[1] Amazon Fine Food Reviews Analysis

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

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).

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

from tqdm import tqdm
import os
#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

warnings.filterwarnings("ignore")

%matplotlib inline
# sets the backend of matplotlib to the 'inline' backend:
#With this backend, the output of plotting commands is displayed inline within frontends li
#directly below the code cell that produced it. The resulting plots will then also be store

#Functions to save objects for later use and retireve it
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

In [2]:

```
#Using sqlite3 to retrieve data from sqlite file

con = sqlite3.connect("totally_processed_DB.sqlite")#Loading Cleaned/ Preprocesed text that

#Using pandas functions to query from sql table
final = pd.read_sql_query("""
SELECT * FROM Reviews
""",con)

#Reviews is the name of the table given
#Taking only the data where score != 3 as score 3 will be neutral and it won't help us much
final.head()
```

2	2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1
3	3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1
4	4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3

In [7]:

```
final.shape
final['Score'].size
```

Out[7]:

```
25000
```

In [8]:

```
#Taking Sample Data
n_samples = 25000
final = final.sample(n_samples)

###Sorting as we want according to time series
final.sort_values('Time',inplace=True)
final.head(10)
```

Out[8]:

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNu
423	423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	
844	844	138000	149768	B00004S1C5	A7P76IGRZZBFJ	E. Thompson "Soooooper Genius"	
270	270	346140	374449	B00004CI84	A3K3YJWV0N54ZO	Joey	
1064	1064	443663	479724	B00005U2FA	A1B226ZPOE0KSZ	Jack Richman	
1116	1116	137932	149700	B00006L2ZT	A19JWUIRF6DXLV	Andrew J Monzon	
251	251	346111	374417	B00004CI84	A23QOAXJSWIBS6	Daniel S. Russell "syzygy121"	
5289	5289	474609	513274	B0000GGHNI	A59SBCWOLJJ16	S. Reeve	
6381	6381	409953	443373	B0000U1OFU	AFKKVFJ2DS4EL	Jonathan R. Pauling	
203019	203019	124630	135144	B001KWK1N8	A3FS8HDE2BTD5Z	C. Boeck "cebii"	
6956	6956	39671	43130	B0000W2SZS	A2BETN6Y2DEFZ1	Catnip	

In [5]:

```
savetofile(final,"sample_25000_knn")
```

In [6]:

```
final = openfromfile("sample_25000_knn")
```

[7.2.2] Bag of Words (BoW)

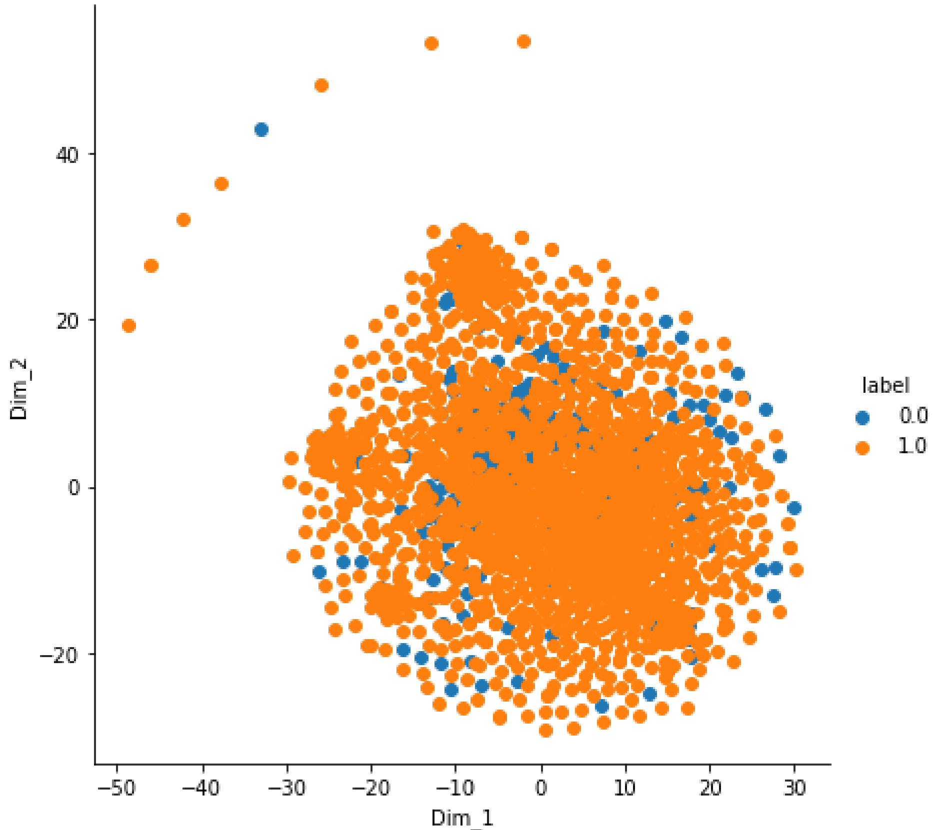
In [11]:

```
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  (25000, 19579)
the number of unique words  19579
```

