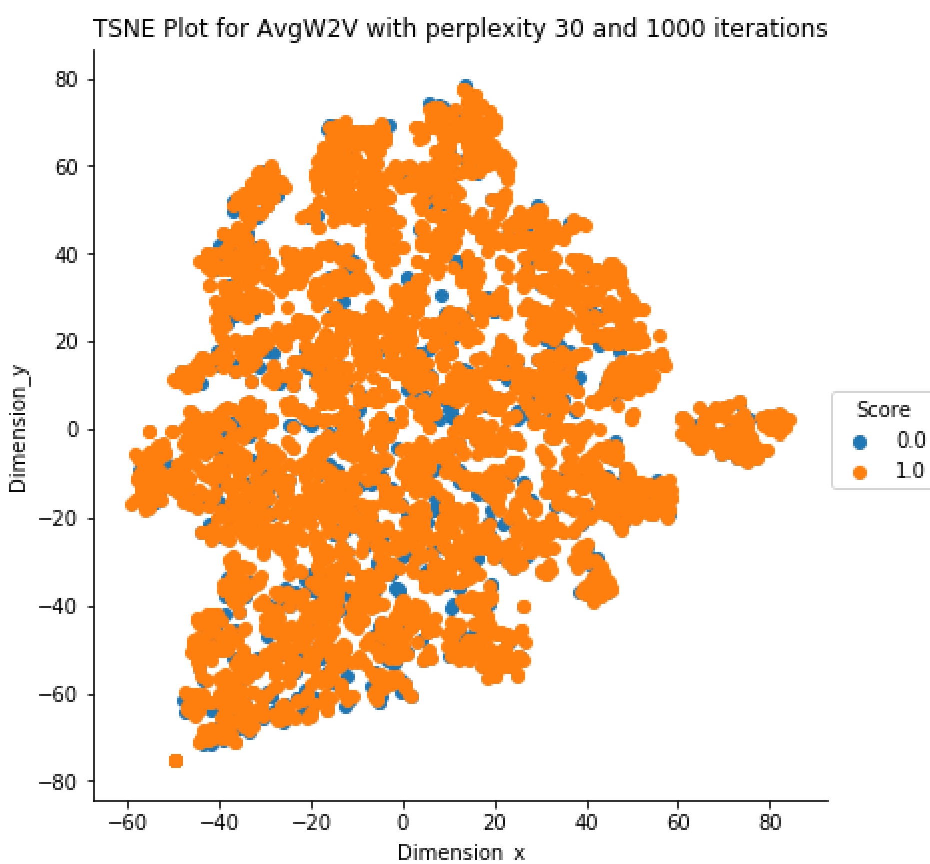
In [71]:
```
for_tsne_df = pd.DataFrame(data=complte_avgw2v, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for AvgW2V with perplexity 30 and 700 iterations')
plt.show()
```



TSNE Plot for AvgW2V with perplexity 30 and 1000 iterations

In [72]:
```
srcmatrix=np.asarray(sent_vectors)
print(srcmatrix[:1])
tsne=TSNE(n_components=2, random_state=0,n_iter =1000)
avgw2v_tsne=tsne.fit_transform(srcmatrix)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_avgw2v=np.hstack((avgw2v_tsne,y))
print(complte_avgw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_avgw2v, columns=['Dimension_x','Dimens
ion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for AvgW2V with perplexity 30 and 1000 iterations')
plt.show()
```

