In [3]:

```
%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
#taking cleaned data i.e in Reviews table  from final sql database
#making connection with database
conn = sqlite3.connect('final.sqlite')
final = pd.read_sql_query(""" SELECT * FROM Reviews """, conn)
```

In [4]:

```
print(final.shape) #number of attributes and size of the data
final.head()
```

```
(364171, 12)
```

Out[4]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue	1	1	positive

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
					"sally sue"			
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	1	positive
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	4	positive

In [14]:

```
#BoW
#The fit_transform method applies to feature extraction objects such as CountVectorizer and TfidfT
ransformer.
#The "fit" part applies to the feature extractor itself: it determines what features it will base
future transformations on.
#The "transform" part is what takes the data and returns some transformed data us

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())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 71624)
the number of unique words  71624
```

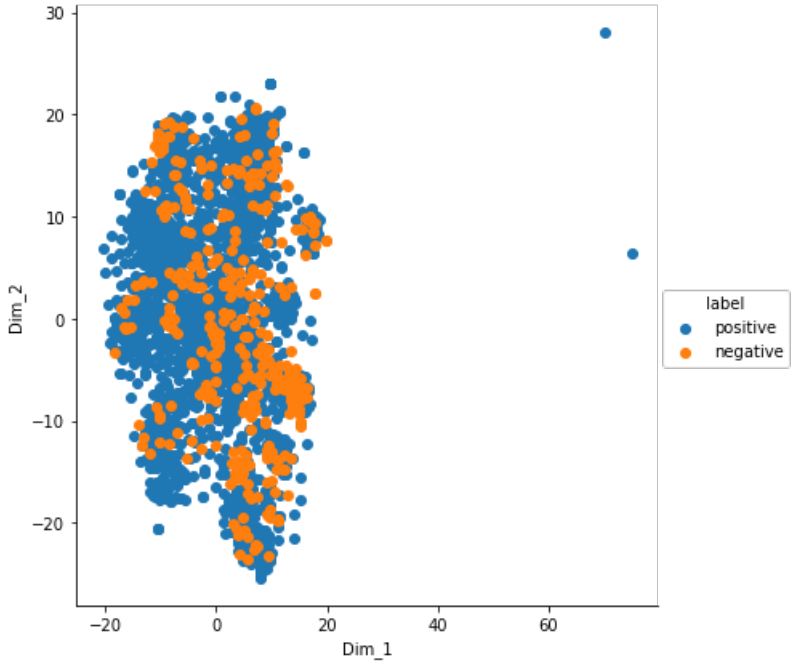
In [13]:

```
#First we will create CountVectorizer Object. Well, you call fit_transform for this object and the
n our cleanedTest column
#of strings. What it does is tokenize the strings and give you a vector for each string, each dime
nsion of which corresponds
#to the number of times a token is found in the corresponding string. Most of the entries in all o
f the vectors will be zero,
#since only the entries which correspond to tokens found in that specific string will have positiv
e values, but the vector is
#as long as the total number of tokens for the whole corpus.
#todense() :- Return a dense matrix representation of this matrix.
from sklearn.feature_extraction.text import CountVectorizer
ctext = final['CleanedText'];
ctext[:10000].count()

vectorizer = CountVectorizer()
finalbow= vectorizer.fit_transform(ctext[:10000]).todense()
print( finalbow )
#print( vectorizer.vocabulary_ )
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

In [16]:

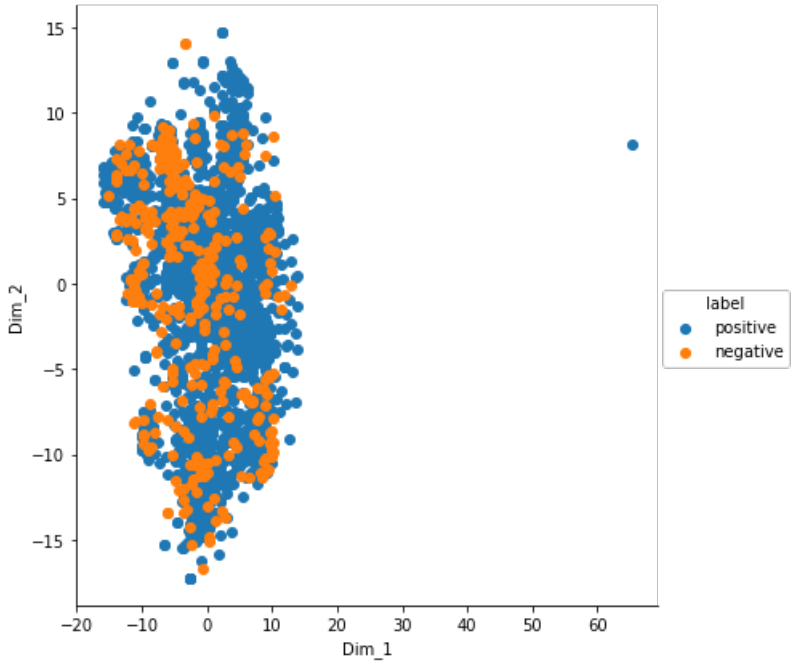
```
#BOW---> TSNE

