In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import nltk

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

import sqlite3
import re
import string
```

In [2]:
```
# Connecting to already Processed data

con=sqlite3.connect('final.sqlite')

# Fetching all Positive and Negetive reviews seperately as our dataset is an imbalanced dataset
fin_data_pos=pd.read_sql_query(''' select * from Reviews where Score==1 ''',con)
fin_data_neg=pd.read_sql_query(''' select * from Reviews where Score==0 ''',con)

print(fin_data_pos.shape)
print(fin_data_neg.shape)
```

```
(307061, 13)
(57110, 13)
```

In [3]:
```
fin_data=[]

# Fetching first 1500 records from each kind of reviews to make final dataset as balanced
# we are doing this becuase if we took random data from an unbalanced dataset then there will be a chance of getting
# all positive reviews as they are more in dataset. So, we are making this move here.
fin_data_pos_500=fin_data_pos[0:1500]
fin_data_neg_500=fin_data_neg[0:1500]
fin_data=pd.concat([fin_data_pos_500,fin_data_neg_500])
print(fin_data.shape)
```

```
(3000, 13)
```

In [4]:
```
#Now we have to seperate our Score colun that contain transformed alues of rating in dataset.
#S, that we can add this to dataframe after performing all transformations on data.
score=fin_data['Score']

print(type(score),'\n')
score=np.array(score) #Changing from Series datatype to array becuase we will add this to an array later.
print('After conversion',type(score))
```

```
<class 'pandas.core.series.Series'>

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

BOW

In [10]:
```
cv_model= CountVectorizer()

final_bow=cv_model.fit_transform(fin_data['CleanedText'].values)

print('The type of final_bow is ',type(final_bow))
print('Shape of final bag of words ',final_bow.get_shape())
print('Number of unique words in the bag are ',final_bow.get_shape()[1])
```

```
The type of final_bow is  <class 'scipy.sparse.csr.csr_matrix'>
Shape of final bag of words  (3000, 8580)
Number of unique words in the bag are  8580
```

In [11]:
```
# As our bow output is of type scipy.sparse.csr_matrix we have to convert it into type of array and,
# then we have to apply tSNE on top of it
final_bow_arr=final_bow.toarray()
print(type(final_bow_arr))

print(final_bow_arr.shape)
```

```
<class 'numpy.ndarray'>
(3000, 8580)
```

Applying tSNE on BOW output

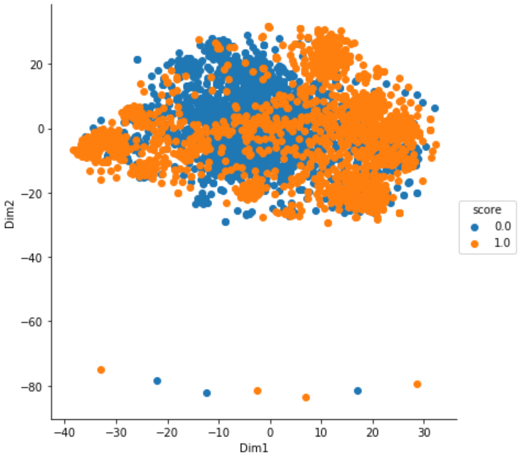
In [37]:
```
from sklearn.manifold import TSNE

model=TSNE(n_components=2,random_state=0)

bow_tsne=model.fit_transform(final_bow_arr)
```

```
<class 'pandas.core.series.Series'>
```

In [40]:

```
bow_tsne_mod=np.vstack((bow_tsne.T,score)).T

print(bow_tsne_mod.shape)
bow_df= pd.DataFrame(bow_tsne_mod,columns=('Dim1','Dim2','score'))

sn.FacetGrid(bow_df, hue="score", size=6).map(plt.scatter, 'Dim1', 'Dim2').add_legend()
plt.show()
```