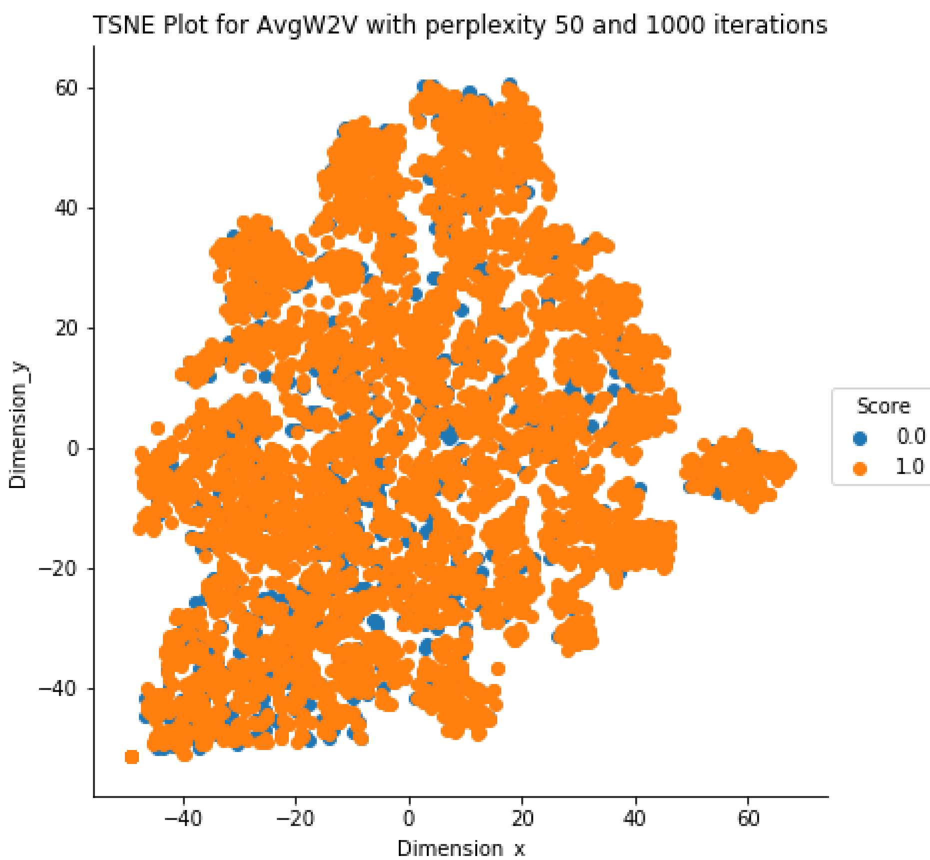
```
[[-1.05615837 -0.36980339  0.49330873 -0.27411663 -0.56960896 -0.14927392
  -0.14618293  0.06495207 -0.15155725 -0.08218197  0.16597637  0.04615958
   0.24062717  0.13274717 -0.18382477 -0.751844    0.02176288 -0.17255606
  -0.32124375  0.07122388 -0.00699232  0.11709234 -0.00143706 -0.42582252
  -0.21435779  0.16394789  0.23077574  0.32471395  0.22027823 -0.36286478
  -0.78164459 -0.16755536 -0.08504115  0.28844828  0.41093772 -0.29634169
   0.49568507 -0.03577016  0.31576719  0.34727043 -0.4082306   0.12498265
   0.2131561  -0.22000687  0.06597471  0.05412627  0.64810923  0.03436346
   0.10032199  0.11607234]]
(4986, 1)
(4986, 3)
```