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
# Picking the top 10000 points
vectorizer = CountVectorizer()
finalbow= vectorizer.fit_transform(ctext[:2000]).todense()
data_2000 = finalbow
labels_2000 = lable[0:2000]

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

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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



In [18]:

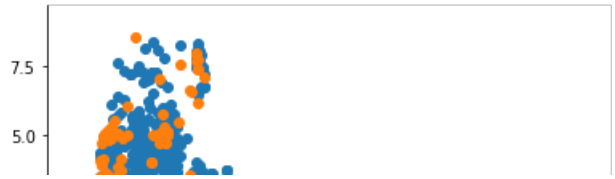
```
#BOW---> TSNE

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
# Picking the top 2000 points
# with perplexity=50
vectorizer = CountVectorizer()
finalbow= vectorizer.fit_transform(ctext[:2000]).todense()
data_2000 = finalbow
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity=50, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30 --- > using perplexity = 50
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000
```

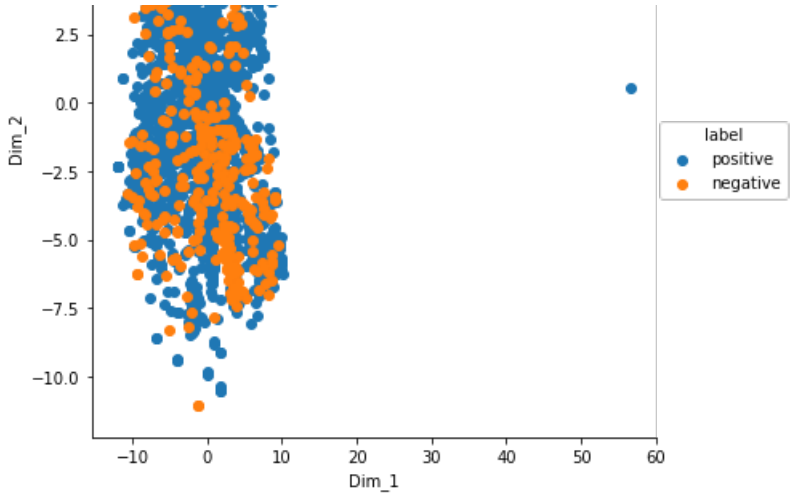
```
tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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



In [19]:

```
#BOW---> TSNE

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
# Picking the top 2000 points
# with perplexity=80
vectorizer = CountVectorizer()
finalbow= vectorizer.fit_transform(ctext[:2000]).todense()
data_2000 = finalbow
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity=80, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30 --- > using perplexity = 50
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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





In [20]:

```
#tf-idf vec
tf_idf_data = final['CleanedText']
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(tf_idf_data[:2000].values)
#print(final_tf_idf)
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])
```

