Amazon Finefood Reviews.

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

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:

```
Id
ProductId - unique identifier for the product
UserId - unqiue identifier for the user
ProfileName
HelpfulnessNumerator - number of users who found the review helpful
HelpfulnessDenominator - number of users who indicated whether they found the review
helpful or not
Score - rating between 1 and 5
Time - timestamp for the review
Summary - brief summary of the review
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 [3]:

```
%matplotlib inline

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
```

```python
# using the SQLite Table to read data.
con = sqlite3.connect(r'C:/Users/Swaroop/Desktop/Srikanth Reddy/database.sqlite')
```

In [4]:

```python
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3
""", con)


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

#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
```

In [37]:

```python
filtered_data.shape #looking at the number of attributes and size of the data
filtered_data.head()
```

Out[37]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	positive	13038624
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	negative	13469760
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	positive	12190176
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	negative	13079232
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	positive	13507776

In [5]:

```
filtered_data.shape #looking at the number of attributes and size of the data
```

Out[5]:

```
(525814, 10)
```

Exploratory Data Analysis

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 [6]:

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

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577(

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 [7]:

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

In [8]:

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

Out[8]:

```
(364173, 10)
```

In [9]:

```
#helpfulness denominator should be always greater or equal to helpful numerator.
import pandas as pd
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)
display
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892[illegible]
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883[illegible]

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 [ ]:

```
# create a variable final which has rows that has denominator >= numerator
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [10]:

```
#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()
```

```
(364171, 10)
```

Out[10]:

```
positive    307061
negative     57110
Name: Score, dtype: int64
```

Text Preprocessing: Stemming, stop-word removal and Lemmatization.

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]:

```
#Text Processing, removing HTML tags
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

stop = set(stopwords.words('english')) #set of stopwords
stop.remove('not')# we are removing the word 'not' for the stop word list.
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence) # 'sub' means substitute text inside <..> with a spac
e ' '.
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence) #substitute any ?,!,/ etc with  a space ''.
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('*************************************')
print(sno.stem('tasty')) # stem word for staty is tasti.
```

```
{"should've", 'yourself', 't', 'haven', 'the', 'themselves', 'on', 'y', 'here', 'until', 'for', 'r
e', "hadn't", 'as', 'its', 'me', 'll', 'that', 'do', 'some', 'am', "doesn't", "aren't", 'them',
'most', 'theirs', 'in', 'other', 'o', 'up', 'm', 'ours', 'with', 'whom', 'it', 'myself', 'they', '
only', 'should', 'i', 'above', 'was', 'before', 'during', "wasn't", 'needn', 'himself', 'we', 'her
s', 'mightn', 'an', 'below', 'each', "she's", 'down', 'against', 'why', 'few', 'further', 'if', 'w
hen', 'to', "you'd", "couldn't", 'from', "you've", 'by', "mightn't", 'having', 'because',
"you're", 'own', 'weren', 'won', 'into', 'under', "shan't", 'didn', 'through', 'did', 'between', '
being', "hasn't", 'shouldn', 'off', 'just', "isn't", "don't", 'itself', 'are', 've', "needn't", 'w
hich', 'no', 'while', 'out', 'how', 'hasn', "that'll", 'herself', 'ma', 'of', 'can', "wouldn't", "
didn't", 'again', 'both', 'be', 'yours', 'd', 'doing', 'wasn', 'those', 'more', 'does', 'too', 'hi
m', 'were', 'all', 'what', 'their', 'about', "won't", 'don', 'this', 'hadn', "mustn't", 'he', 'the
se', 'your', 'had', 'has', 'been', 'such', "weren't", 'ourselves', 'my', 'any', 'isn', 'aren', 'mu
stn', 'than', 'or', 'her', 'who', 'have', 'where', 'our', 'you', 'a', 'over', 'at', 'ain', 'she',
'doesn', 'once', 'will', "it's", 'there', 's', 'shan', 'then', 'couldn', 'now', "you'll", 'very',
'his', 'is', 'and', "shouldn't", 'yourselves', 'wouldn', 'but', 'after', "haven't", 'so', 'same',
'nor'}
```

```
************************************
tasti
```

In [15]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values: # for each review in Text column
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop): #conver to lowe case
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')  # we are doing stemming in this line.
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
final['CleanedText']=final_string #adding a new column named CleanedText which displays the data after pre-processing of the review

    # In this whole code after clearing HTML tags, cleaninng puncutaions, alphanumeric words, and after checkingif a word
    #lenght greater than two, converitng each word to lower case and applying stemming ...etc, for all the reviews(rows),
    #we will get set of words which will be joined to form a sentence(filtered_entence), the text in filtered sentence will
    # not make any sence to read(e.g. Pasta tasy amazing feeling great time). Now we will create a new column named cleaned text
    #and store all the filteded sentence for each review in it on which we will finally apply BoW, tf-idf, etc.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

In [16]:

```
final_string #below the processed review can be seen in the CleanedText Column
final.head(3)
```

Out[16]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
138706	150524	0006641040	ACITT7DI6IDDL	zychinski	0	0	positive	93
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive	11
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	positive	11

In [17]:

```
# store final table into an SQlLite table for future.
conn = sqlite3.connect('C:/Users/Swaroop/Desktop/Srikanth Reddy/final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, index_label=None, chun
ksize=None, dtype=None)
```

In [18]:

```
# now accesing the new database for performing Bow...etc.
conn = sqlite3.connect('C:/Users/Swaroop/Desktop/Srikanth Reddy/final.sqlite')
final = pd.read_sql_query("""
SELECT *
FROM Reviews
""", conn)
```

Bag of Words(BoW)

In [19]:

```
#BoW, for each review Ri, we have to compute a vector Vi.
count_vect = CountVectorizer() #in scikit-learn
final_counts = count_vect.fit_transform(final['CleanedText'].values)
final_counts.get_shape()
```

Out[19]:

```
(364173, 71626)
```

In [20]:

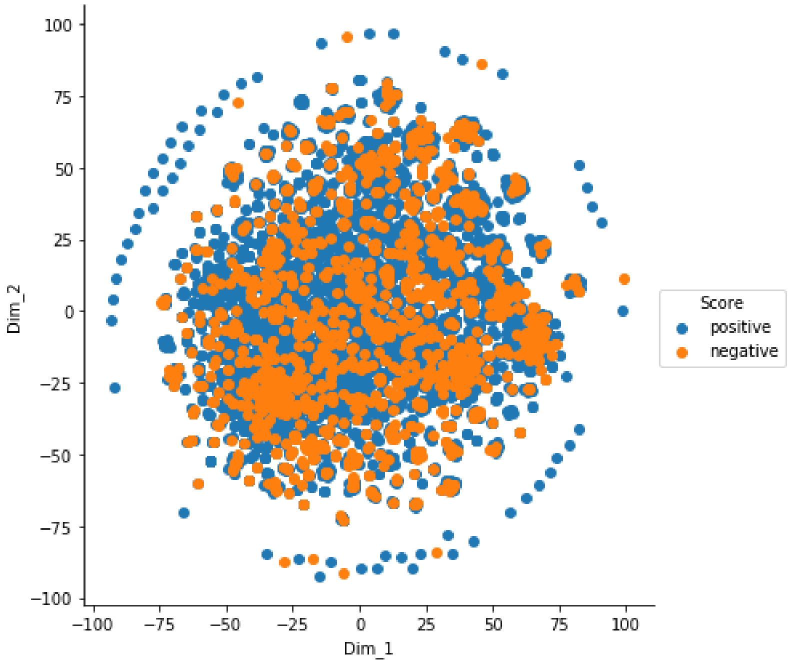
```
#Applying TSNE to this Bow Representation
# Standardization of Data
from sklearn.preprocessing import MaxAbsScaler

standardized_data = MaxAbsScaler().fit_transform(final_counts)
data = standardized_data[0:10000,0:10000].toarray()
```

In [21]:

```
#The labes are scores(positive/negative)
label = final.Score[0:10000]
```

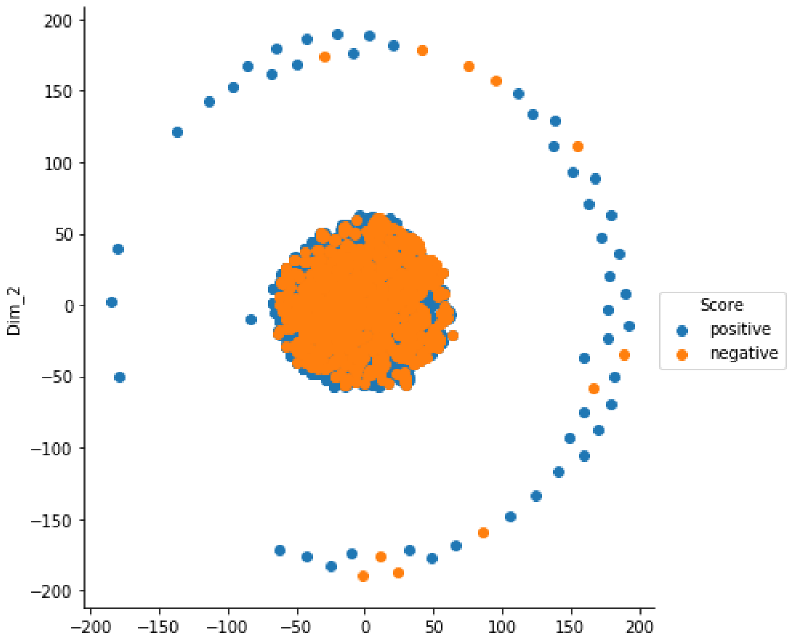
In [22]:

```
#T-SNE on Bag of Word
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



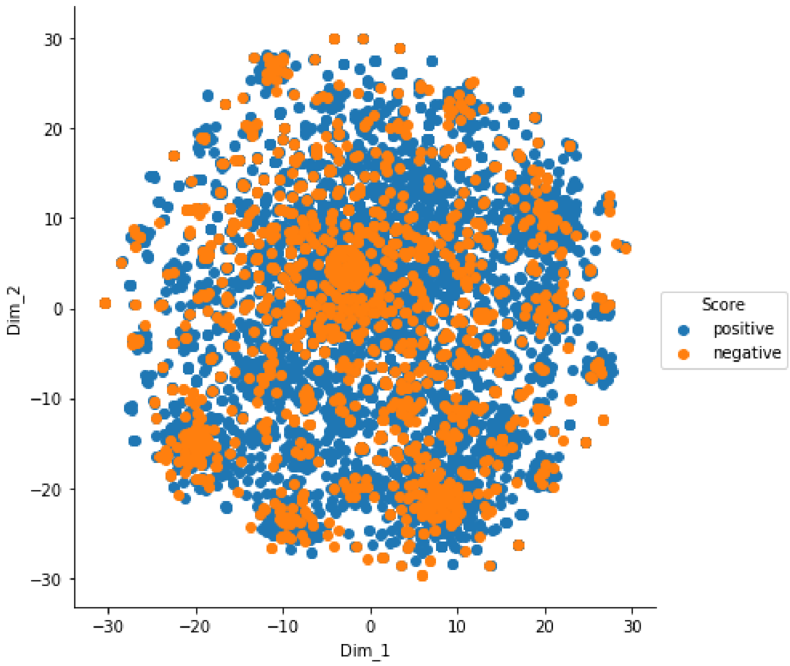
In [23]:

```
# Plotting with different perplexity and iterations.
model = TSNE(n_components=2, random_state=0, perplexity=100, n_iter=5000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



In [24]:

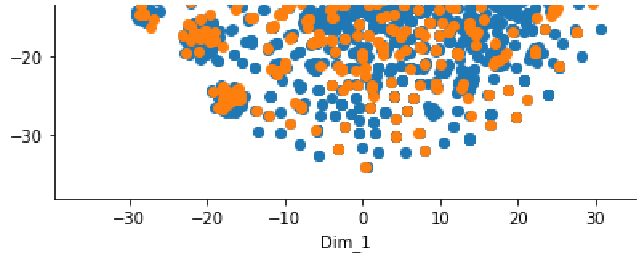
```
%%time
# Plotting with different perplexity and iterations.
# Using Multicore TSNE for fast results since we can use alll cores what we have in our machine.
from MulticoreTSNE import MulticoreTSNE as TSNE

model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=200, n_iter=3000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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




```
Wall time: 9min 25s
```