TSNE Plot for AvgW2V with perplexity 50 and 1000 iterations

In [73]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1000,perplexity=50)
avgw2v_tsne=tsne.fit_transform(srcmatrix)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_avgw2v=np.hstack((avgw2v_tsne,y))
print(complte_avgw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_avgw2v, columns=['Dimension_x','Dimens
ion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for AvgW2V with perplexity 50 and 1000 iterations')
plt.show()
```

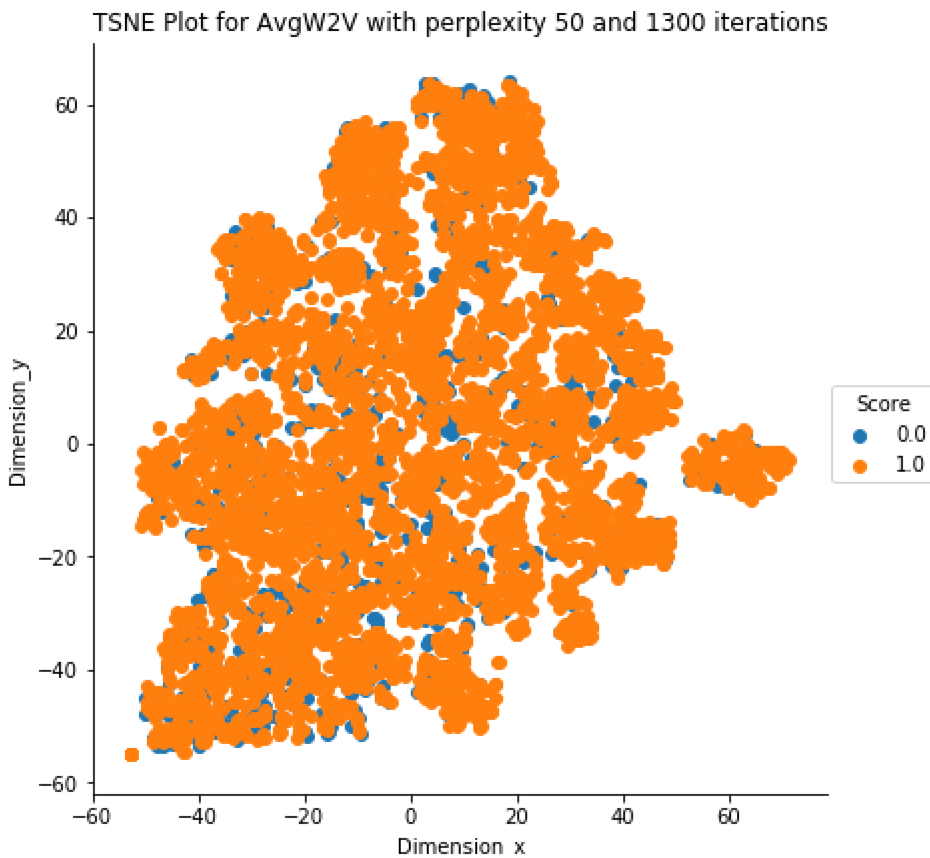
```
(4986, 1)
(4986, 3)
```



TSNE Plot for AvgW2V with perplexity 50 and 1300 iterations

In [74]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1300,perplexity=50)
avgw2v_tsne=tsne.fit_transform(srcmatrix)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_avgw2v=np.hstack((avgw2v_tsne,y))
print(complte_avgw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_avgw2v, columns=['Dimension_x','Dimens
ion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for AvgW2V with perplexity 50 and 1300 iterations')
plt.show()
```