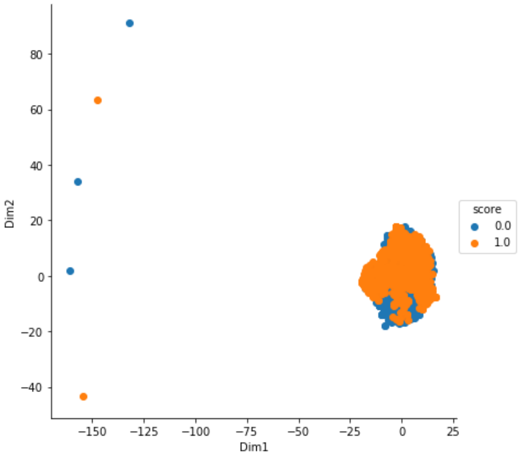
```
(3000, 3)
```



In [13]:

```
# With increased perplexity from 30 to 50 and also iterations increased from 1000 t0 5000 and learning rate from 200 to 600.

from sklearn.manifold import TSNE

model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000,learning_rate=600.0)

bow_tsne=model.fit_transform(final_bow_arr)

bow_tsne_mod=np.vstack((bow_tsne.T,score)).T

print(bow_tsne_mod.shape)
bow_df= pd.DataFrame(bow_tsne_mod,columns=('Dim1','Dim2','score'))

sn.FacetGrid(bow_df, hue="score", size=6).map(plt.scatter, 'Dim1', 'Dim2').add_legend()
plt.show()
```

```
(3000, 3)
```



TFIDF implementation

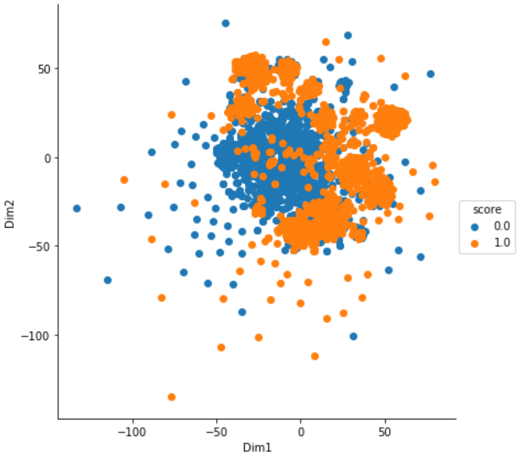
In [5]:

```
#Applying TFIDF on already filtered 3000 records/documents sized Corpus

tfidf_model=TfidfVectorizer(ngram_range=(1,2))

tfidf_data=tfidf_model.fit_transform(fin_data['CleanedText'].values)

print('Data type of tfidf_data is ',type(tfidf_data))

print('\n Number of words in tfidf_data is ',tfidf_data.get_shape()[1])
```

```
Data type of tfidf_data is  <class 'scipy.sparse.csr.csr_matrix'>

 Number of words in tfidf_data is  105115
```

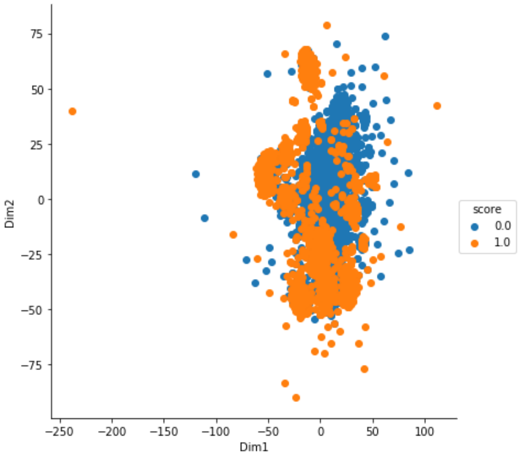
In [17]:

```
#converting scipy matrix to array
tfidf_data_arr=tfidf_data.toarray()

#Applying TSNE on tfidf result
tsne_tfidf_model=TSNE(n_components=2,random_state=0,perplexity=30,n_iter=2000)

tsne_tfidf_data=tsne_tfidf_model.fit_transform(tfidf_data_arr)

final_tfidf=np.vstack((tsne_tfidf_data.T,score)).T

tfidf_df=pd.DataFrame(final_tfidf,columns=('Dim1','Dim2','score'))

sn.FacetGrid(tfidf_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()

plt.show()
```



In [7]:

```
#converting scipy matrix to array
tfidf_data_arr=tfidf_data.toarray()

from sklearn.manifold import TSNE

#Applying tSNE on Tfidf with icreased perplexity to 50 from 30, number of iterations from 2k to 5k
tsne_tfidf_model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)

tsne_tfidf_data=tsne_tfidf_model.fit_transform(tfidf_data_arr)

final_tfidf=np.vstack((tsne_tfidf_data.T,score)).T

tfidf_df=pd.DataFrame(final_tfidf,columns=('Dim1','Dim2','score'))

sn.FacetGrid(tfidf_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()

plt.show()
```



Avg W2V

In [8]:

```
list_of_items=[]

#We are splitting each record in the corpus into individual words and
#we are appending all those words from all records into a list
for item in fin_data['CleanedText'].values:
    list_of_items.append(item.split())
```

