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 [37]:

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



import sqlite3
import pandas as pd
import numpy as np
import nltk
```

```python
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
from sklearn.preprocessing import StandardScaler
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]:

```python
# 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 500
000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1000
0""", con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rat
ing.
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)
print(actualScore.shape)
```

```
Number of data points in our data (10000, 10)
(10000,)
```

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)
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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']
```

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

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()
```

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

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

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

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

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

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

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

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False
, 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
```

```
(9564, 10)
```

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

```
95.64
```

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()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for Son at College

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure co taste wi crunchy almond: inside

In [12]:

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

In [13]:

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

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

```
(9564, 10)


1    7976
0    1588
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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
15 month old loves to eat them on the go! They seem great for a healthy, quick, and easy snack!
==================================================
These chips are truly amazing. They have it all. They're light, crisp, great tasting, nice texture, AND they're all natura
l... AND low in fat and sodium! Need I say more? I recently bought a bag of them at a regular grocery store, and couldn't be
live my taste buds. That's why I excited why I saw them here on Amazon, and decided to buy a case!
==================================================
```

```
These tablets definitely made things sweeter -- like lemons, limes, and grapefruit.  But it wasn't to the point of sheer ama
zement.  They also had an interesting effect on cheeses and vinegar, but still did virtually nothing for beer and wine. The
tablets are a bit pricey but they do work.  If you've got extra money, sure, give them a try, but if you're looking for some
amazing way to get your kids to eat broccoli or something along those lines then this is not the answer. Fun experiment, but
not life-changing. :)
==================================================
```

In [15]:

```python
# 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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-ta
gs-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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
15 month old loves to eat them on the go! They seem great for a healthy, quick, and easy snack!
==================================================
These chips are truly amazing. They have it all. They're light, crisp, great tasting, nice texture, AND they're all natura
l... AND low in fat and sodium! Need I say more? I recently bought a bag of them at a regular grocery store, and couldn't be
live my taste buds. That's why I excited why I saw them here on Amazon, and decided to buy a case!
==================================================
These tablets definitely made things sweeter -- like lemons, limes, and grapefruit.  But it wasn't to the point of sheer ama
zement.  They also had an interesting effect on cheeses and vinegar, but still did virtually nothing for beer and wine. The
tablets are a bit pricey but they do work.  If you've got extra money, sure, give them a try, but if you're looking for some
amazing way to get your kids to eat broccoli or something along those lines then this is not the answer. Fun experiment, but
not life-changing. :)
```

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
```

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

In [18]:

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

```
These chips are truly amazing. They have it all. They are light, crisp, great tasting, nice texture, AND they are all natura
l... AND low in fat and sodium! Need I say more? I recently bought a bag of them at a regular grocery store, and could not b
elive my taste buds. That is why I excited why I saw them here on Amazon, and decided to buy a case!
==================================================
```

In [19]:

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

```
We have used the Victor fly bait for  seasons.  Can't beat it.  Great product!
```

In [20]:

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

```
These chips are truly amazing They have it all They are light crisp great tasting nice texture AND they are all natural AND
low in fat and sodium Need I say more I recently bought a bag of them at a regular grocery store and could not belive my tas
te buds That is why I excited why I saw them here on Amazon and decided to buy a case
```

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', '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', 'ha
d', '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", 'no
w', '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 [22]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopword
s)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 9564/9564 [00:03<00:00, 2836.82it/s]
```

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

```
'chips truly amazing light crisp great tasting nice texture natural low fat sodium need say recently bought bag regular groc
ery store could not belive taste buds excited saw amazon decided buy case'
```

[3.2] Preprocess Summary

In [24]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

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

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

```
some feature names  ['aa', 'aaaa', 'aahhhs', 'ab', 'aback', 'abandon', 'abates', 'abberline', 'abbott', 'abby']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (9564, 18244)
the number of unique words  18244
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/module
```

```
s/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=10000)
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_co
unts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (9564, 5765)
the number of unique words including both unigrams and bigrams  5765
```

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()
[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.ge
t_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able eat', 'able find', 'able get', 'able
order', 'able use', 'absolute', 'absolute best']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text TFIDF vectorizer  (9564, 5765)
the number of unique words including both unigrams and bigrams  5765
```

[4.4] Word2Vec

In [28]:

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

In [29]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True
```

```python
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
 train your own w2v ")
```

```
[('excellent', 0.9054721593856812), ('good', 0.8834060430526733), ('looking', 0.7835540771484375), ('overall', 0.7798551917076111), ('quick', 0.7764415144920349), ('wonderful', 0.7751519680023193), ('presentation', 0.7612698674201965), ('super', 0.7447444200515747), ('especially', 0.7387913465499878), ('sincerely', 0.7373694181442261)]
==================================================
[('hands', 0.9863061904907227), ('consumed', 0.9860837459564209), ('disappointing', 0.9857912063598633), ('gourmet', 0.9824321866035461), ('sweetest', 0.9817327260971069), ('greatest', 0.9817072153091431), ('kettle', 0.9816833734512329), ('consistent', 0.9815331101417542), ('realizing', 0.9805448055267334), ('jamaica', 0.9804708361625671)]
```

In [30]:

```python
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  5652
sample words  ['youngest', 'inside', 'hey', 'tast', 'newmans', 'mg', 'restricted', 'push', 'year', 'dishes', 'optimal', 'stuffers', 'residue', 'asks', 'buzz', 'sorrento', 'oregon', 'growing', 'begin', 'pro', 'say', 'rushed', 'craving', 'terrific',
```

```
'wegman', 'language', 'pecans', 'cow', 'coating', 'unlike', 'happened', 'periods', 'crema', 'celebrate', 'adapt', 'double', 'testing', 'fermentation', 'anticipated', 'comparison', 'served', 'possibly', 'advantage', 'coupon', 'tact', 'usual', 'plock
ys', 'torn', 'run', 'goodies']
```

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

[4.4.1.1] Avg W2v

In [31]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to c
hange 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%|██████████| 9564/9564 [00:25<00:00, 379.87it/s]

9564
```

```
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# 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 [33]:

```
# 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%|██████████| 9564/9564 [01:44<00:00, 91.72it/s]
```

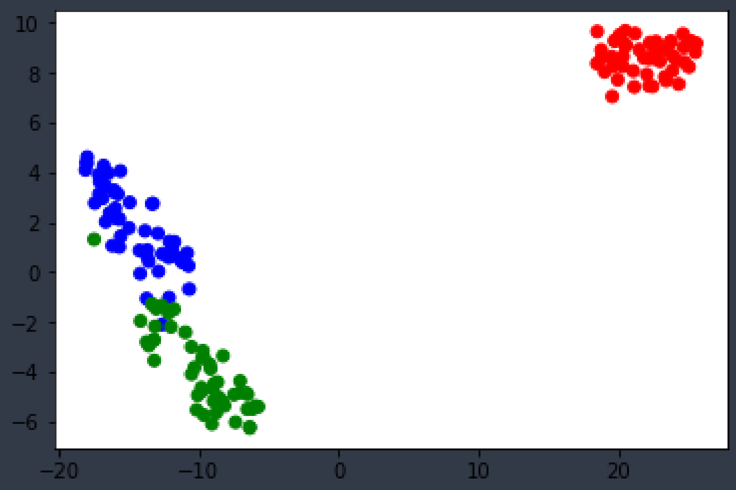
[5] Applying TSNE

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 [34]:

```
# https://github.com/pavlin-policar/fastTSNE you can try this also, this version is litt
le 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.toar
ray()) , .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['Scor
e'].apply(lambda x: colors[x]))
plt.show()
```



[5.1] Applying TNSE on Text BOW vectors

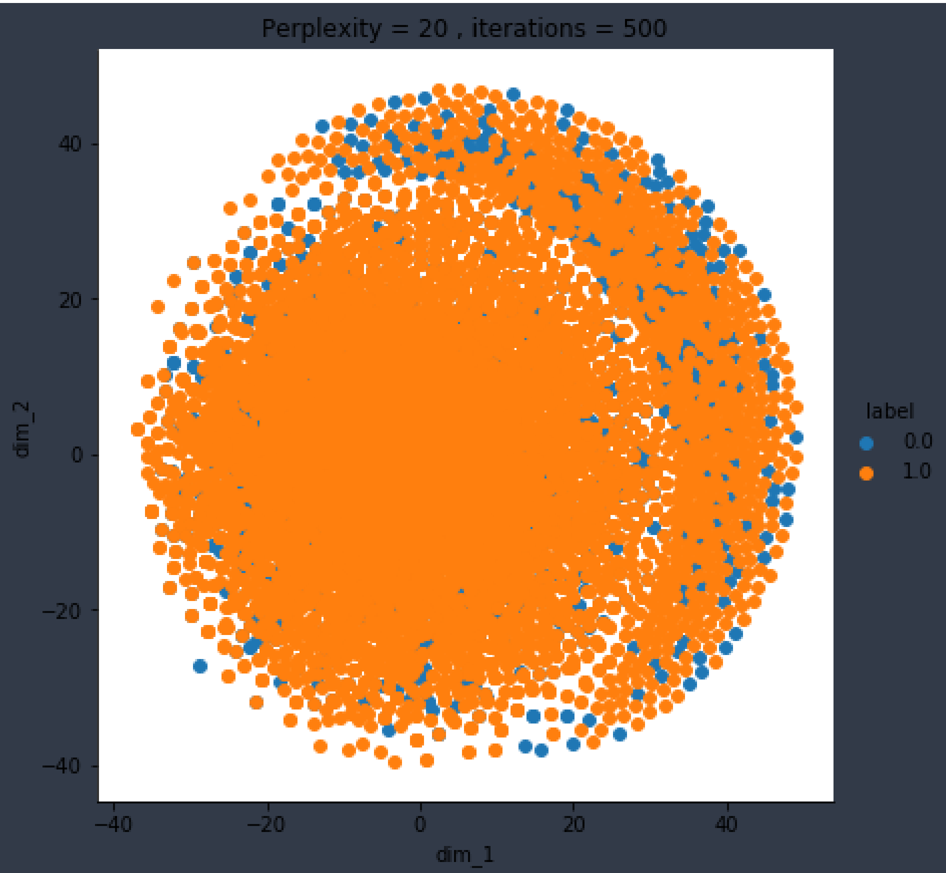
Instruction

please write all the code with proper documentation, and proper titles for each subsection 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

TSNE on Text BOW vectors with perplexity=20 , iterations=500 and working on 10k data set

In [52]:

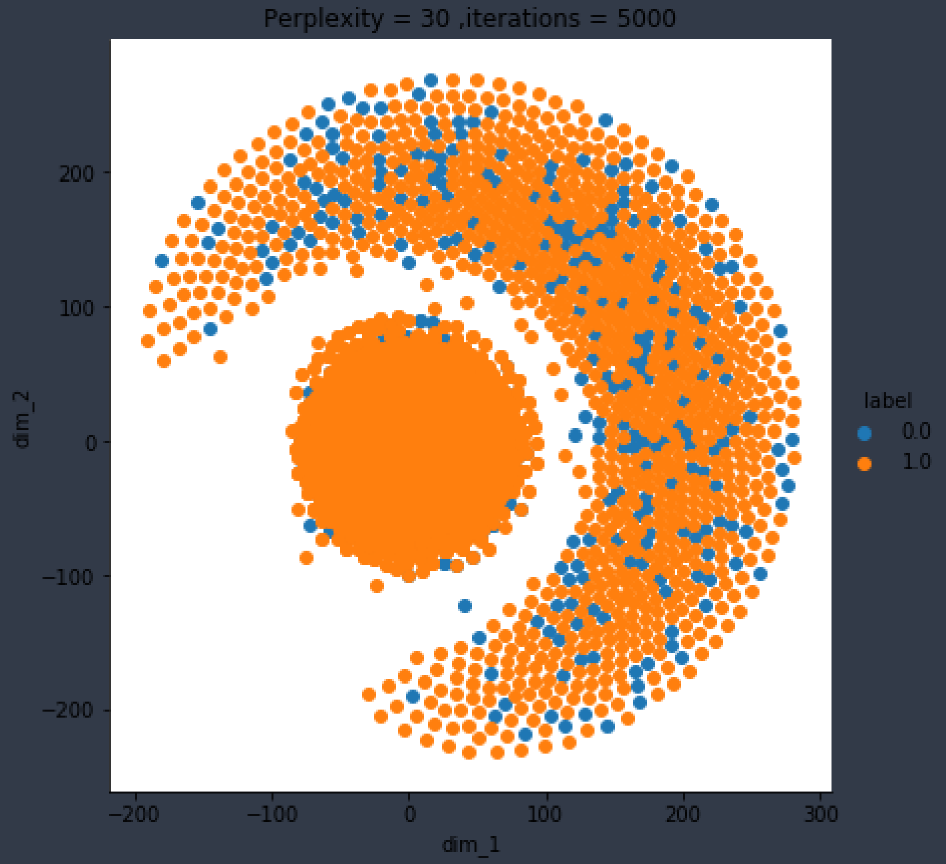
```
from sklearn.preprocessing import StandardScaler

