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



# using the SQLite Table to read data.
con = sqlite3.connect('C:/AI/amazon-fine-food-reviews/database.sqlite')



#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 """,


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

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print(filtered_data.shape)
```

```
(525814, 10)
```

In [2]:
```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplac
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
final.shape
```

Out[2]:
```
(364173, 10)
```

In [3]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[3]:
```
69.25890143662969
```

In [4]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

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

```
(364171, 10)
```

Out[4]:
```
positive    307061
negative     57110
Name: Score, dtype: int64
```

In [5]:
```
final = final.sample(2000)
final.shape
```

Out[5]:
```
(2000, 10)
```

In [6]:
```
stop = set(stopwords.words('english')) #set of stopwords
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)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or specia
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [7]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing ph
# 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:
    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):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to d
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to d
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("*********************************************************************

    final_string.append(str1)
    i+=1
```

In [8]:
```
final['CleanedText']=final_string #adding a column of CleanedText which displays
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
final.head(3)
```

Out[8]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulnes |
|---|---|---|---|---|---|---|
| **138977** | 150821 | B00146K7MU | A23OJZCP25VVIE | S. J. Arceneaux Jr. | 0 | |
| **275811** | 298908 | B0012C2GFM | A1XQMQMF07QZQS | GalCalif "Visit my Amazon Profile page!" | 6 | |
| **467003** | 504984 | B003DVMYUW | A1WYLU6QMLT3YP | Michelle | 0 | |

In [9]:
```
# store final table into an SQLLite table for future.
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, inde
```

In [10]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
final_counts = count_vect.fit_transform(final['CleanedText'].values)
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  (2000, 6353)
```

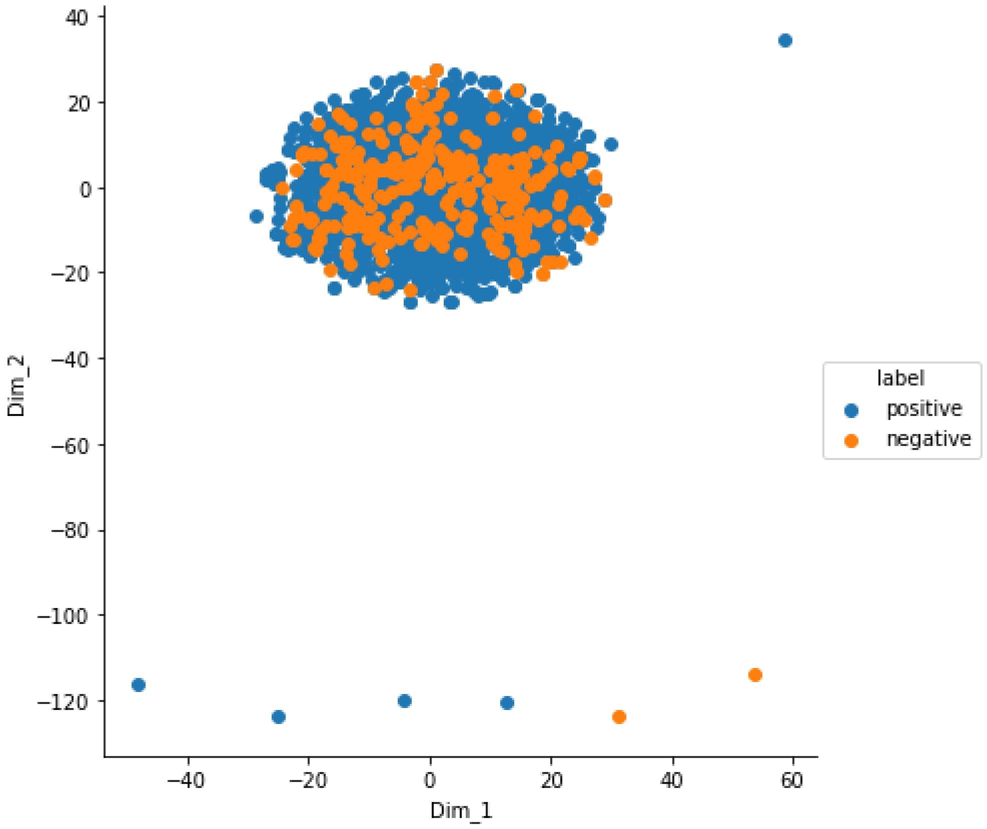
In [11]:
```
final_counts= final_counts.todense()
```