In [20]:
```
from gensim.models import Word2Vec
from tqdm import tqdm
from gensim.models import KeyedVectors
import os
import pickle


w2v= Word2Vec(list_of_items,min_count=4,size=50,workers=4)
#min_count - minimum frequency of a word in the Corpus to consider
#size - no of neighbours to be consider in a cluster.
#workers - No of threads to be perform in the backend.


# Storing final words filtered from Word2vec as a list.
w2v_words=list(w2v.wv.vocab)


print('The totatl no. of words in corpus is ',len(w2v_words))
```

```
The totatl no. of words in corpus is  3284
```

In [21]:
```
w2v.wv.most_similar('good') #Displays all the words in the corpus that are similar to 'Good'
```

Out[21]:
```
[('great', 0.9989895820617676),
 ('better', 0.9987933039665222),
 ('flavor', 0.9987620115280151),
 ('much', 0.9983972311019897),
 ('natur', 0.9980062246322632),
 ('meat', 0.9979257583618164),
 ('grain', 0.9979186654090881),
 ('stuff', 0.9978964328765869),
 ('product', 0.9977508783340454),
 ('expens', 0.997738242149353)]
```

In [22]:
```
w2v.wv.most_similar('well')
```

Out[22]:
```
[('natur', 0.9993279576301575),
 ('varieti', 0.9992753267288208),
 ('wet', 0.9991913437843323),
 ('turkey', 0.9991177916526794),
 ('healthi', 0.9989650845527649),
 ('meat', 0.9989104270935059),
 ('purina', 0.9988874197006226),
 ('liver', 0.9988400936126709),
 ('adult', 0.9988037347793579),
 ('beef', 0.9987547397613525)]
```

In [27]:
```
# 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_items): # 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.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]))
print(type(sent_vectors))
```

```
100%|████████████████████████████████████| 3000/3000 [00:08<00:00, 347.98it/s]

3000
50
<class 'list'>
```

In [31]:
```
sent_vectors=np.array(sent_vectors)
print(type(sent_vectors))
print('\n',sent_vectors.shape)
```

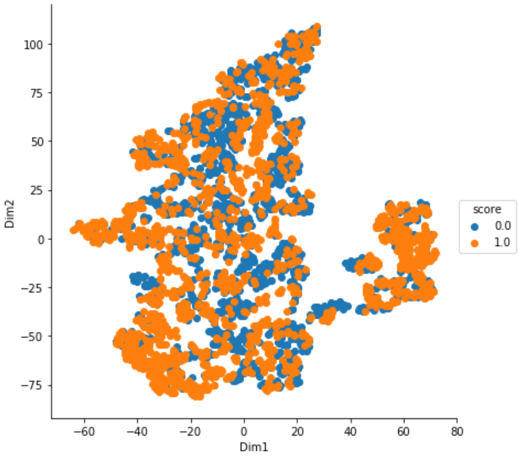
```
<class 'numpy.ndarray'>

 (3000, 50)
```

Applying tSNE on AvgW2V

In [32]:
```
from sklearn.manifold import TSNE

avg_tsne_model=TSNE(n_components=2,random_state=0,perplexity=30,n_iter=3000)

avg_tsne=avg_tsne_model.fit_transform(sent_vectors)
```

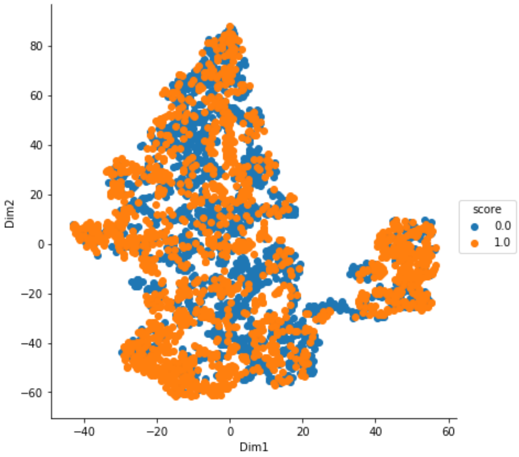
In [34]:

```
final_avgw2v=np.vstack((avg_tsne.T,score)).T

avgw2v_df=pd.DataFrame(final_avgw2v,columns=('Dim1','Dim2','score'))

sn.FacetGrid(avgw2v_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()


plt.show()
```