In [22]:

```
%%time
from MulticoreTSNE import MulticoreTSNE as TSNE

model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=150, n_iter=3000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



Bi Gram/ n-Gram

Motivation

Now that we have our list of words describing positive and negative reviews lets analyse them.

We begin analysis by getting the frequency distribution of the words as shown below

In [25]:

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

```
Most Common Positive Words :  [(b'not', 146798), (b'like', 139432), (b'tast', 129052), (b'good', 1
12766), (b'flavor', 109626), (b'love', 107360), (b'use', 103889), (b'great', 103871), (b'one', 967
28), (b'product', 91034), (b'tri', 86793), (b'tea', 83888), (b'coffe', 78814), (b'make', 75108), (
b'get', 72126), (b'food', 64802), (b'would', 55568), (b'time', 55264), (b'buy', 54198),
(b'realli', 52717)]
Most Common Negative Words :  [(b'not', 54378), (b'tast', 34585), (b'like', 32330), (b'product', 2
8218), (b'one', 20569), (b'flavor', 19575), (b'would', 17972), (b'tri', 17753), (b'use', 15302), (
b'good', 15041), (b'coffe', 14716), (b'get', 13786), (b'buy', 13752), (b'order', 12871), (b'food',
12754), (b'dont', 11877), (b'tea', 11665), (b'even', 11085), (b'box', 10844), (b'amazon', 10073)]
```

Observation:- From the above it can be seen that the most common positive and the negative words overlap for eg. 'like' could be used as 'not like' etc.
So, it is a good idea to consider pairs of consequent words (bi-grams) or q sequnce of n consecutive words (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) ) #in scikit-learn
final_bigram_counts = count_vect.fit_transform(final['CleanedText'].values)
final_bigram_counts.get_shape()
```

Out[26]:

```
(364173, 2905350)
```

In [28]:

```
#Applying TSNE to this n-Gram Representation

from sklearn.preprocessing import MaxAbsScaler

standardized_data = MaxAbsScaler().fit_transform(final_bigram_counts)
data = standardized_data[0:10000,0:10000].toarray()
```

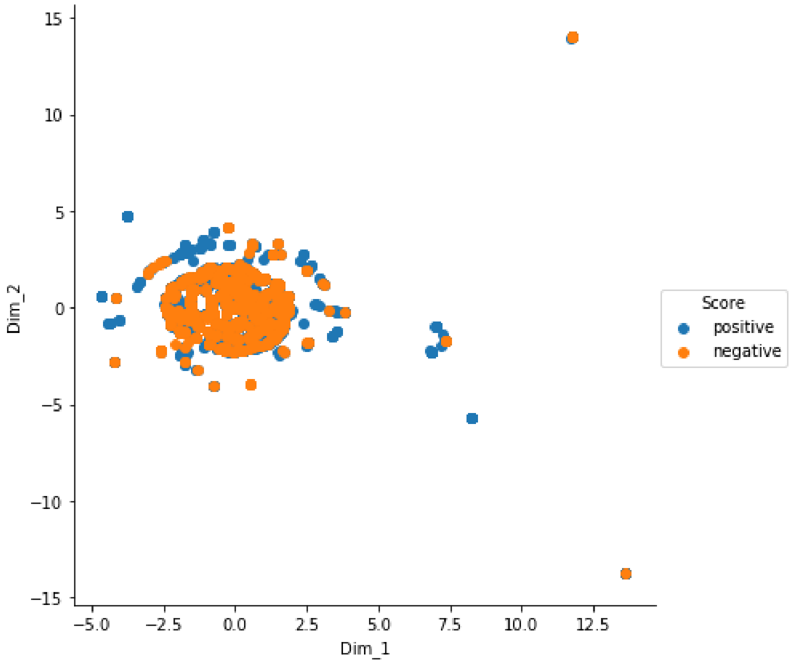
In [29]:

```
label = final.Score[0:10000]
```

In [30]:

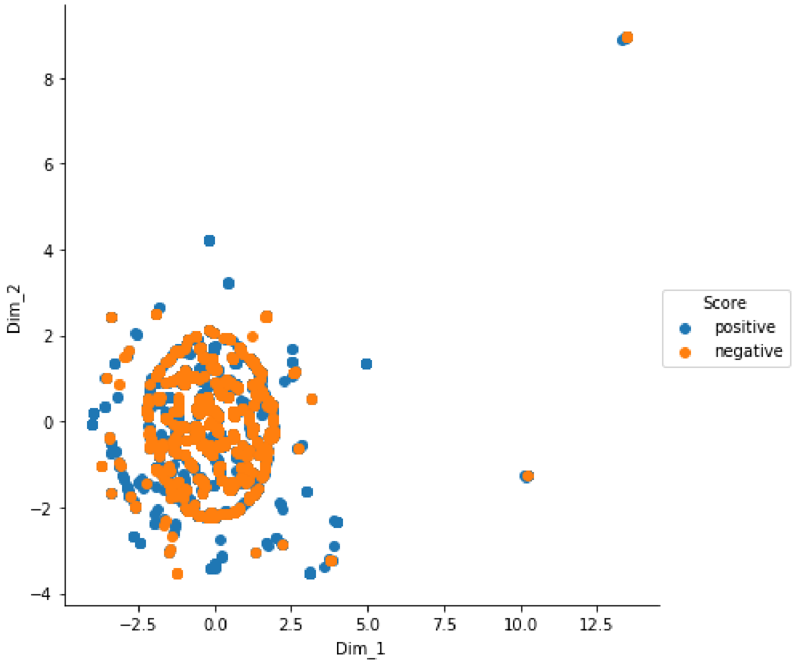
```
from MulticoreTSNE import MulticoreTSNE as TSNE
```

```python
model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=30, n_iter=3000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



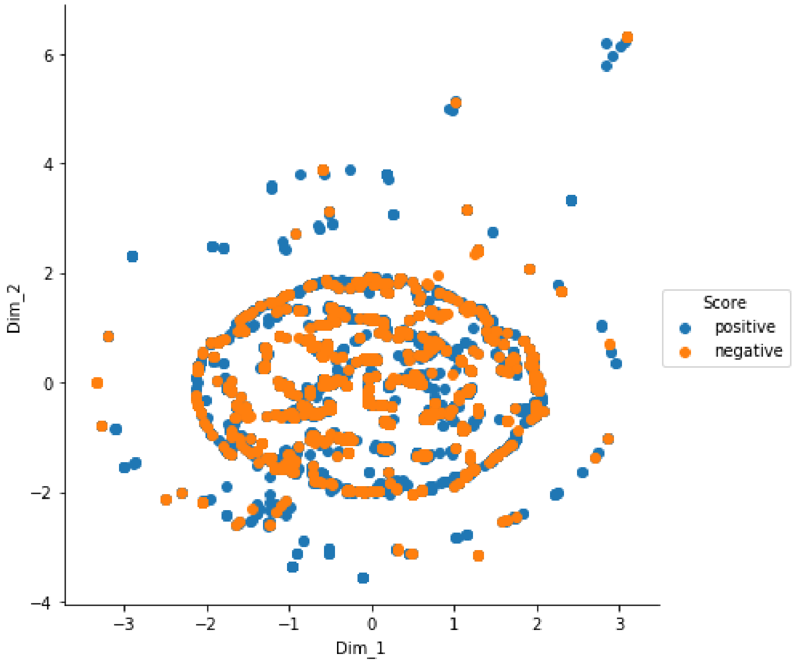
In [31]:

```python
# Plotting with different perplexity and iterations.
from MulticoreTSNE import MulticoreTSNE as TSNE

model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=60, n_iter=3000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



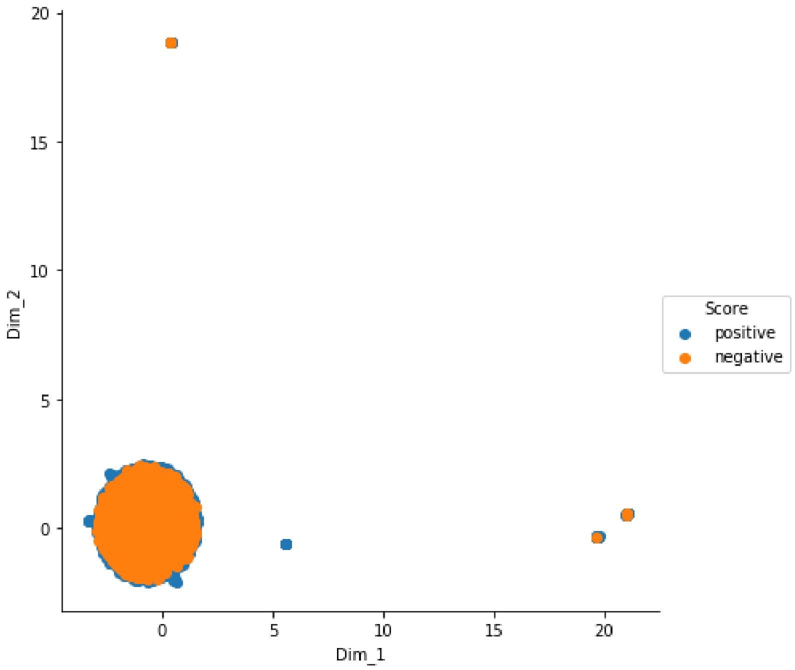
In [32]:

```
# Plotting with different perplexity and iterations.
from MulticoreTSNE import MulticoreTSNE as TSNE

model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=120, n_iter=3000)
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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



In [33]:

```
# Applying UMAP on n-Gram data
import umap
model = umap.UMAP().fit_transform(data)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```



TF-IDF

In [34]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final['CleanedText'].values)
```

In [35]:

```
final_tf_idf.get_shape()
```

Out[35]:

```
(364173, 2905350)
```

In [36]:

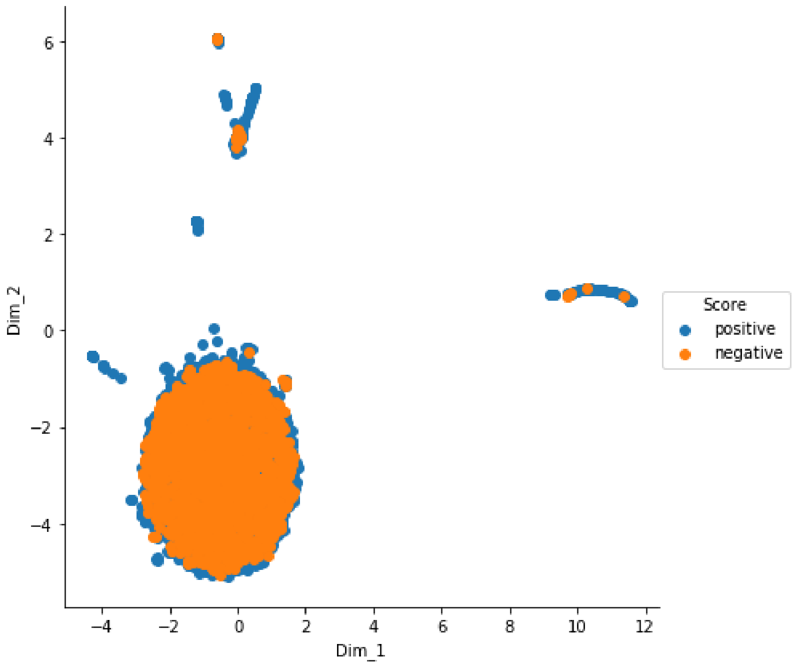
```
from sklearn.preprocessing import MaxAbsScaler

standardized_data = MaxAbsScaler().fit_transform(final_tf_idf)
data = standardized_data[0:10000,0:10000].toarray()
```

In [37]:

```
label = final.Score[0:10000]
```

In [38]:

```
import umap
model = umap.UMAP().fit_transform(data)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```

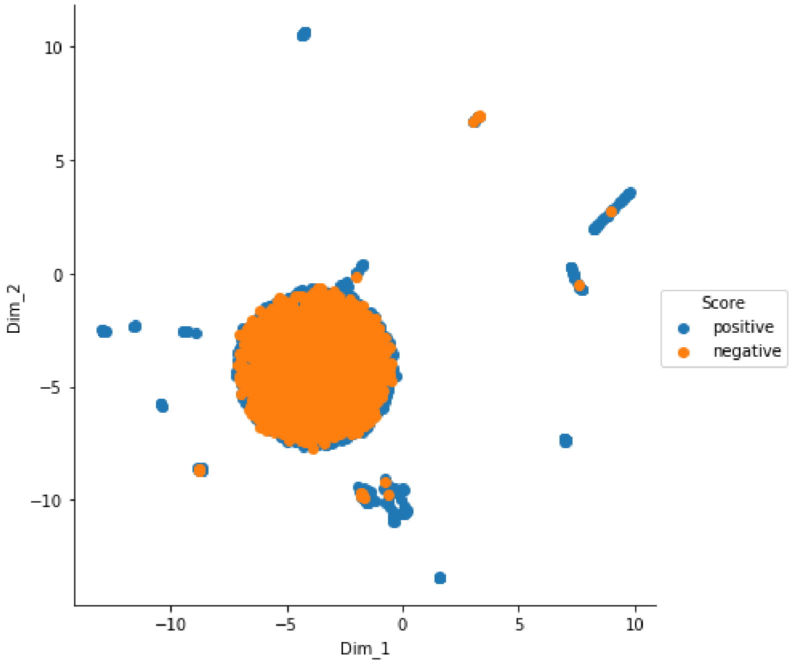


In [39]:

```
# Plotting with different parameters.
import umap
model = umap.UMAP(n_neighbors=5,min_dist=0.3).fit_transform(data)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
```

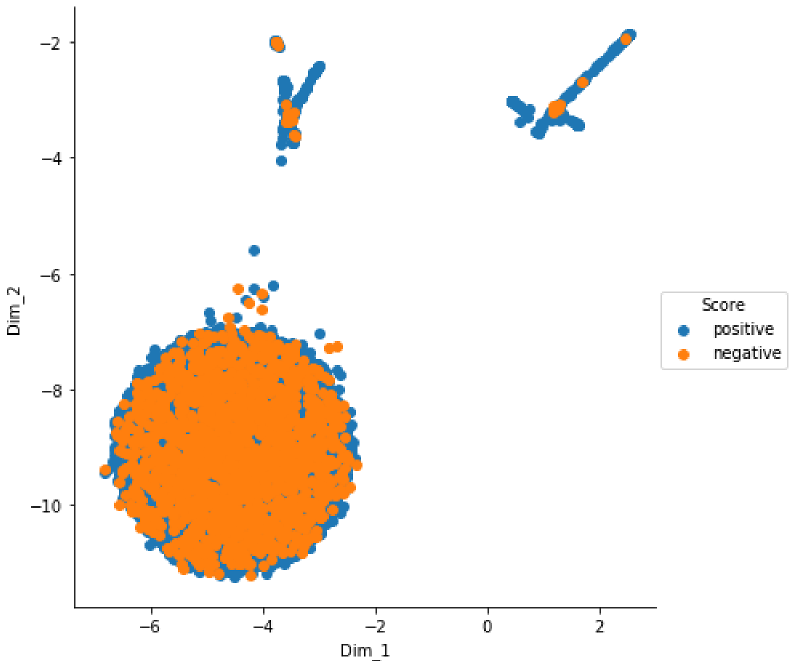
```
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\umap\spectral.py:229: UserWarning: Embedding 5
connected components using meta-embedding (experimental)
  n_components
```



In [40]:

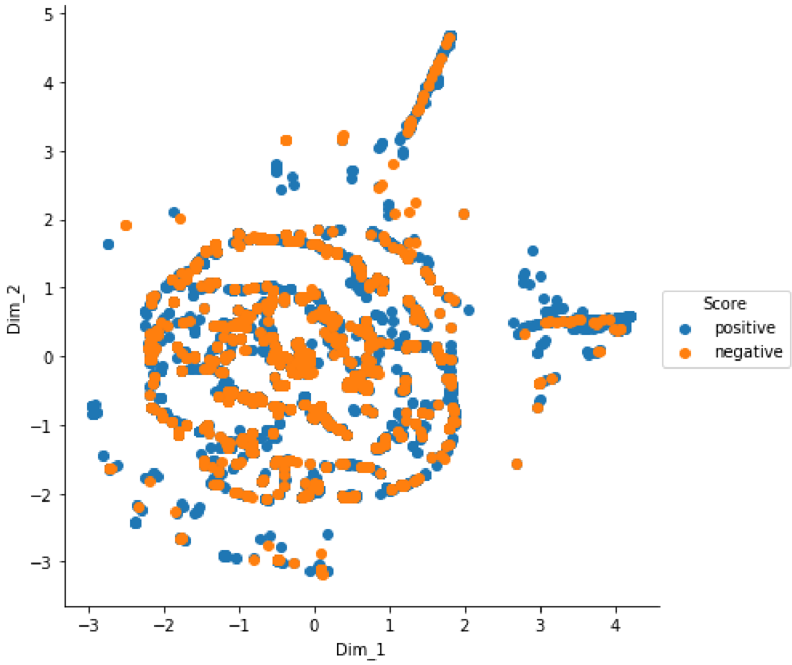
```
import umap
# Plotting with different perplexity and iterations.
model = umap.UMAP(n_neighbors=30,min_dist=0.1).fit_transform(data)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```



In [41]:

```
%%time
from MulticoreTSNE import MulticoreTSNE as TSNE

model = TSNE(n_jobs=6,n_components=2, random_state=0, perplexity=80, n_iter=3000)
```

```
tsne_data = model.fit_transform(data)
tsne_data1 = np.vstack((tsne_data.T, label)).T
tsne_df = pd.DataFrame(data=tsne_data1, columns=("Dim_1", "Dim_2", "Score"))

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




```
Wall time: 14min 27s
```

Word to Vector(W2V)

Here, in word to vector, it takes word as input and creates a vector in high dimentions(typically 100,200,300).

In [42]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
# Train own Word2Vec model using your own text corpus
import gensim
i=0
list_of_sent=[]
for sent in final['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
# here we are creating a lists[] in list where each list in a list is a sentence. eg:[['pasta', ve
ry', 'delicious'],['daddy', 'coming', 'late'].......[]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; al
iasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [43]:

```
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=6)
#training a w2v
# min_count parameter means if a word doesnt occur atleast 3 tymes then dont create a vector for i
t.
```

```
#size param means, for a word we need to create a vector so what dimention we want.
# workers means, no.of cores we want to use, e.g: we have 6 cores
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater perf
ormance
```

Average Word to Vector(Avg W2V)

In [44]:

```
# 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 list_of_sent:     # for each review/sentence
    sent_vec = np.zeros(250)  # as word vectors are of zero length
    cnt_words =0;             # num of words with a valid vector in the sentence/review
    for word in sent:         # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: RuntimeWarning: invalid value
encountered in true_divide
```

```
364173
250
```

In [46]:

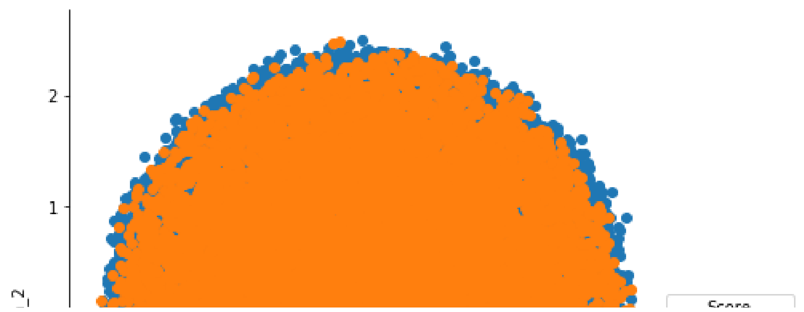
```
from sklearn.preprocessing import StandardScaler
shp_mtx = np.reshape(sent_vectors,(364173,250))
avg_mtx=np.nan_to_num(shp_mtx)
std_data = StandardScaler().fit_transform(avg_mtx)
```

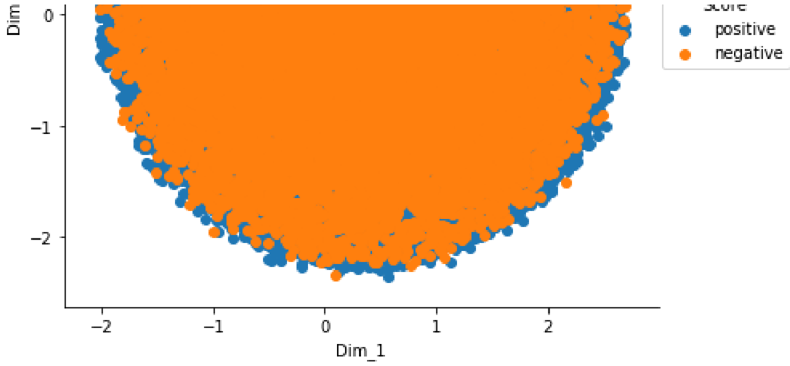
In [47]:

```
data1 = std_data[0:100000,:]
label = final.Score[0:100000]
```

In [48]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=30,min_dist=0.1).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```





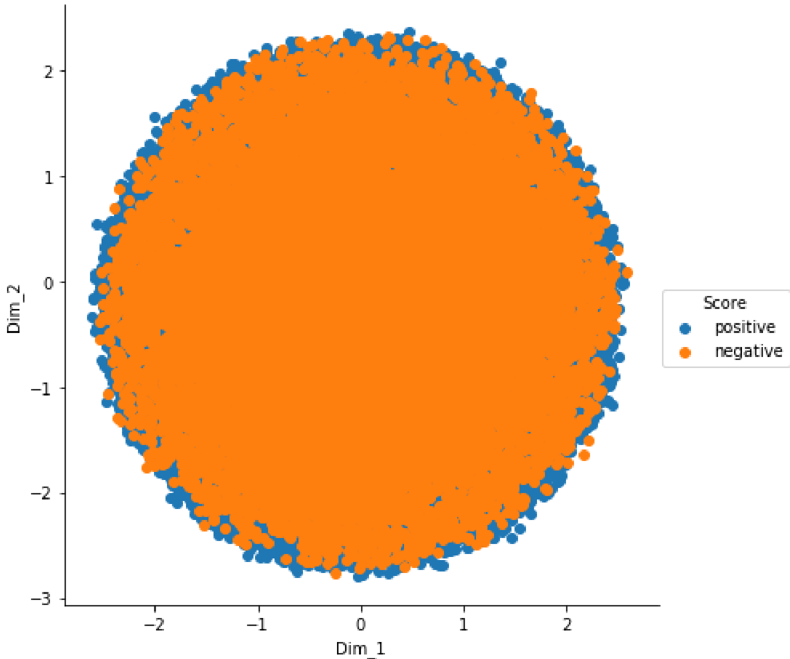

```
Wall time: 7min 56s
```

In [76]:

```
data1 = std_data[0:5000,:]
label = final.Score[0:5000]
```

In [49]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=30,min_dist=0.2).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```




```
Wall time: 7min 48s
```