In [12]:
```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(final_counts)
print(standardized_data.shape)
```

```
(2000, 6353)
```

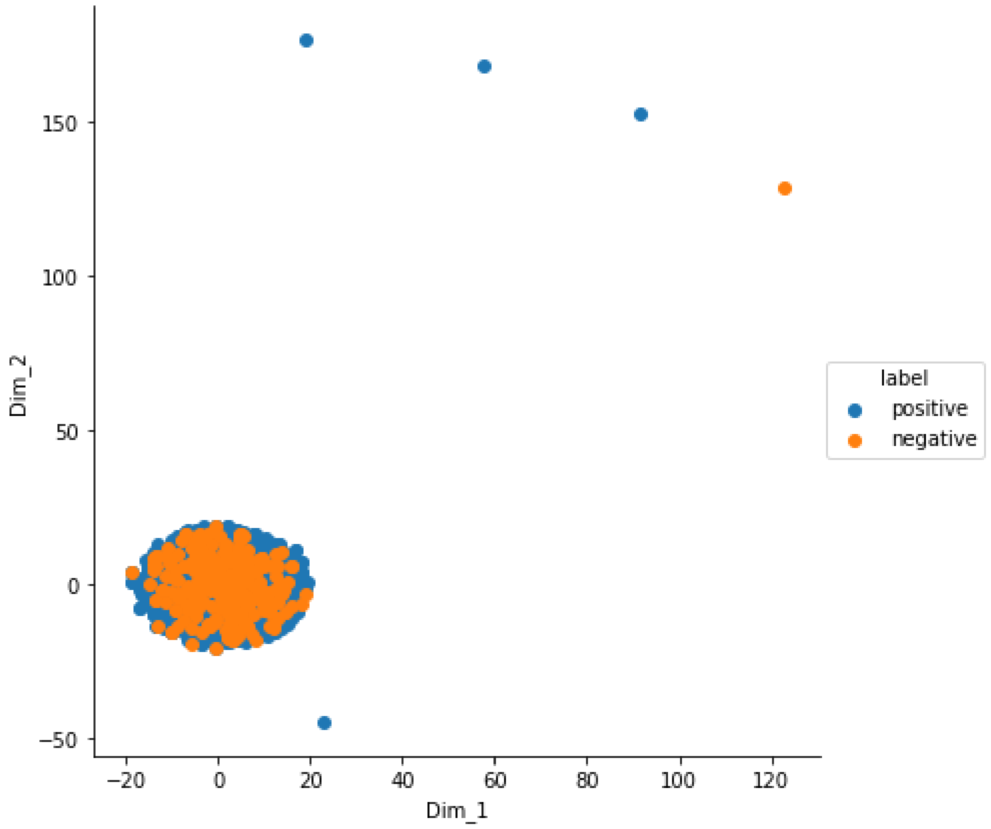
In [89]:

```
# TSNE

from sklearn.manifold import TSNE

labels = final['Score']

model = TSNE(n_components=2, random_state=0, perplexity =20, n_iter = 2000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(final_counts)


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

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



In [57]:

```
# TSNE

from sklearn.manifold import TSNE

labels = final['Score']

model = TSNE(n_components=2, random_state=0, perplexity =25, n_iter = 5000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(final_counts)


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

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



# TF-IDF

In [24]:

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2000, 83703)
the number of unique words including both unigrams and bigrams  83703
```

In [25]:

```
features = tf_idf_vect.get_feature_names()
len(features)
print("some sample features(unique words in the corpus)",features[1000:1020])
```