model = TSNE(n_components=2,perplexity=20,n_iter=500,random_state=0)
# converting sparse to a dense array
final_count= final_counts.toarray()
# standardizing our data making mean=0 and std_dev=1
standardize = StandardScaler().fit_transform(final_count)
tsne_data = model.fit_transform(standardize)
#creating a new data which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# Plotting the result of tsne
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 20 , iterations = 500')
plt.show()
```



TSNE on TEXT BOW vectors with perplexity=30 ,iterations=5000 and working on 10k data set

In [53]:
```
model = TSNE(n_components=2,perplexity=30,n_iter=5000,random_state=0)
tsne_data = model.fit_transform(standardize)
#creating a new data which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
```

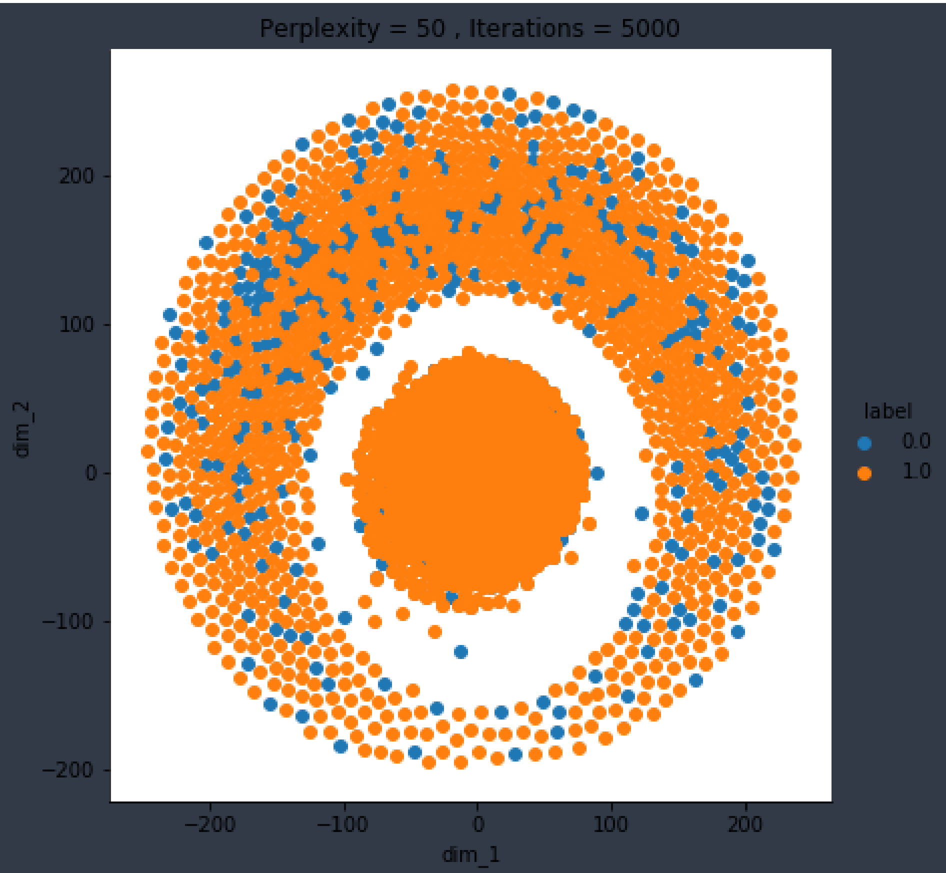
```
# Plotting our result
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 30 ,iterations = 5000')
plt.show()
```



TSNE on TEXT BOW vectors with perplexity=50 ,iterations=5000 and working on 10k data set

In [54]:

```
model = TSNE(n_components=2,perplexity=50,n_iter=5000,random_state=0)
tsne_data = model.fit_transform(standardize)
#creating a new data which help us in plotting the result data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# plotting our reult
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 50 , Iterations = 5000')
plt.show()
```



Observation

From the above plots we can see a very dense cluster of positive reviews and another outer circle shaped plot which is dense cluster of postive and negative point although there are both negative and postive in the middle dense cluster too so we can say that we cannot make any conclusions or see any seperation in our BOW TSNE plot

[5.1] Applying TNSE on Text TFIDF vectors

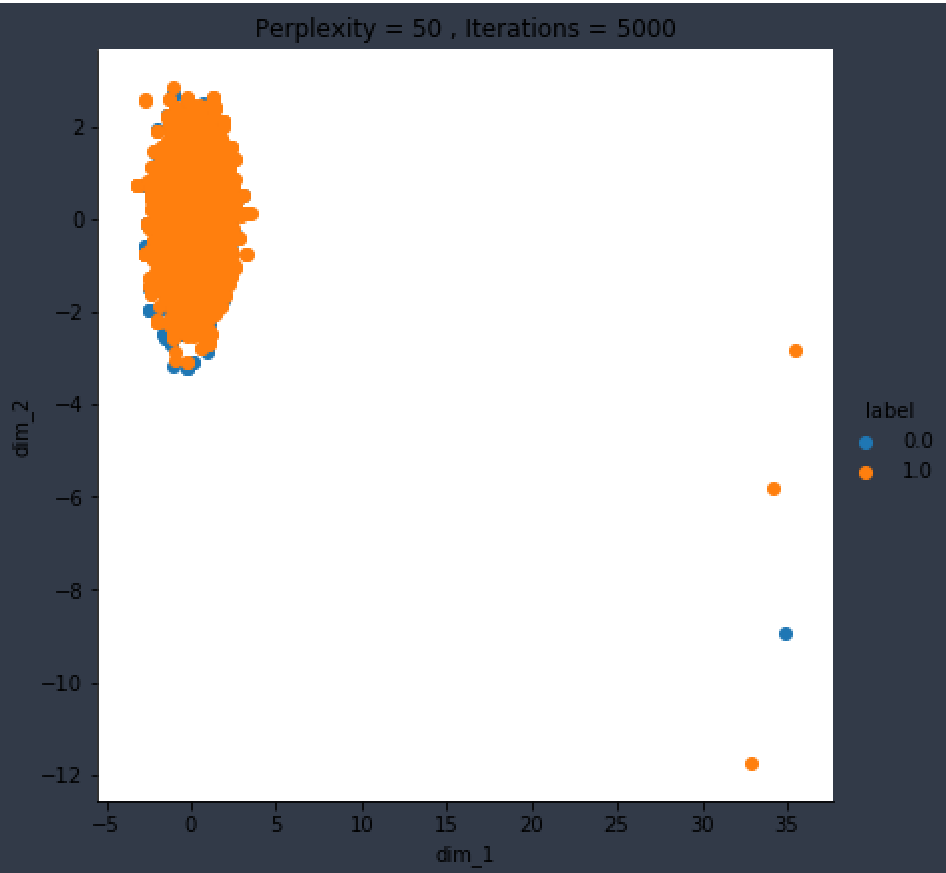
Instructions

please write all the code with proper documentation, and proper titles for each subsection 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

TSNE on Text TFIDF vectors with Perplexity = 50 , Iterations= 5000 and working on a 10K data set

In [56]:

```
model = TSNE(n_components=2,perplexity=50,n_iter=5000,random_state=0)
# converting sparse to dense array
final_count= final_tf_idf.toarray()
# standardizing our data
standardize = StandardScaler().fit_transform(final_count)
tsne_data = model.fit_transform(standardize)
#creating a new data which helps us in plotting a new data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 50 , Iterations = 5000')
plt.show()
```

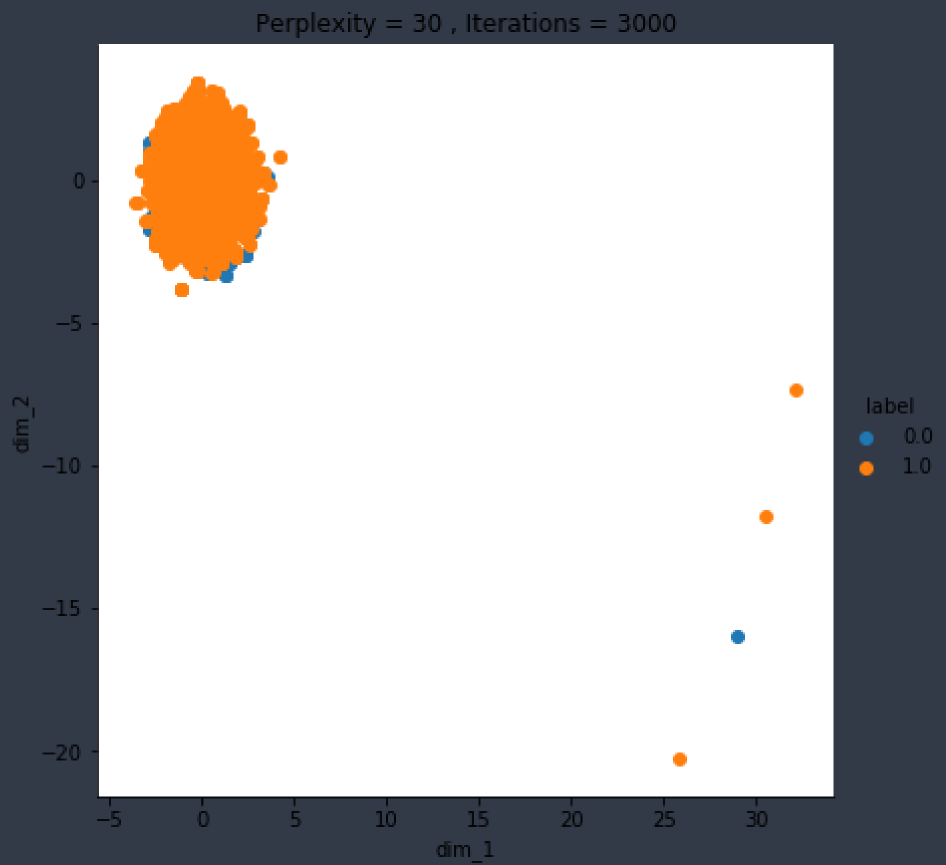


TSNE on Text TFIDF vectors with perplexity = 30 , Iterations = 3000 and working on a 10k data set

In [57]:

```
model = TSNE(n_components=2,n_iter=3000,random_state=0)
tsne_data = model.fit_transform(standardize)
# creating a new data which helps us in plotting a new data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
```