In [35]:

```
from sklearn.manifold import TSNE

#Applying tSNE on AvgW2v with increased perplexity(30 -> 50), no. of iterations(1000 to 5000)
avg_tsne_model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000)

avg_tsne=avg_tsne_model.fit_transform(sent_vectors)

final_avgw2v=np.vstack((avg_tsne.T,score)).T

avgw2v_df=pd.DataFrame(final_avgw2v,columns=('Dim1','Dim2','score'))

sn.FacetGrid(avgw2v_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()

plt.show()
```



Tfidf weighted word 2 vec

In [36]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_model=TfidfVectorizer()

tfidf_data_model=tfidf_model.fit_transform(fin_data['CleanedText'].values)

#Converting tfidf value into dictionary with its word name as Key and tfidf value as value.
tfidf_dict=dict(zip(tfidf_model.get_feature_names(),list(tfidf_model.idf_)))
```

```
In [38]: # TF-IDF weighted Word2Vec
         tfidf_feat = tfidf_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_items): # 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:
                     vec = w2v.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 = tfidf_dict[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%|█████████████████████████████████████| 3000/3000 [00:11<00:00, 250.47it/s]
```

Applying tSNE on Tfidf weighted W2V

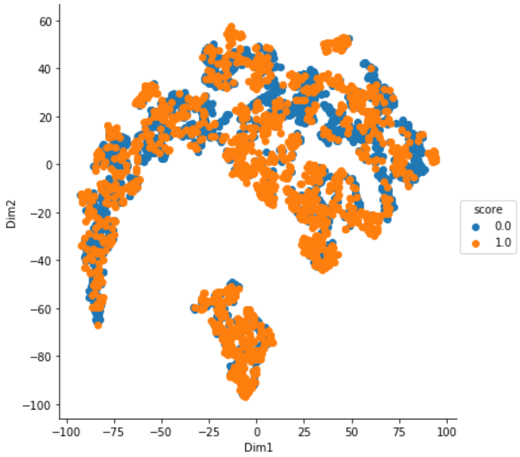
```
In [40]: from sklearn.manifold import TSNE
```

```
In [46]: print(type(tfidf_sent_vectors))

         tfidf_sent_arr= np.array(tfidf_sent_vectors) #Converting datatype of tfidfw2v values from list to array

         print('\n',type(tfidf_sent_arr))
```

```
<class 'list'>

 <class 'numpy.ndarray'>
```

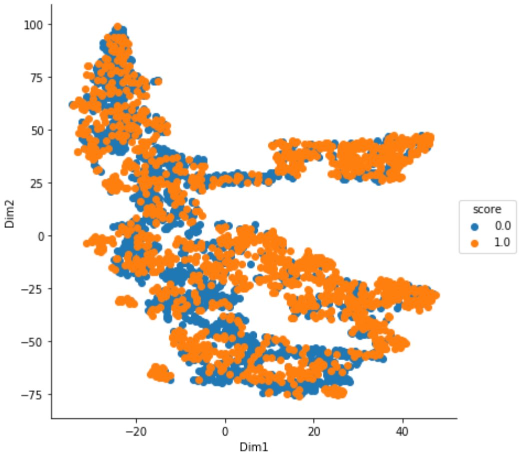
```
In [48]: tsne_model=TSNE(n_components=2,random_state=0,perplexity=30,n_iter=3000,learning_rate=500)

         tfidf_sent=tsne_model.fit_transform(tfidf_sent_arr)# Applying tSNE on tfidfw2v values

         fin_tfidfw2v=np.vstack((tfidf_sent.T,score)).T

         tfidfw2v_df=pd.DataFrame(fin_tfidfw2v,columns=('Dim1','Dim2','score'))
```

```
In [49]: sn.FacetGrid(tfidfw2v_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()

         plt.show()
```



In [50]:

```
#Applying tSNE on tfidfw2v with increased perplexity  and no of iterations and learning rate.
tsne_model=TSNE(n_components=2,random_state=0,perplexity=50,n_iter=5000,learning_rate=500)

tfidf_sent=tsne_model.fit_transform(tfidf_sent_arr)

fin_tfidfw2v=np.vstack((tfidf_sent.T,score)).T

tfidfw2v_df=pd.DataFrame(fin_tfidfw2v,columns=('Dim1','Dim2','score'))

sn.FacetGrid(tfidfw2v_df,hue='score',size=6).map(plt.scatter,'Dim1','Dim2').add_legend()

plt.show()
```


Dim2
Dim1
score
● 0.0
● 1.0