```
some sample features(unique words in the corpus) ['833', '833 5330', '84', '84
61', '85', '85 lean', '85g', '85g sodium', '86', '86 and', '88', '88 or', '88 p
er', '89', '89 cents', '89 was', '8g', '8g br', '8g now', '8oz']
```

In [26]:

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

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

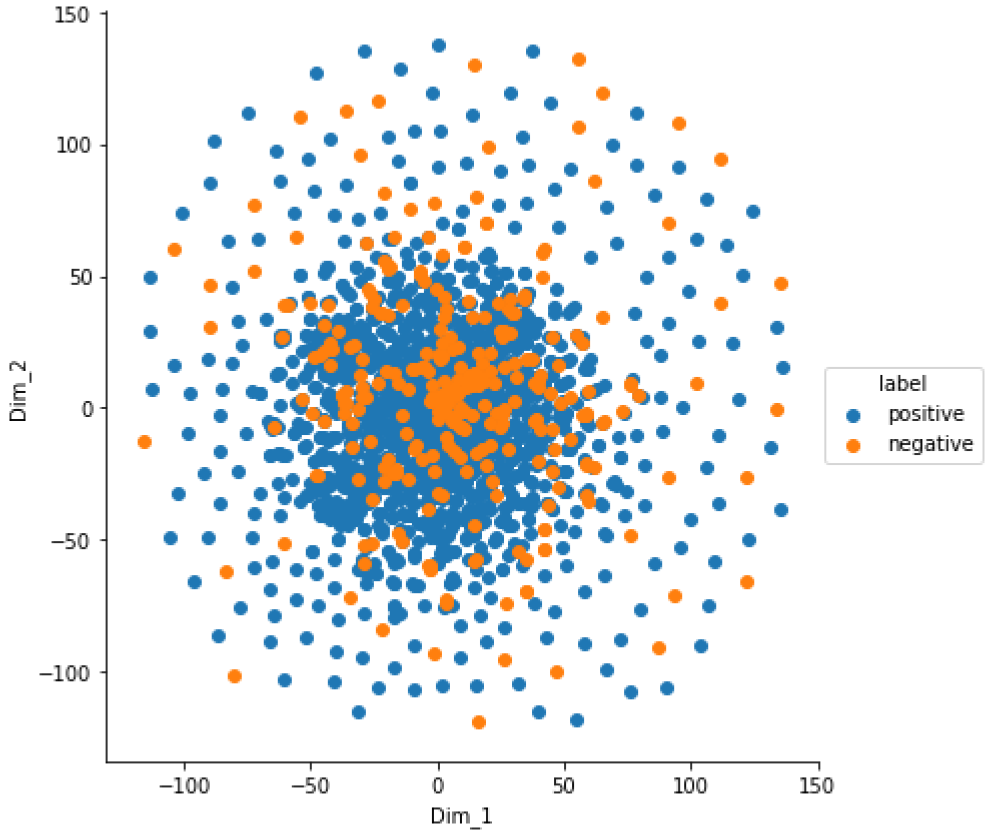
In [28]:

```
top_tfidf
```

Out[28]:

|  | feature | tfidf |
|---|---|---|
| **0** | br | 0.176346 |
| **1** | powder | 0.144223 |
| **2** | small | 0.116493 |
| **3** | are small | 0.106974 |
| **4** | not powder | 0.106974 |
| **5** | br br | 0.100235 |
| **6** | things | 0.096806 |
| **7** | good things | 0.096697 |
| **8** | are | 0.095742 |
| **9** | the freezer | 0.094950 |
| **10** | them | 0.094145 |
| **11** | freezer | 0.088400 |
| **12** | meals | 0.088400 |
| **13** | these are | 0.088002 |
| **14** | of other | 0.086420 |
| **15** | of | 0.079829 |
| **16** | seeds | 0.078662 |
| **17** | but if | 0.078662 |
| **18** | lots of | 0.078123 |
| **19** | lots | 0.075687 |
| **20** | are not | 0.075244 |
| **21** | in | 0.073416 |
| **22** | they are | 0.073348 |
| **23** | and | 0.072829 |
| **24** | or | 0.072673 |

In [66]:
```
# TSNE

from sklearn.manifold import TSNE

labels = final['Score']
final_tf_idf = final_tf_idf.todense()

model = TSNE(n_components=2, random_state=0, perplexity =10, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(final_tf_idf)