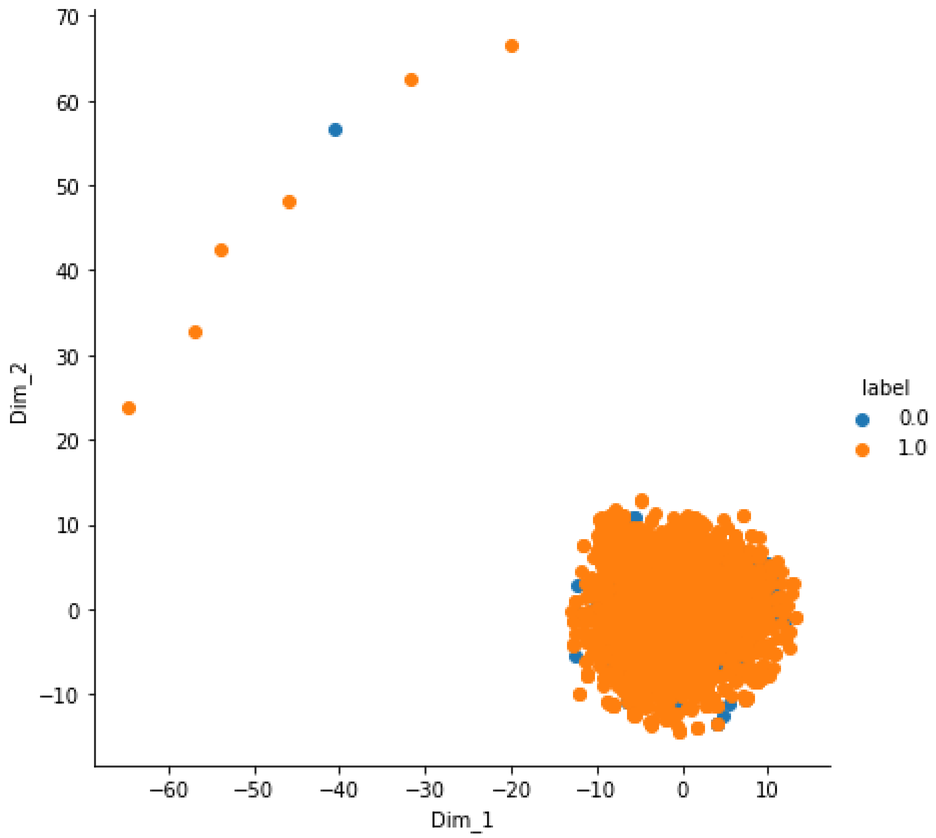
In [12]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_counts[0:2000,:].todense()
labels_2000 = final["Score"][0:2000]