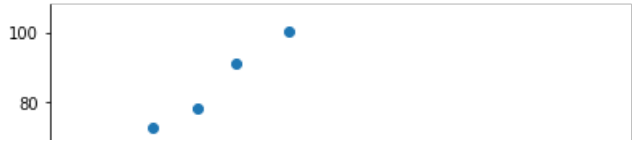
```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (2000, 76693)
the number of unique words including both unigrams and bigrams  76693
```

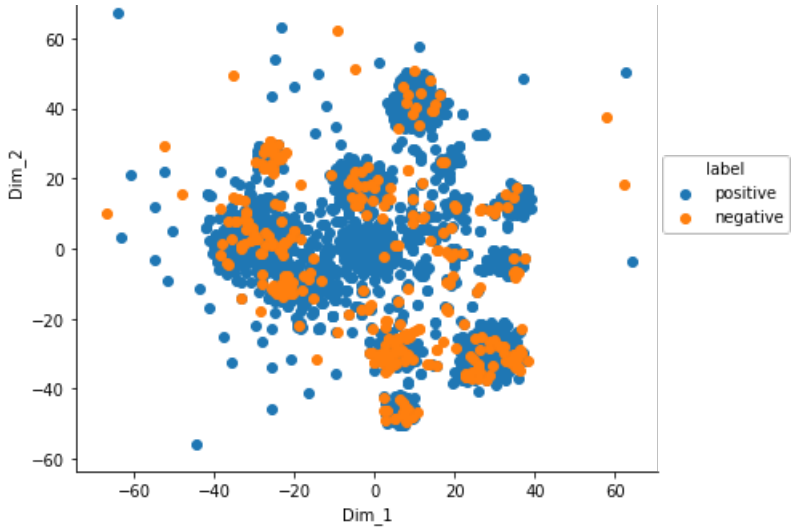
In [21]:

```
#tfidf
from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
# Picking the top 2000 points
tf_idf_data = final['CleanedText']
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(tf_idf_data[:2000].values).todense()
data_2000 = final_tf_idf
labels_2000 = lable[0:2000]

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

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[21]:

```
<seaborn.axisgrid.FacetGrid at 0x20760ddf748>
```



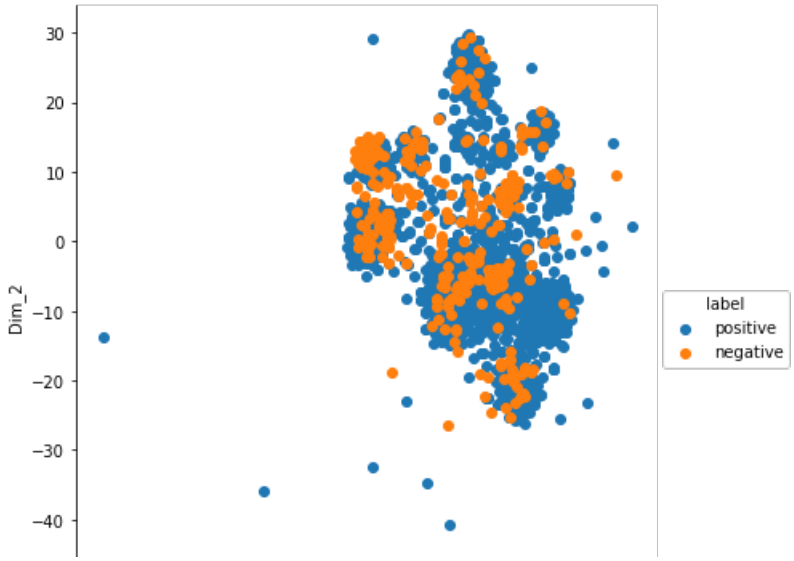


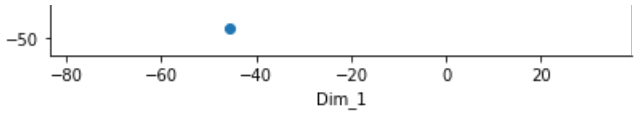
In [22]:

```
#tfidf
from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
# Picking the top 2000 points
#with change in perplexity =80
tf_idf_data = final['CleanedText']
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(tf_idf_data[:2000].values).todense()
data_2000 = final_tf_idf
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity=80, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30 --> with change in perplexity =80
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[22]:

```
<seaborn.axisgrid.FacetGrid at 0x20764ff39e8>
```





In [24]:

```
#Word2Vec mode
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sent=[]
for sent in final['CleanedText'].values:
    list_of_sent.append(sent.split())

print(final['CleanedText'].values[0])
print("\n--------------Spliting each sentence into words--------word list of ie data corpus--------
-\n")
print(list_of_sent[:2])
#word list of ie data corpus
```

```
witti littl book make son laugh loud recit car drive along alway sing refrain hes learn whale
india droop love new word book introduc silli classic book will bet son still abl recit memori col
leg

--------------Spliting each sentence into words--------word list of ie data corpus---------

[['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'car', 'drive', 'along', 'alw
ay', 'sing', 'refrain', 'hes', 'learn', 'whale', 'india', 'droop', 'love', 'new', 'word', 'book',
'introduc', 'silli', 'classic', 'book', 'will', 'bet', 'son', 'still', 'abl', 'recit', 'memori', '
colleg'], ['grew', 'read', 'sendak', 'book', 'watch', 'realli', 'rosi', 'movi', 'incorpor', 'love'
, 'son', 'love', 'howev', 'miss', 'hard', 'cover', 'version', 'paperback', 'seem', 'kind',
'flimsi', 'take', 'two', 'hand', 'keep', 'page', 'open']]
```

In [25]:

```
#The Word to Vec model produces a vocabulary, with each word being represented by
#an n-dimensional numpy array
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_model.wv['man']
```

Out[25]:

```
array([ 0.48298913,  2.1642413 ,  0.06987273,  0.3015431 , -0.31633124,
       -0.21095029, -0.9512432 , -0.49110314,  0.9814498 , -2.043004  ,
       -1.9518774 , -0.7718721 , -0.6482141 ,  1.2035885 ,  2.050793  ,
        0.81042564,  1.0750993 , -1.0640193 ,  0.2727537 ,  0.6583929 ,
        1.2642627 ,  0.711387  ,  1.0291779 , -0.5976841 , -0.09448117,
        0.06119768,  0.1058379 , -1.6150448 , -0.29101595,  0.32835075,
        0.62694144,  0.65934515, -0.5282842 ,  2.0031388 ,  1.335773  ,
        1.4079427 ,  2.066061  , -0.9159664 , -2.5419009 , -1.2010802 ,
        1.5252602 , -1.5026264 ,  0.50440353,  0.98826224,  0.6000344 ,
        0.7116026 , -0.6992033 ,  1.6632123 , -0.2933709 ,  0.9843001 ],
      dtype=float32)
```

In [27]:

```
#labels = []
tokens = []

for word in w2v_model.wv.vocab:
    tokens.append(w2v_model[word])
    #labels.append(word)

print(len(tokens));
tokens[:1]
```

```
21938
```

Out[27]:

```
[array([ 1.27351597e-01,  3.12062562e-01,  8.42558742e-02,  1.96281020e-02,
         9.54858959e-02, -7.91134760e-02,  1.59643039e-01, -4.40884978e-02,
         1.52912870e-01,  3.43161523e-02, -2.90893354e-02, -1.98353231e-01,
         2.45877896e-02,  6.88392743e-02,  1.49498489e-02,  6.51749894e-02
```