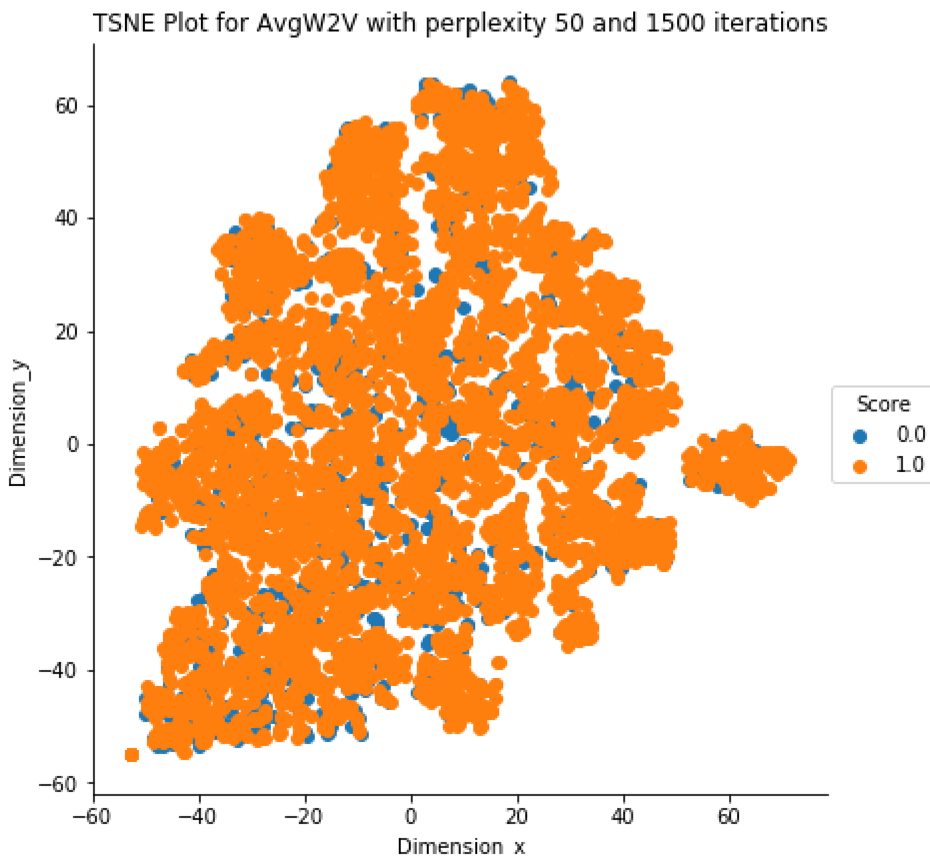
```
(4986, 1)
(4986, 3)
```



TSNE Plot for AvgW2V with perplexity 50 and 1500 iterations

In [75]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1300,perplexity=50)
avgw2v_tsne=tsne.fit_transform(srcmatrix)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_avgw2v=np.hstack((avgw2v_tsne,y))
print(complte_avgw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_avgw2v, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for AvgW2V with perplexity 50 and 1500 iterations')
plt.show()
```

```
(4986, 1)
(4986, 3)
```



Conclusion Applying TNSE on Text Avg W2V vectors

Both 0 and 1 datapoints seems to be highly overlaping suggesting there may be highly cluttered data in higher dimensions too

And Not able to classify the Score using TFIDF vector representation on reviews and applying TSNE on the same as there is no linear seperation as data is highly ovelaps=ing

The shape seems to be stable in 700-1500 iterations

Not change in the shape when there is a change in perplexity

[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [0]:
```
# please write all the code with proper documentation, and proper titles for e
ach subsection
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the rea
der
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

calculating TFIDF

In [76]:
```
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

Calculating TFIDF weighted W2V vectors using already trained w2v model for avg W2v above

In [77]:
```
tfidf_feat = model.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in tqdm(list_of_sentance):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            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
```

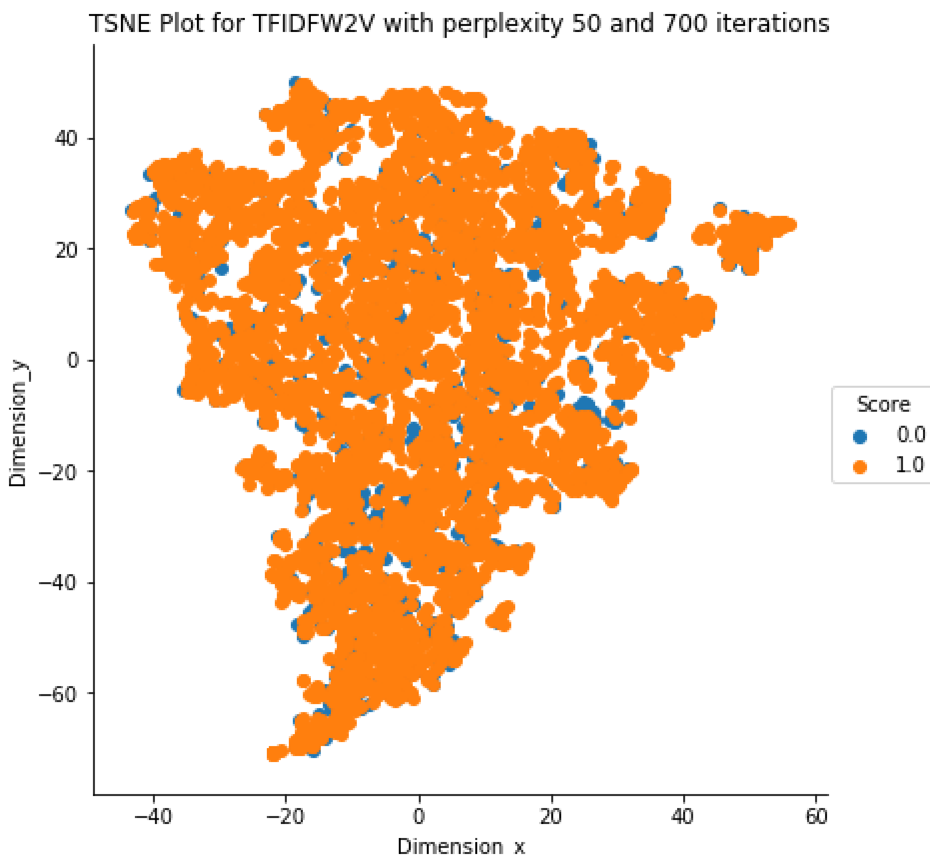
```
100%|████████████████████████████████████████████████████████████████████████
█████| 4986/4986 [00:19<00:00, 253.99it/s]
```

In [ ]:
```
#print(type(tfidf_sent_vectors))
print(tfidf_sent_vectors[:1])
print(len(tfidf_sent_vectors))
tfidf_sent_vectors_array=np.asarray(tfidf_sent_vectors)
print(type(tfidf_sent_vectors_array))
print(tfidf_sent_vectors_array[:1])
print(tfidf_sent_vectors_array.shape)
```

TSNE Plot for TFIDFW2V with perplexity 50 and 700 iteration

In [83]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =700,perplexity=50)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dime
nsion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 50 and 700 iterations')
plt.show()
```