model = TSNE(n_components=2, random_state=0,perplexity = 20,n_iter=500,)
# 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.title('With perplexity = 50')
plt.show()
```



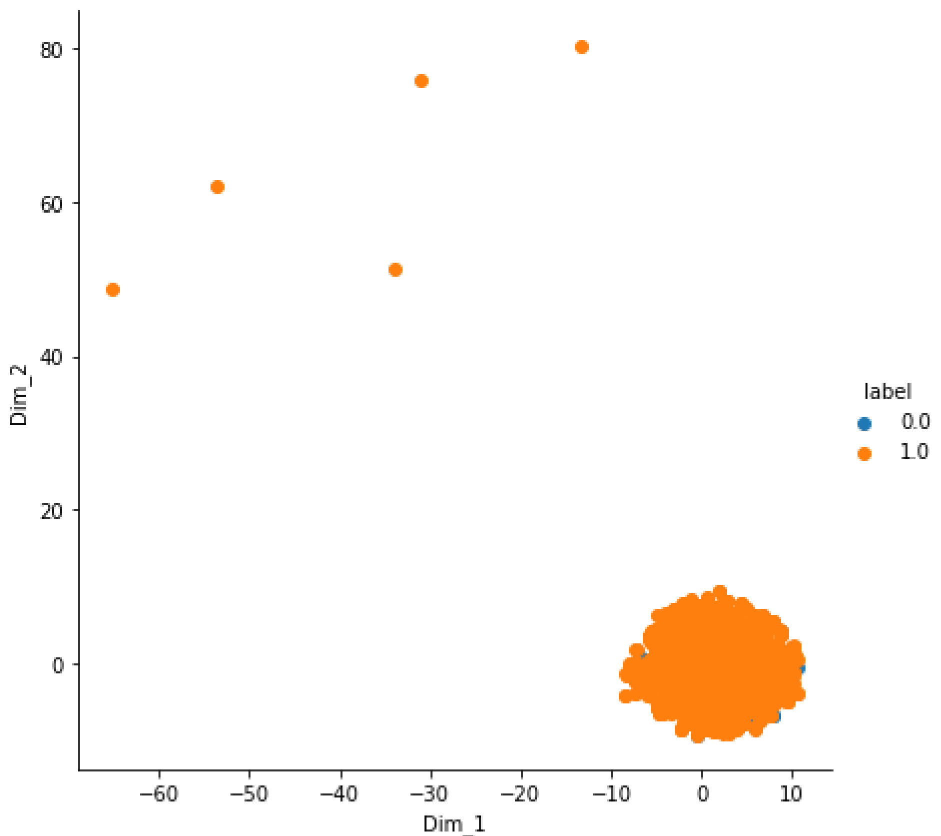
In [13]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_counts[0:2000,:].todense()
labels_2000 = final["Score"][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.title('With perplexity = 50')
plt.show()
```



In [14]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_counts[0:2000,:].todense()
labels_2000 = final["Score"][0:2000]

model = TSNE(n_components=2, random_state=0,perplexity = 40,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(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.title('With perplexity = 50')
plt.show()
```



[7.2.5] TF-IDF

In [22]:

```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(final['CleanedText'].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_idf.get_s
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (25000, 19579)
the number of unique words including both unigrams and bigrams  19579
```

In [23]:

```
features = tf_idf_vect.get_feature_names()
print("some sample features(unique words in the corpus)",features[1000:1010])
```

```
some sample features(unique words in the corpus) ['assimil', 'assist', 'asso
ci', 'assort', 'asst', 'assuag', 'assum', 'assumpt', 'assur', 'asterisk']
```

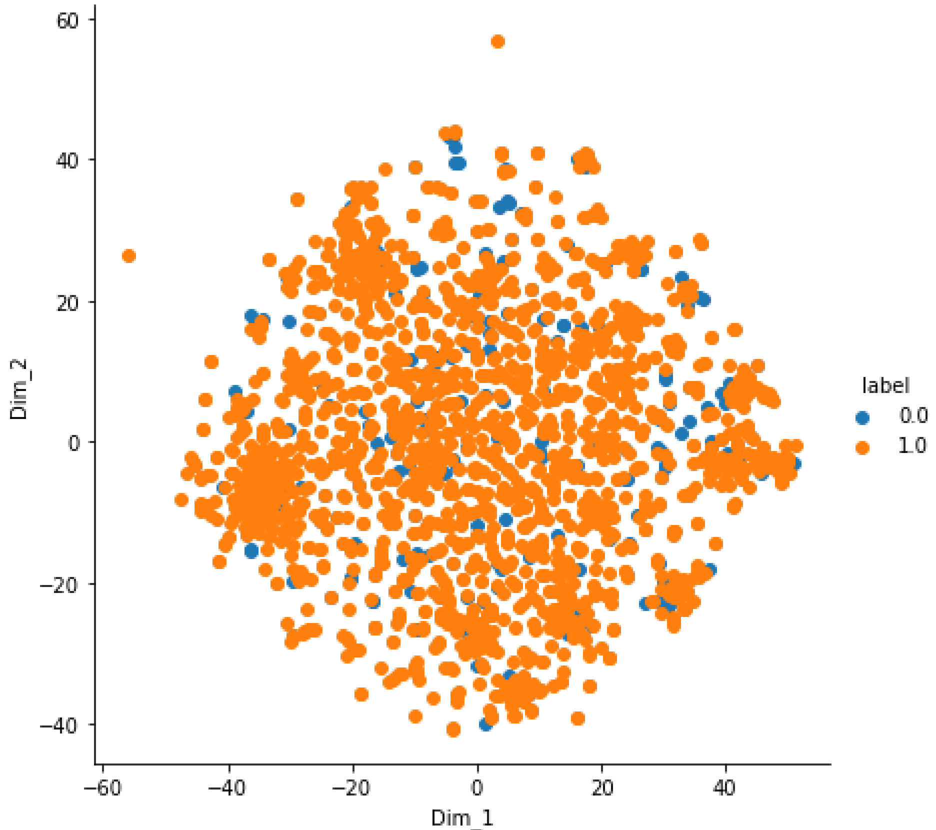
In [24]:

```
# source: https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature name
    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)
```

top_tfidf

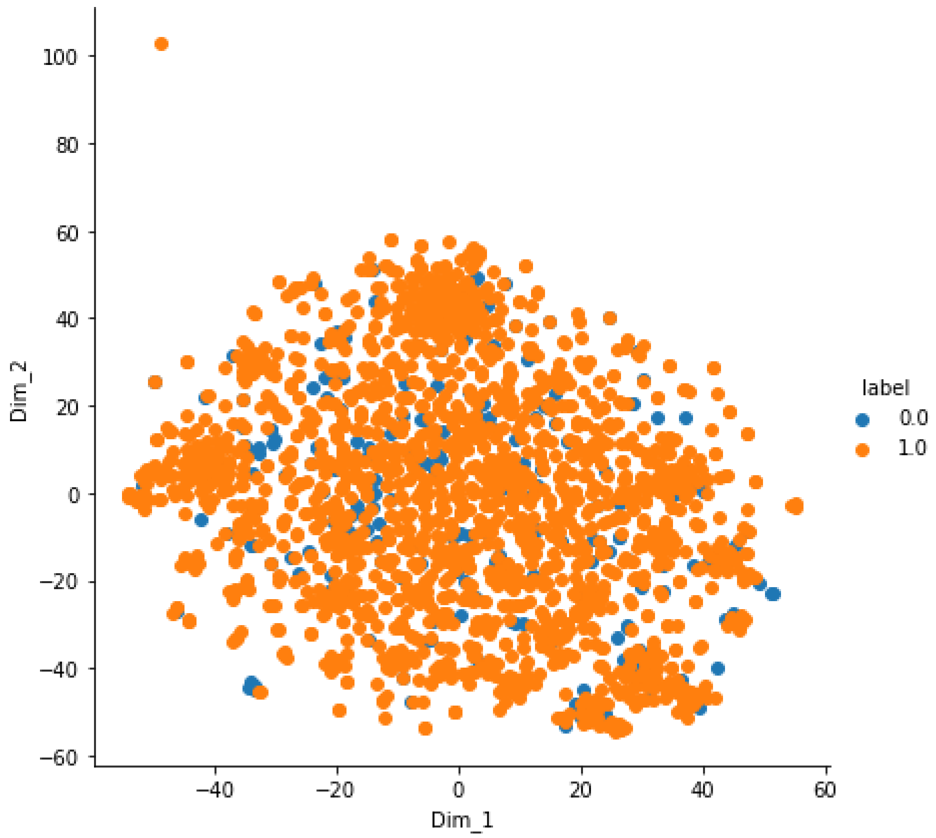
In [25]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_tf_idf[0:2000,:].todense()
labels_2000 = final["Score"][0:2000]
model = TSNE(n_components=2, random_state=0,perplexity = 20,n_iter=500,)
# 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.title('With perplexity = 50')
plt.show()
```