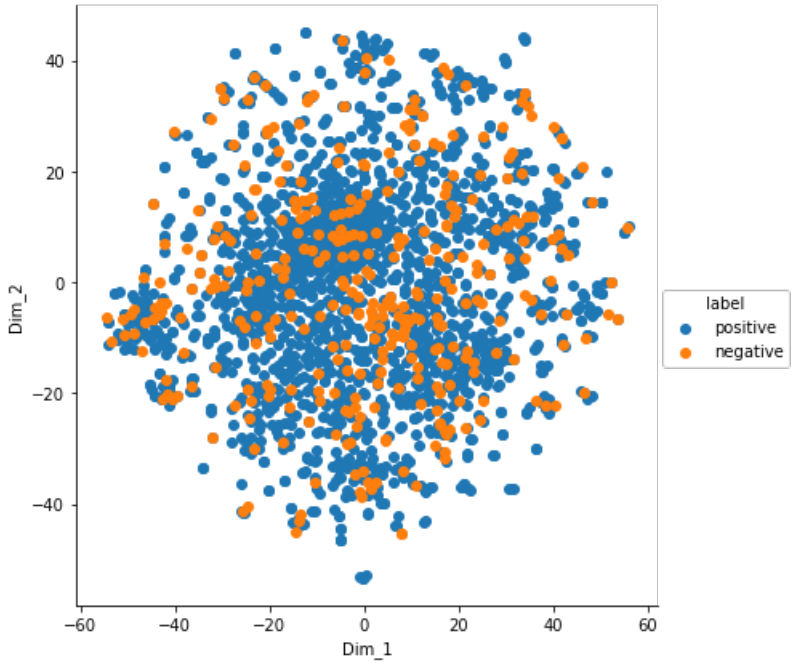
```
 2.45677696e-02,  6.86592745e-02,  1.49490469e-02,  6.51749694e-02,
 9.85189676e-02,  1.29295373e-02, -3.32962833e-02,  2.46775582e-01,
 2.88242865e-02,  8.99391174e-02,  4.62419875e-02,  4.94736433e-02,
-1.12126365e-01, -1.07689034e-02,  1.29506616e-02, -4.76543307e-02,
-5.86799271e-02, -1.27620161e-01, -2.82129049e-01,  9.51365903e-02,
-6.62747473e-02,  1.76300377e-01, -6.64312243e-02,  9.59163681e-02,
 1.34712264e-01, -1.45000979e-01, -6.55070096e-02,  1.02417665e-02,
 3.99509518e-05, -1.42186925e-01, -2.79008038e-02, -3.42702121e-02,
-2.49737259e-02,  1.27951816e-01, -6.97588697e-02,  1.79184943e-01,
-1.23403333e-01,  2.00367540e-01], dtype=float32)]
```

In [28]:

```
#w2vec --> TSNE
from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']

data_2000 = tokens[:2000]
labels_2000 = lable[0:2000]

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

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[28]:

```
<seaborn.axisgrid.FacetGrid at 0x20732021978>
```



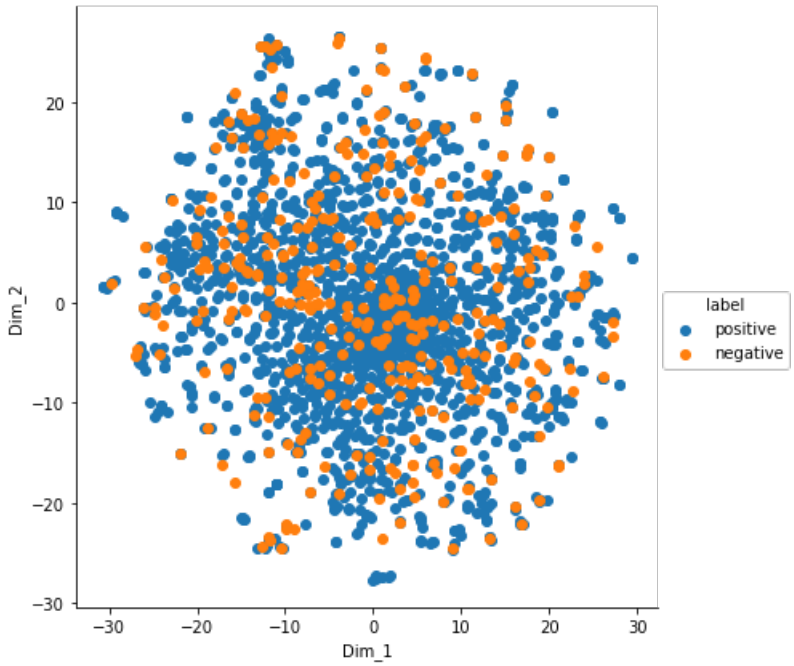
In [29]:

```
#w2vec --> TSNE
#with change in perplexity = 80
from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']
```

```
data_2000 = tokens[:2000]
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity = 80, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30 --> with change in perplexity = 80
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[29]:

```
<seaborn.axisgrid.FacetGrid at 0x2079d8bbba8>
```



In [31]:

```
#avg-w2v
labels = []
tokens = []

for word in w2v_model.wv.vocab:
    tokens.append(w2v_model[word])
    labels.append(word)

# labels
tokens[:2]
```

Out[31]:

```
[array([ 1.27351597e-01,  3.12062562e-01,  8.42558742e-02,  1.96281020e-02,
         9.54858959e-02, -7.91134760e-02,  1.59643039e-01, -4.40884978e-02,
         1.52912870e-01,  3.43161523e-02, -2.90893354e-02, -1.98353231e-01,
         2.45877896e-02,  6.88392743e-02,  1.49498489e-02,  6.51749894e-02,
         9.85189676e-02,  1.29295373e-02, -3.32962833e-02,  2.46775582e-01,
         2.88242865e-02,  8.99391174e-02,  4.62419875e-02,  4.94736433e-02,
        -1.12126365e-01, -1.07689034e-02,  1.29506616e-02, -4.76543307e-02,
        -5.86799271e-02, -1.27620161e-01, -2.82129049e-01,  9.51365903e-02,
        -6.62747473e-02,  1.76300377e-01, -6.64312243e-02,  9.59163681e-02,
         1.34712264e-01, -1.45000979e-01, -6.55070096e-02,  1.02417665e-02,
         3.99509518e-05, -1.42186925e-01, -2.79008038e-02, -3.42702121e-02,
```

```
        -2.49737259e-02,  1.27951816e-01, -6.97588697e-02,  1.79184943e-01,
        -1.23403333e-01,  2.00367540e-01], dtype=float32),
 array([-1.0391155 ,  0.27242482,  1.9437786 ,  0.0428596 , -2.3075078 ,
        -1.2287023 , -0.18617941,  2.852096  , -0.28995928,  3.9036295 ,
        -2.4604146 ,  2.5605137 ,  0.22095278,  1.5730187 , -0.06471132,
         0.6129404 ,  0.32757914, -1.4311916 ,  0.354883  , -1.160348  ,
         0.01831347,  0.5074931 , -0.95592993,  0.3138117 ,  1.3921626 ,
         1.0225055 ,  0.7316783 , -2.0178227 ,  1.0580341 ,  0.14533265,
         1.4072881 ,  2.6587842 , -3.808838  , -0.6149528 , -0.4854196 ,
         0.0751783 ,  1.5252509 , -0.48867682,  1.3510249 ,  1.4553956 ,
        -0.48789185, -0.15812382, -0.6785673 ,  2.1216366 , -1.8864447 ,
        -0.07575194, -1.4769679 , -3.988311  , -0.9321874 ,  0.5879292 ],
       dtype=float32)]
```

In [32]:

```
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])
# 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]))
```