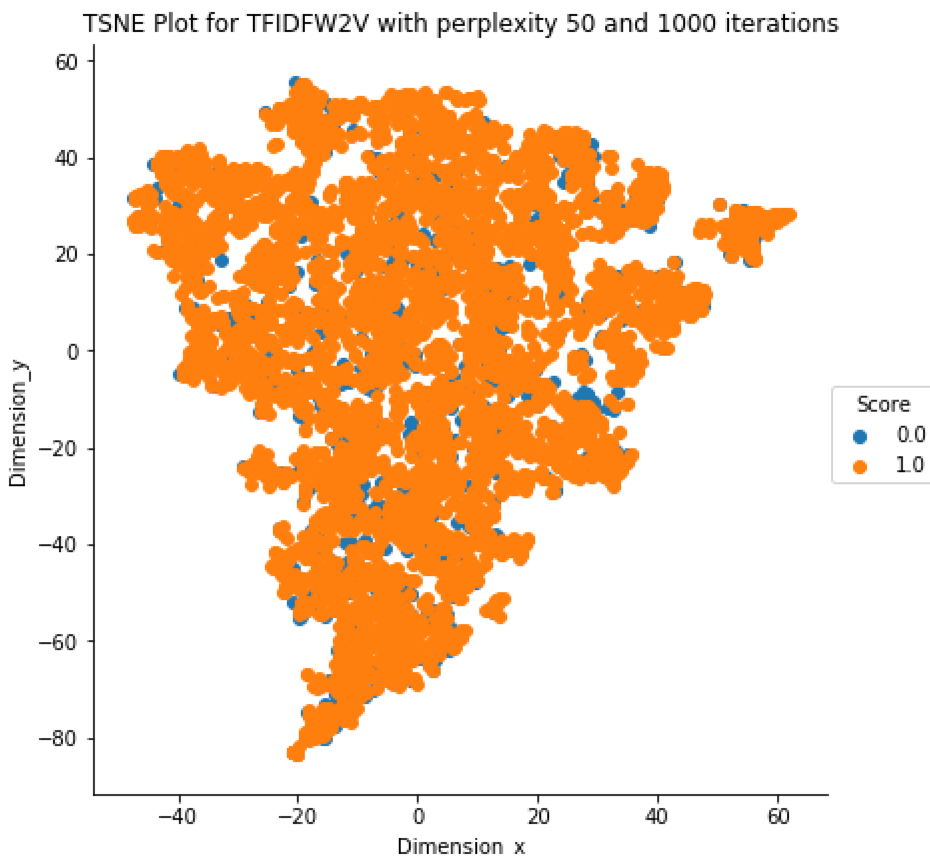
```
(4986, 1)
(4986, 3)
```



TSNE Plot for TFIDFW2V with perplexity 50 and 1000 iteration

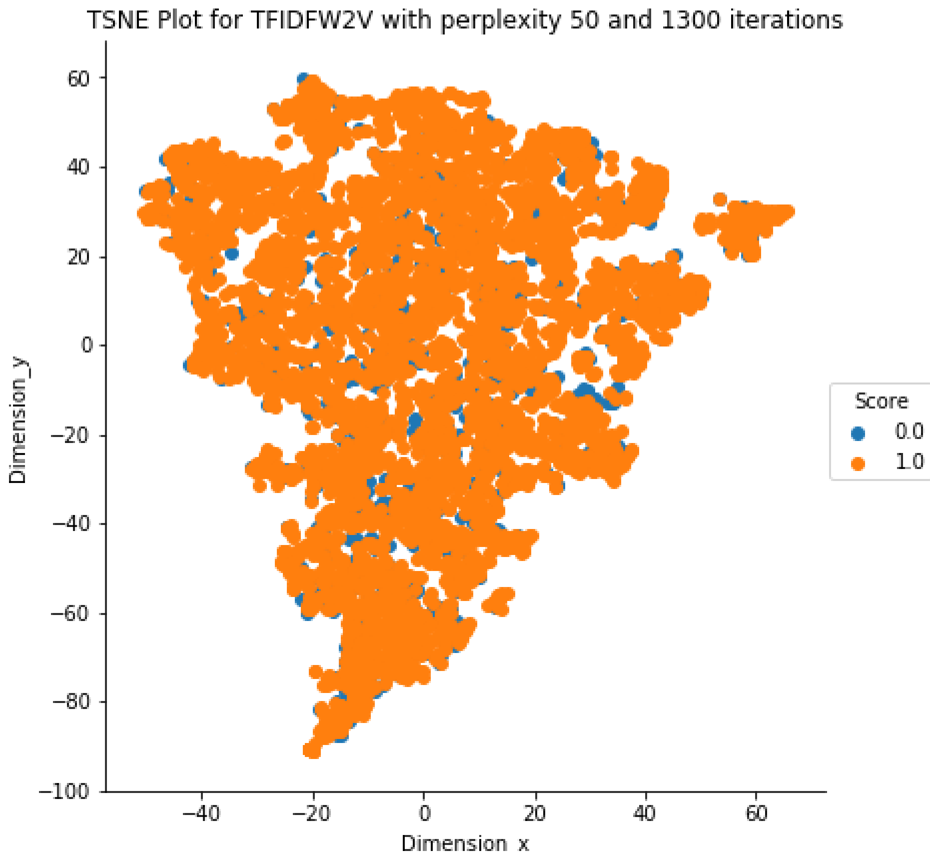
In [84]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1000,perplexity=50)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 50 and 1000 iterations')
plt.show()
```

```
(4986, 1)
(4986, 3)
```



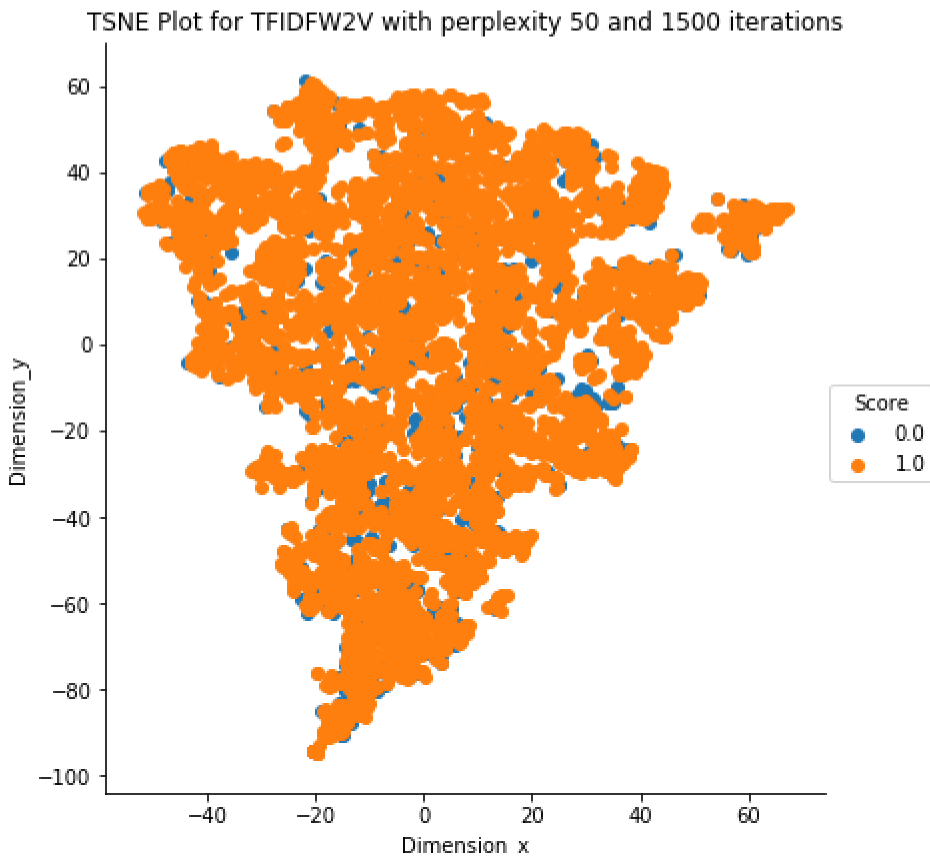
TSNE Plot for TFIDFW2V with perplexity 50 and 1300 iteration

In [85]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1300,perplexity=50)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
#print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
#print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dimension_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x', 'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 50 and 1300 iterations')
plt.show()
```