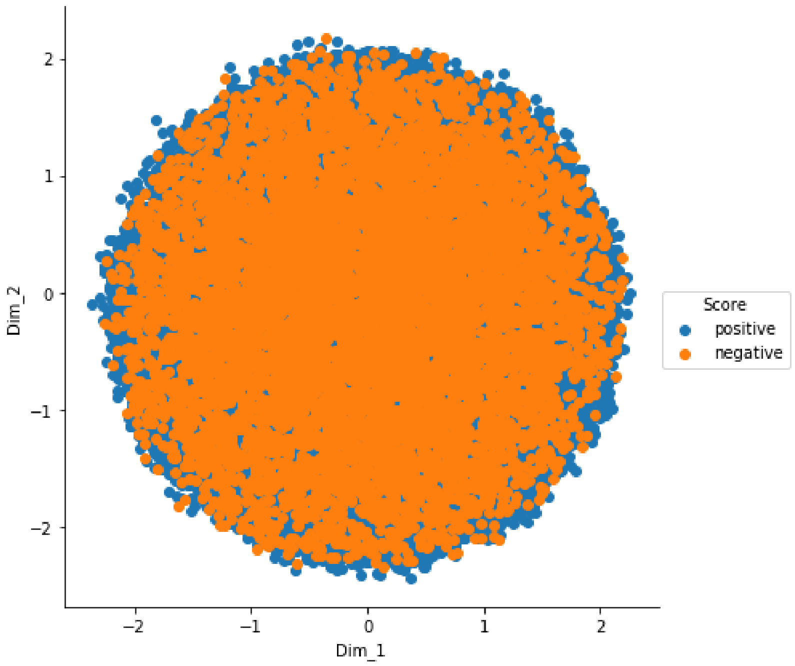
In [50]:

```
data1 = std_data[0:50000,:]
label = final.Score[0:50000]
```

In [51]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=30,min_dist=0.05).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
```

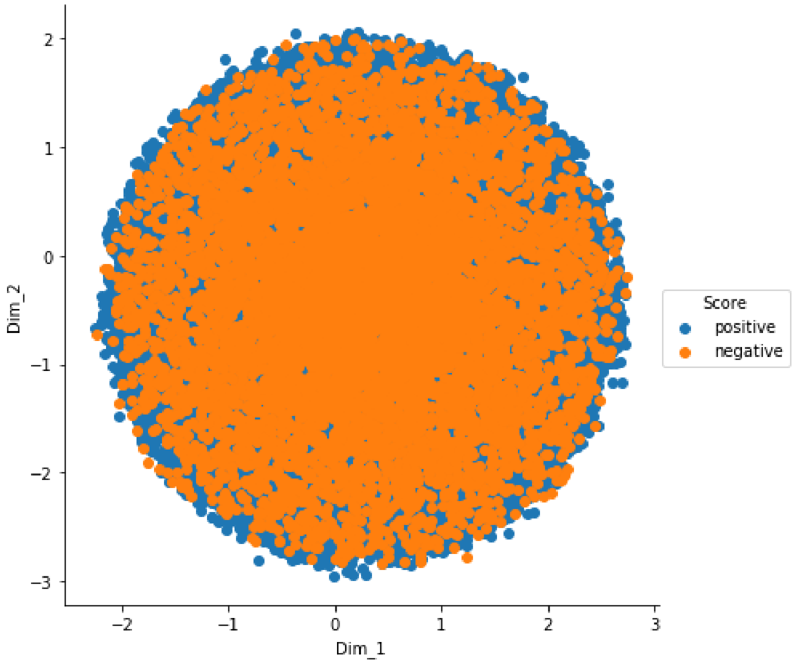
```
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```



Wall time: 3min 31s

In [52]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=100,min_dist=0.1).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```



Wall time: 10min 54s

Tf-id Word2Vec

In [53]:

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

tfidf_sent_vectors = [] # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sent[0:1000]: # for each review/sentence
    sent_vec = np.zeros(100) # 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
        try:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tfidf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:19: RuntimeWarning: invalid value
encountered in true_divide
```

```
Wall time: 2h 20min 37s
```

In [54]:

```
import numpy as np
from sklearn.preprocessing import StandardScaler
shp_mtx = np.reshape(tfidf_sent_vectors,(1000,100))
avg_mtx=np.nan_to_num(shp_mtx)
std_data = StandardScaler().fit_transform(avg_mtx)
```

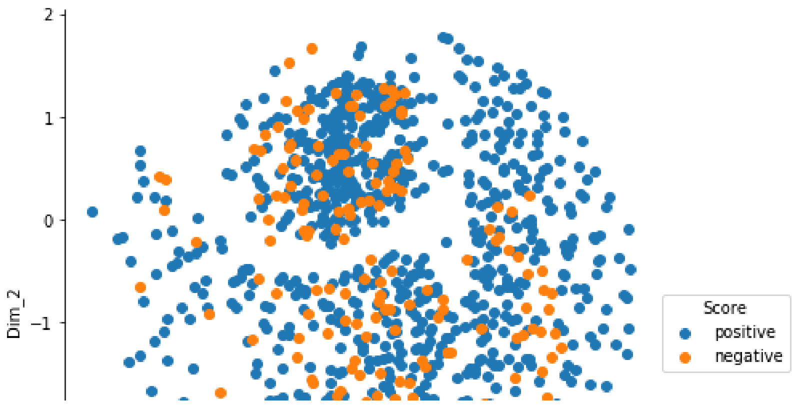
In [55]:

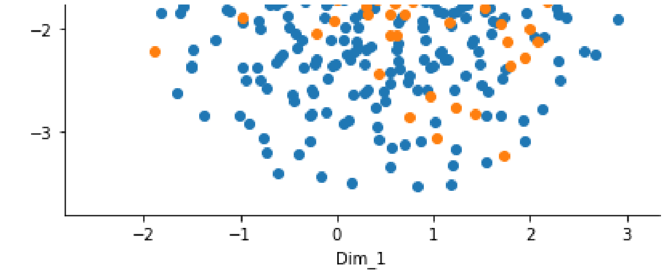
```
data1 = std_data
label = final.Score[0:1000]
```

In [56]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=30,min_dist=0.1).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:257: RuntimeWarning:
invalid value encountered in sqrt
  return distances if squared else np.sqrt(distances, out=distances)
```