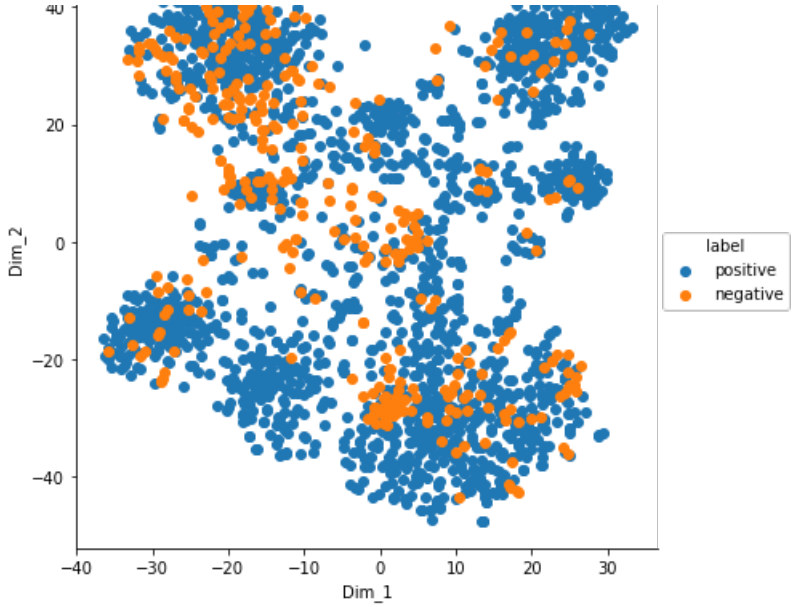
```
364171
50
```

In [33]:

```
#avg-w2v

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']

data_2000 = sent_vectors[:2000]
labels_2000 = lable[0:2000]

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

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[33]:

```
<seaborn.axisgrid.FacetGrid at 0x2079d341dd8>
```

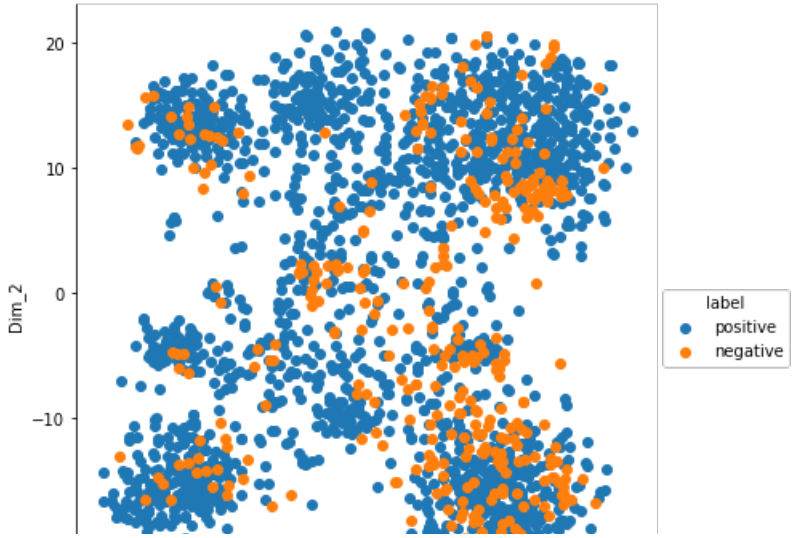


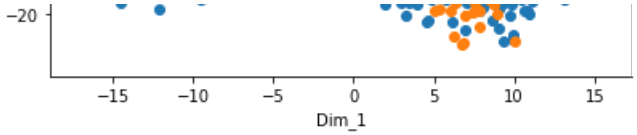
In [34]:

```
#avg-w2v
#with change in perplexity = 80

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']

data_2000 = sent_vectors[:2000]
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity = 80, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30   ->>with change in perplexity = 80
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[34]:

```
<seaborn.axisgrid.FacetGrid at 0x2079d3cacf8>
```





In [35]:

```
# 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: # 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

print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors[0]))
```