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

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



# Word2Vec

In [13]:

```

i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [14]:

```
print(final['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
delici enjoy mani peopl pastel melt requir special storag handl
*****************************************************************
['delici', 'enjoy', 'mani', 'peopl', 'pastel', 'melt', 'requir', 'special', 'st
orag', 'handl']
```

In [16]:

```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [17]:

```
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  2019
sample words  ['delici', 'enjoy', 'mani', 'peopl', 'melt', 'requir', 'special',
'storag', 'handl', 'hemp', 'seed', 'lot', 'good', 'thing', 'go', 'includ', 'fib
er', 'protein', 'high', 'content', 'fatti', 'acid', 'load', 'put', 'tbsp', 'sal
ad', 'health', 'benefit', 'real', 'flavor', 'one', 'way', 'anoth', 'pretti', 's
mall', 'smaller', 'quit', 'happili', 'get', 'lost', 'mix', 'food', 'toss', 'dia
bet', 'treat', 'diet', 'chang', 'longer', 'eat', 'like']
```

In [18]:

```
w2v_model.wv.most_similar('tasti')
```

Out[18]:

```
[('gave', 0.9998287558555603),
 ('fish', 0.9998136162757874),
 ('red', 0.9998095631599426),
 ('put', 0.9998030662536621),
 ('nut', 0.9998030662536621),
 ('pretti', 0.9997985363006592),
 ('still', 0.9997972846031189),
 ('regular', 0.9997951984405518),
 ('top', 0.9997913837432861),
 ('amount', 0.9997901916503906)]
```

In [20]:

```
w2v_model.wv.most_similar('like')
```

Out[20]:

```
[('smell', 0.9997526407241821),
 ('pretti', 0.9997172355651855),
 ('sweet', 0.999687910079956),
 ('nice', 0.9996700882911682),
 ('vanilla', 0.9996681213378906),
 ('cream', 0.9996570348739624),
 ('drink', 0.9996509552001953),
 ('better', 0.9996441602706909),
 ('regular', 0.9996427893638611),
 ('spici', 0.9996349811553955)]
```

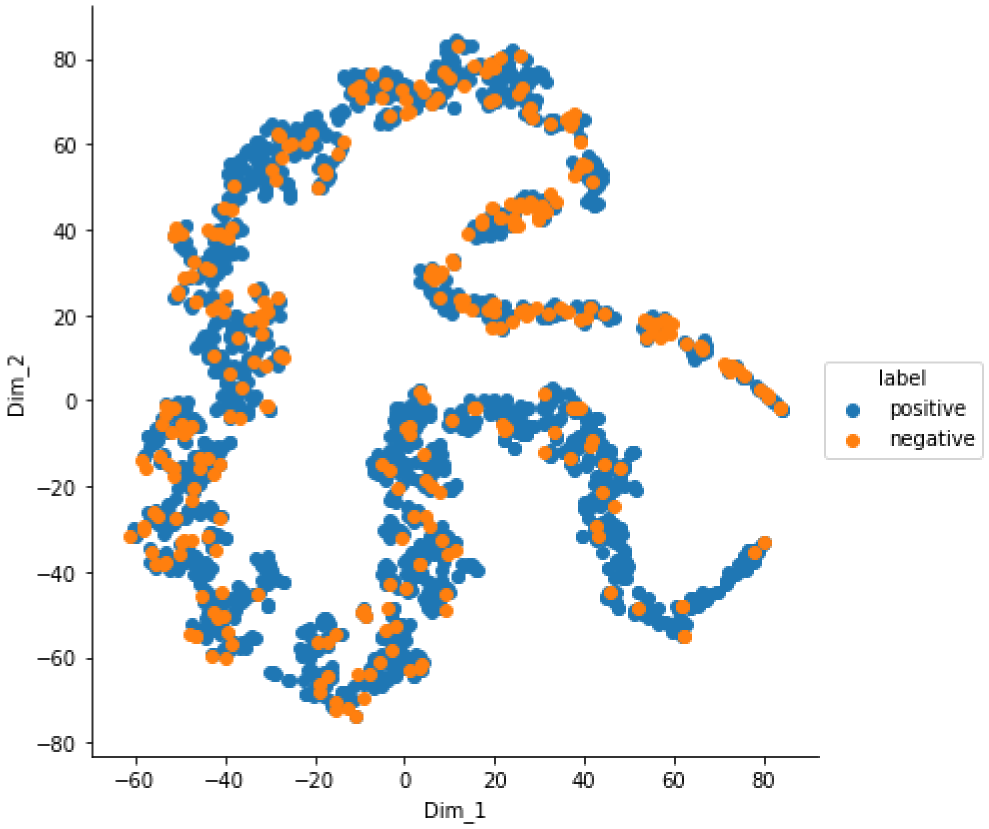
In [21]:

```
# 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(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
2000
50
```