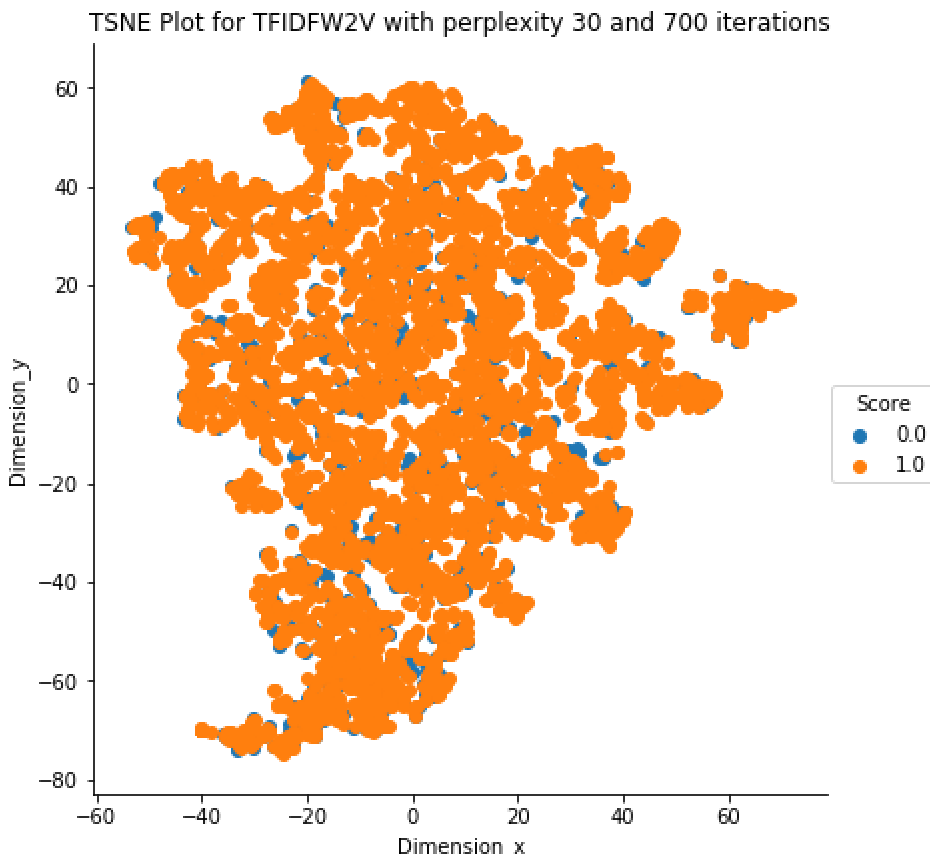
TSNE Plot for TFIDFW2V with perplexity 50 and 1500 iteration

In [86]:
```
tsne=TSNE(n_components=2, random_state=0,n_iter =1500,perplexity=50)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
#print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
#print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dime
nsion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 50 and 1500 iterations')
plt.show()
```