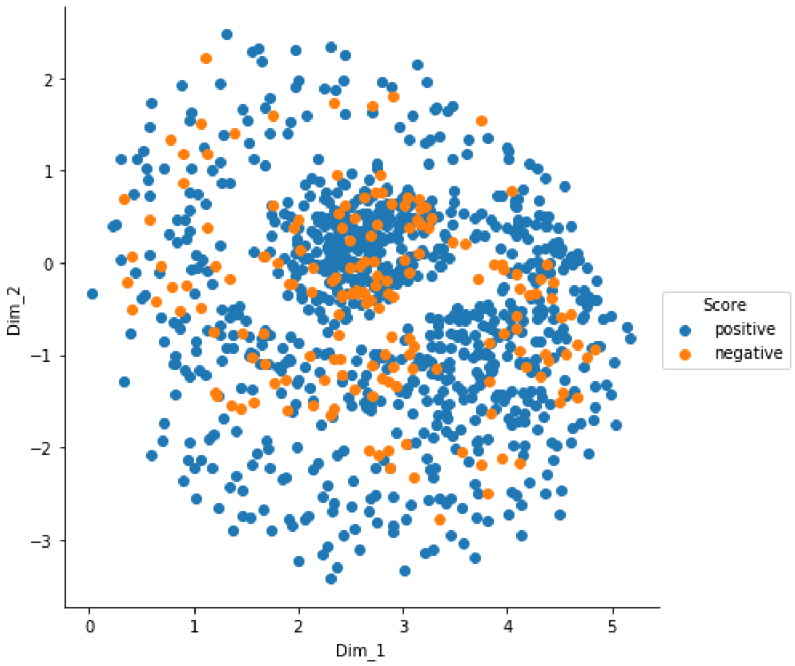



```
Wall time: 15 s
```

In [30]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=30,min_dist=0.05).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:257: RuntimeWarning:
invalid value encountered in sqrt
  return distances if squared else np.sqrt(distances, out=distances)
```

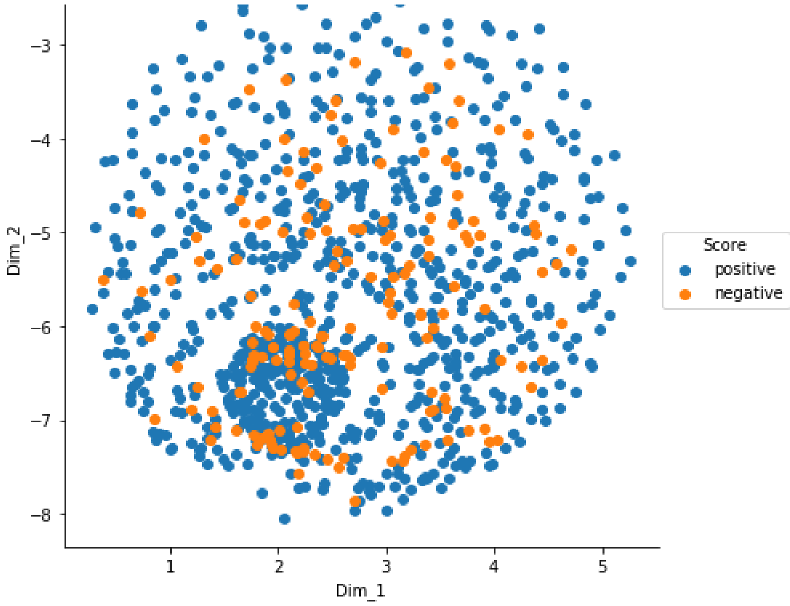



```
Wall time: 11.7 s
```

In [57]:

```
%%time
import umap
model = umap.UMAP(n_neighbors=100,min_dist=0.1).fit_transform(data1)

umapdata1 = np.vstack((model.T, label)).T
umap_df = pd.DataFrame(data=umapdata1, columns=("Dim_1", "Dim_2", "Score"))
sns.FacetGrid(umap_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\pairwise.py:257: RuntimeWarning:
invalid value encountered in sqrt
  return distances if squared else np.sqrt(distances, out=distances)
```




```
Wall time: 36.5 s
```

Conclusion

1. We named a review as positive if the rating given by the user is 4or5 and negative if the rating given is 1or2. We left out the reviews with rating as 3 because it could neither be stated as positive nor negative.
2. We performed data cleaning. If removed all the duplicate reviews given by the same user on the same product.
3. We used some domain knowledge and solved some problems(i.e helpfulness numerator<= helpfulness denominator).
4. After data cleaning, we successfully reduced the number of rows from 568,454 to 364171.
5. We perfomed text processing like Lemmatization, stopword removal,Stemming to make the data ready for further process.
6. After text processing applied techniques like BoW, Tf-Idf, W2V, Avg-W2v, tfidf-w2v on the data so that this result data will be used for training purpose for algorithms like K-NN, Naive Bayes,SVM,etc.