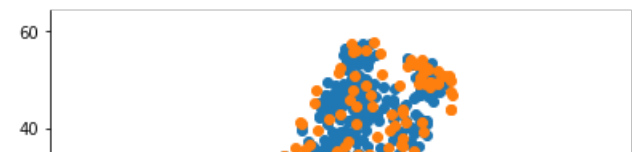
```
364171
50
```

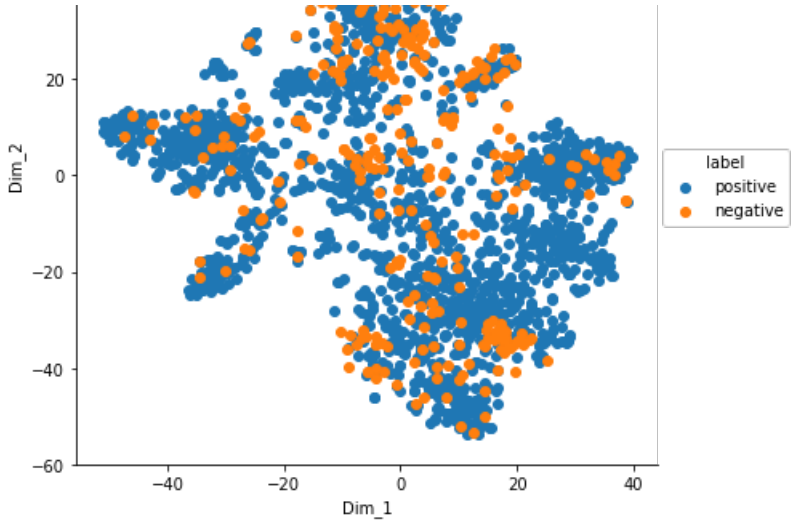
In [36]:

```
#TF-IDF weighted Word2Vec -> TSNE

from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']

data_2000 = tfidf_sent_vectors[:2000]
labels_2000 = lable[0:2000]

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

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[36]:

```
<seaborn.axisgrid.FacetGrid at 0x207a8e686a0>
```



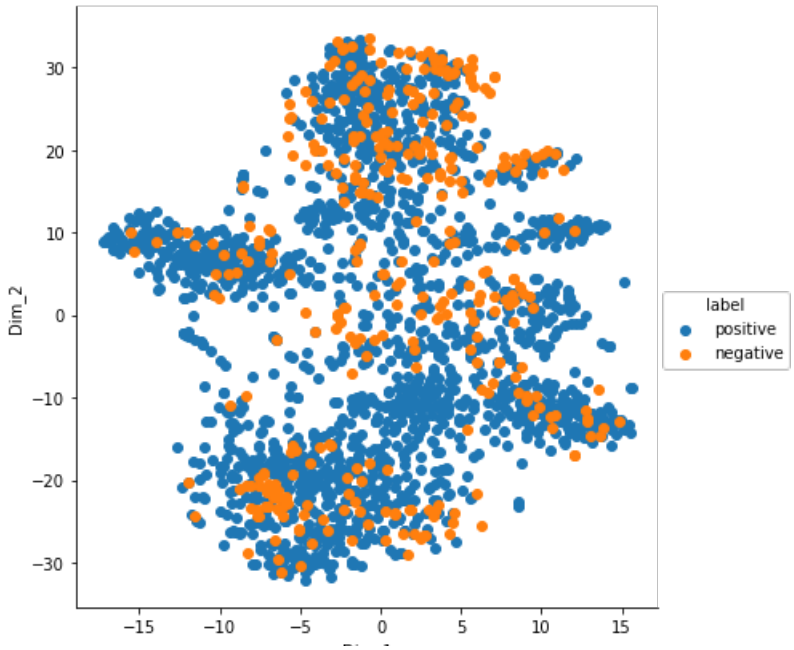


In [37]:

```
#TF-IDF weighted Word2Vec -> TSNE
#with change in perplexity = 80
from sklearn.manifold import TSNE
import seaborn as sn
lable= final['Score']

data_2000 = tfidf_sent_vectors[:2000]
labels_2000 = lable[0:2000]

model = TSNE(n_components=2,perplexity = 80, random_state=0)
# configuring the parameteres
# the number of components = 2
# default perplexity = 30 ---> with change in perplexity = 80
# default learning rate = 200
# default Maximum number of iterations for the optimization = 1000

tsne_data = model.fit_transform(data_2000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_2000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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

Out[37]:

```
<seaborn.axisgrid.FacetGrid at 0x207346e63c8>
```




Dim_1