TSNE Plot for TFIDFW2V with perplexity 30 and 700 iterations

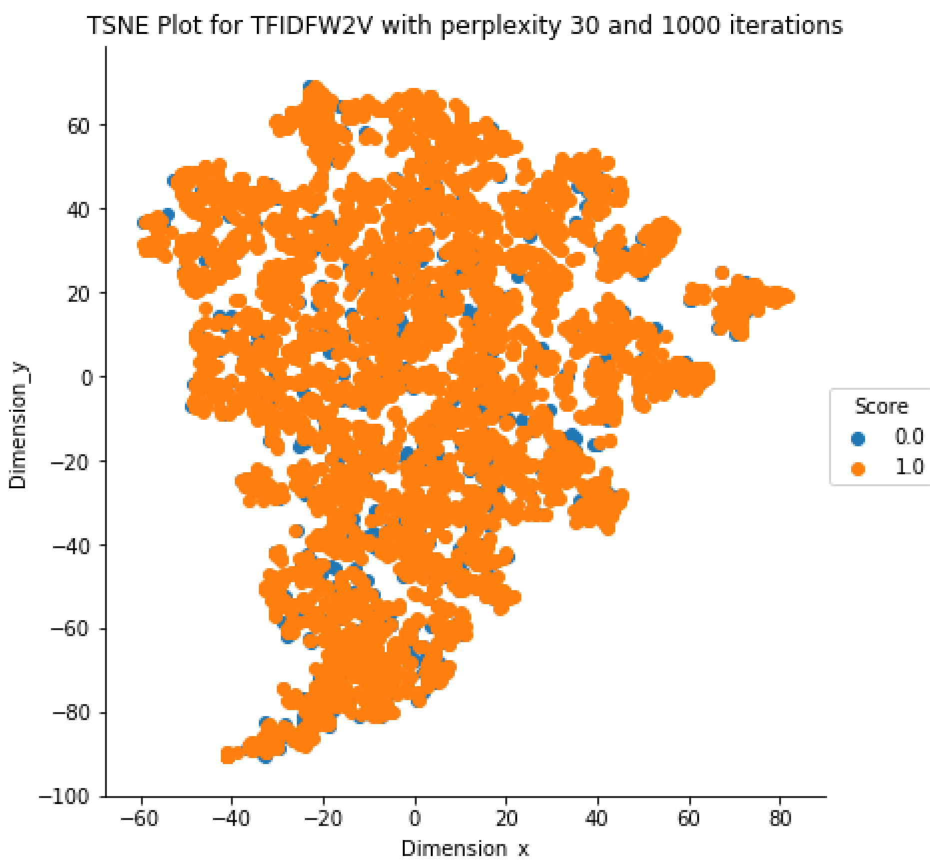
In [89]:
```
tsne=TSNE(n_components=2,random_state=0,n_iter=700,perplexity=30)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
#print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
#print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dime
nsion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 30 and 700 iterations')
plt.show()
```



TSNE Plot for TFIDFW2V with perplexity 30 and 1000 iterations

In [90]:
```
tsne=TSNE(n_components=2,random_state=0,n_iter=1000,perplexity=30)
TFIDF_W2V=tsne.fit_transform(tfidf_sent_vectors_array)
y=final["Score"]
y=y.as_matrix()
y=y.reshape(-1,1)
#print(y.shape)
complte_tfidfw2v=np.hstack((TFIDF_W2V,y))
print(complte_tfidfw2v.shape)
for_tsne_df = pd.DataFrame(data=complte_tfidfw2v, columns=['Dimension_x','Dime
nsion_y','Score'])
sn.FacetGrid(for_tsne_df, hue="Score", size=6).map(plt.scatter, 'Dimension_x',
'Dimension_y').add_legend()
plt.title('TSNE Plot for TFIDFW2V with perplexity 30 and 1000 iterations')
plt.show()
```

```
(4986, 3)
```



Conclusions TFIDF weighted W2v vectors

Data points of Both +ve&-ve reviews / 0 & 1 class labels extremely overlap in TSNE Represented 2D space we may infer that the points may overlap in the actual higher dimensions

Rerunning for multiple iteratios from 700-1500 and for perplexities of 30 and 50 the shape remains to be same

[6] Conclusions

Applying various text to vector conversion methods from BOW,TFIDF,avgW2V,TfIdfW2V and visulaising theese high dimensional vectors by using TSNE to transform them to two dimensions it can be observed the data points belonging to each class label is highly overlaped in TSNE 2D space suggesting the datapoints may be overlaped in the original high dimensional space

Since the data is highly overlaped we could not observe any linear or visual demarcation or possiballity for classification of the classlabels

the TSNE plots in general maintained stable shapes arount 1000-1500 iterations and the general shapes remained same for perplexities of 30 and 50