In [22]:

```
# TSNE

from sklearn.manifold import TSNE

labels = final['Score']

model = TSNE(n_components=2, random_state=0, perplexity =10, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(sent_vectors)


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

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

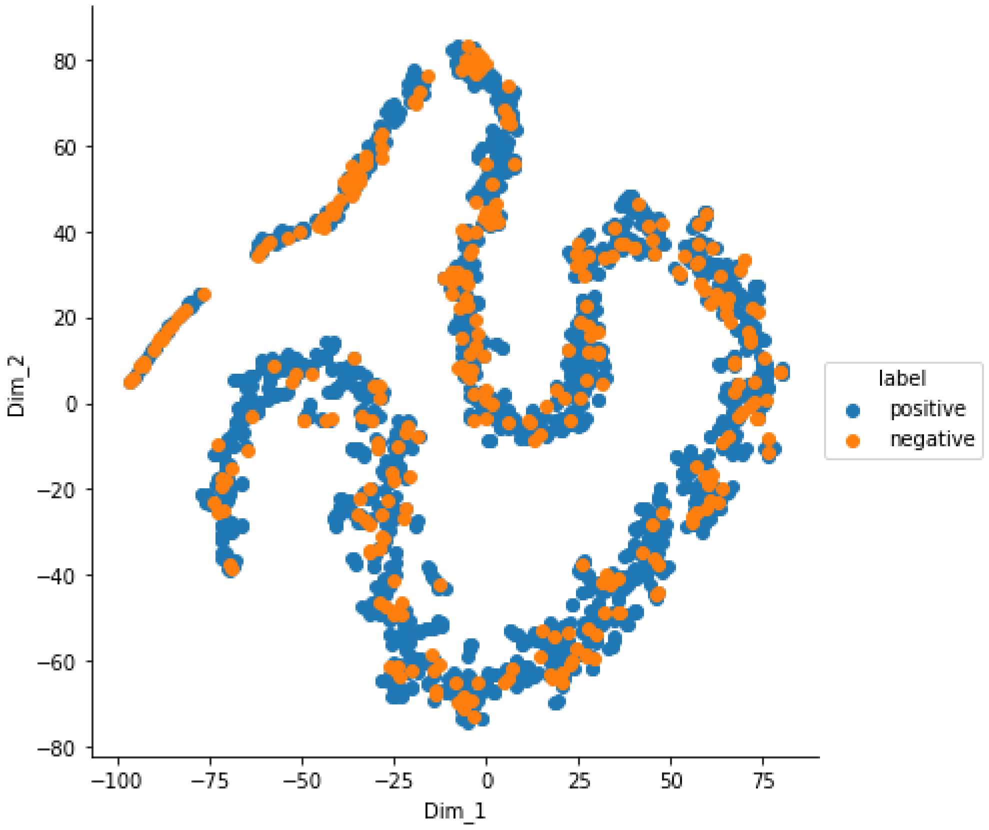


In [31]:

```
# 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 = t

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
row=0;
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent:# for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [34]:

```
# TSNE

from sklearn.manifold import TSNE

labels = final['Score']


model = TSNE(n_components=2, random_state=0, perplexity =10, n_iter = 1000)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(tfidf_sent_vectors)


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

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


Dim_2
Dim_1
label
● positive
● negative