```python
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 30 , Iterations = 3000')
plt.show()
```



Observation

From the above plots we can see that there is a dense cluster at the upper left portion of our plot which is a complete overlap of positive and negative points densely and there are outliers too in our plot at the right side which are completely away from our dense cluster , overall we can say TSNE on TFIDF vectors results in very poor plot and we cannot seperate our positive and negative class.

[5.3] Applying TNSE on Text avg W2V vectors

Instructions

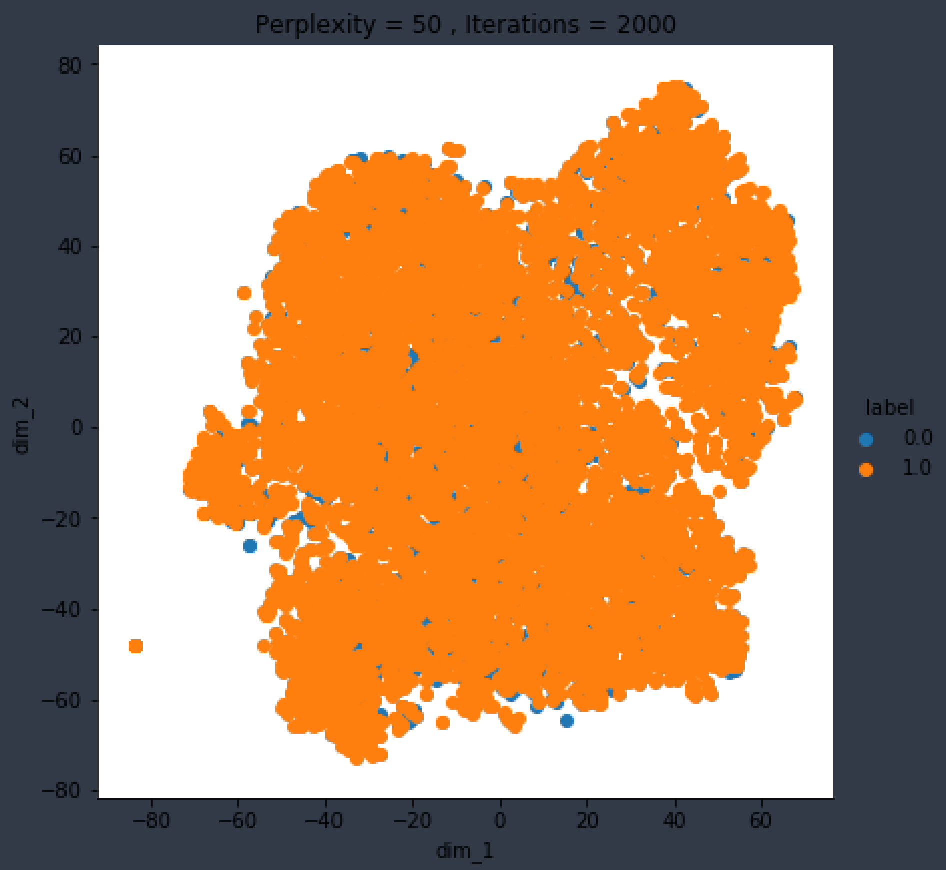
please write all the code with proper documentation, and proper titles for each subsection when you plot any graph make sure you use a. Title, that describes your plot, this will be very helpful to the reader b. Legends if needed c. X-axis label d. Y-axis label

TSNE on Text avg W2V vectors with Perplexity = 50 , Iterations = 2000 and working on 10k data set

In [35]:

```
model = TSNE(n_components=2,perplexity=50,n_iter=2000,random_state=0)
sent_vector=np.asarray(sent_vectors)
# Standardizing our data
standardize = StandardScaler().fit_transform(sent_vector)
tsne_data = model.fit_transform(standardize)
# Creating a new data which helps us in plotting a new data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 50 , Iterations = 2000')
plt.show()
```

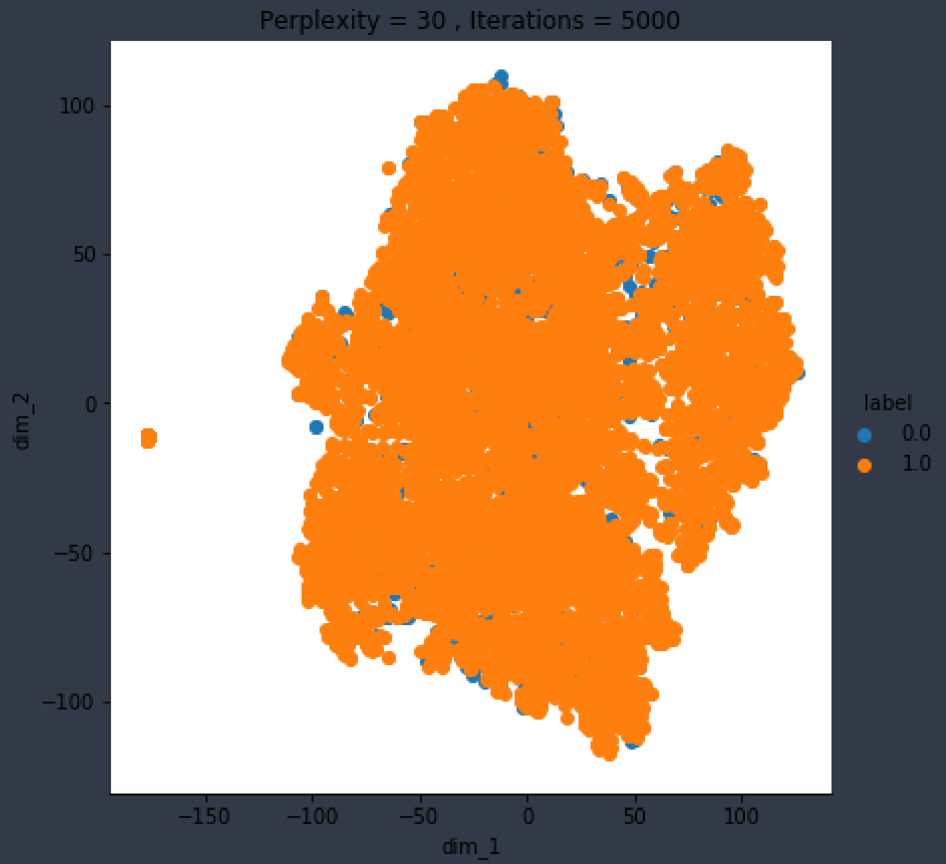
```
/usr/local/lib/python3.5/dist-packages/seaborn/axisgrid.py:230: UserWarning: The `size` paramter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)
```



TSNE on Text avg W2V vectors with Perplexity = 50 , Iterations = 2000 and working on 10k data set

In [38]:
```
model = TSNE(n_components=2,perplexity=30,n_iter=5000,random_state=0)
tsne_data = model.fit_transform(standardize)
# Creating a new data which helps us in plotting a new data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
```

```
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 30 , Iterations = 5000')
plt.show()
```



Observation

From the above plots we cannot make any conclusion cause all we can see is a wide dense cluster of overlapping positive and negative points so we can say we cannot see any seperation of positive and negative classes on TSNE plot of Avg W2V model

[5.4] Applying TNSE on Text TFIDF weighted W2V vectors

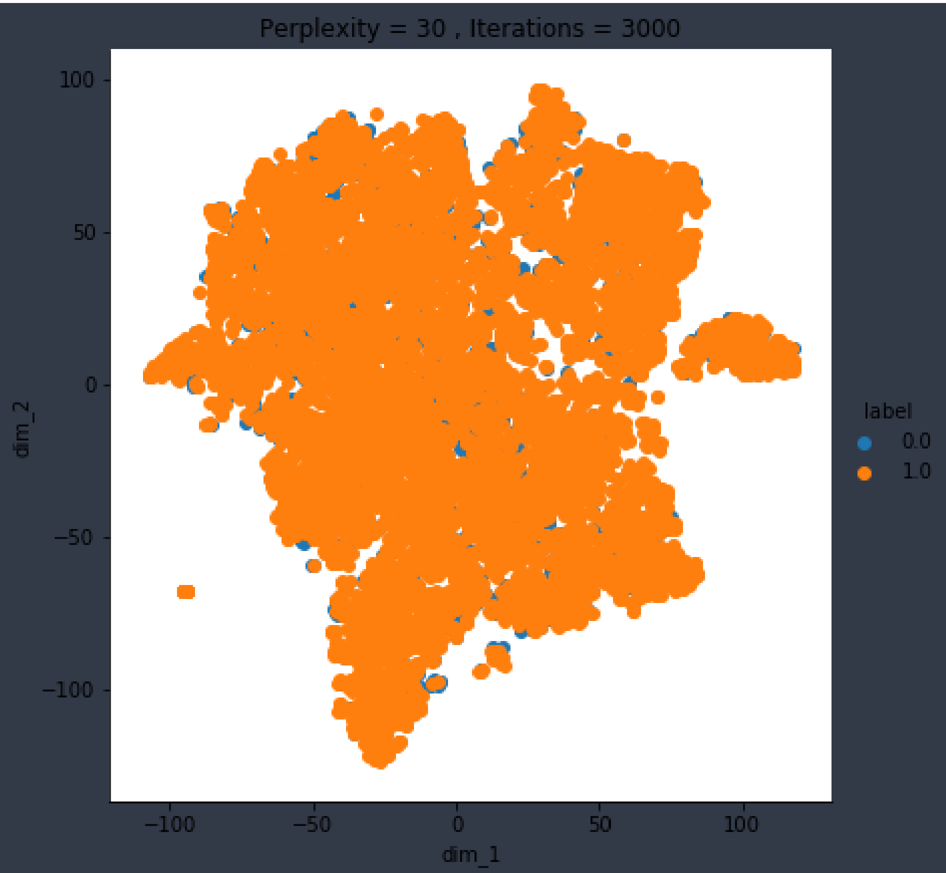
Instructions

please write all the code with proper documentation, and proper titles for each subsection 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

TSNE on Text TFIDF weighted W2V vectors with Perplexity = 30 , Iterations = 3000 and working on 10k data set

In [39]:

```
model = TSNE(n_components=2,perplexity=30,n_iter=3000,random_state=0)
tfidf_sent_vectors=np.asarray(tfidf_sent_vectors)
# Standardizing our data
standardize = StandardScaler().fit_transform(tfidf_sent_vectors)
tsne_data = model.fit_transform(standardize)
# Creating a new data which helps us in plotting data
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 30 , Iterations = 3000')
plt.show()
```

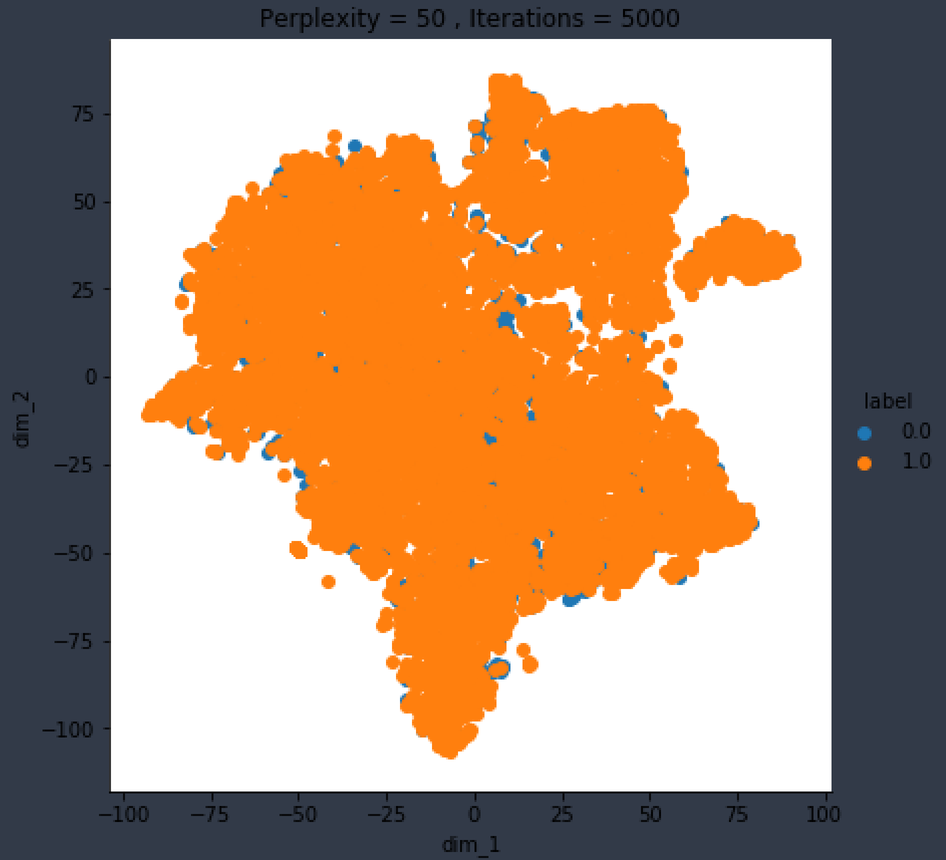


TSNE on Text TFIDF weighted W2V vectors with Perplexity = 30 , Iterations = 3000 and working on 10k data set

In [40]:

```
model = TSNE(n_components=2,perplexity=50,n_iter=5000,random_state=0)
tsne_data = model.fit_transform(standardize)
# Creating a new data set for plotting
tsne_data = np.vstack((tsne_data.T,final['Score'])).T
tsne_df = pd.DataFrame(data = tsne_data,columns=("dim_1","dim_2","label"))
```

```python
# Plotting our data
sns.FacetGrid(tsne_df,hue="label",size=6).map(plt.scatter,"dim_1","dim_2").add_legend()
plt.title('Perplexity = 50 , Iterations = 5000')
plt.show()
```



Observation

From the above plots we cannot make any conclusion cause all we can see is a wide dense cluster of overlapping positive and negative points so we can say we cannot see any seperation of positive and negative classes on TSNE plot of TFIDF Weighted W2V model

[6] Conclusions

From the above four models none of them can clearly tell us anything about the reviews the clusters are so dense and overlapping that we can't see any seperation in our data but in the Bag of words model we can see that their is dense circle in center where we can say that most of the positive points lie in that circle with some negative one and rest points at the outer incomplete circle which consists of both positive and negative points but still after plotiing all the models we cannot make any conclusion or we cannot seperate our points in any of the models just by watching it's TSNE plot