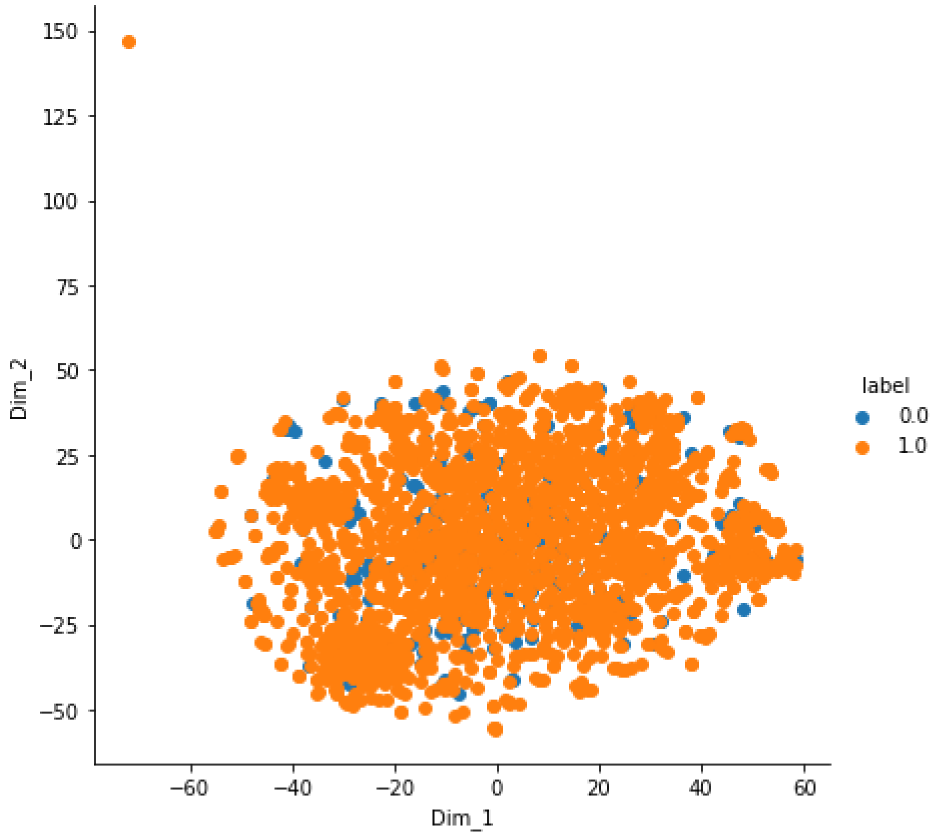
In [26]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_tf_idf[0:2000,:].todense()
labels_2000 = final["Score"][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.title('With perplexity = 50')
plt.show()
```



In [27]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 2000 points as TSNE takes a lot of time for 364K points
data_2000 = final_tf_idf[0:2000,:].todense()
labels_2000 = final["Score"][0:2000]
model = TSNE(n_components=2, random_state=0,perplexity = 40,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(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.title('With perplexity = 50')
plt.show()
```



[7.2.6] Word2Vec

In [28]:

```
# 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())
```

In [29]:

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

```
simpli put beetlejuic funniest comedi kind sinc ghostbust michael keaton pla
y titl charact fun love ghost like mischief beetlejuic call coupl davi baldw
in get rid peopl live thier hous
*****************************************************************
['simpli', 'put', 'beetlejuic', 'funniest', 'comedi', 'kind', 'sinc', 'ghost
bust', 'michael', 'keaton', 'play', 'titl', 'charact', 'fun', 'love', 'ghos
t', 'like', 'mischief', 'beetlejuic', 'call', 'coupl', 'davi', 'baldwin', 'g
et', 'rid', 'peopl', 'live', 'thier', 'hous']
```

In [30]:

```
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [31]:

```
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  6847
sample words  ['simpli', 'put', 'beetlejuic', 'kind', 'sinc', 'michael', 'ke
aton', 'play', 'titl', 'charact', 'fun', 'love', 'ghost', 'like', 'call', 'c
oupl', 'get', 'rid', 'peopl', 'live', 'thier', 'hous', 'use', 'previous', 'm
odel', 'year', 'happi', 'one', 'downsid', 'earlier', 'offer', 'wine', 'love
r', 'never', 'knew', 'vacuum', 'achiev', 'make', 'sure', 'damag', 'oxygen',
'bottl', 'got', 'napa', 'month', 'ago', 'glad', 'not', 'deliv', 'promis']
```

In [32]:

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

Out[32]:

```
[('yummi', 0.8532200455665588),
 ('delici', 0.8373256921768188),
 ('satisfi', 0.8049272298812866),
 ('hearti', 0.7947583794593811),
 ('crunch', 0.7691254019737244),
 ('nutriti', 0.7381624579429626),
 ('crunchi', 0.7379350662231445),
 ('dessert', 0.7163292169570923),
 ('dens', 0.7141882181167603),
 ('incred', 0.7132158279418945)]
```

In [33]:

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

Out[33]:

```
[('prefer', 0.6960845589637756),
 ('weird', 0.6875526905059814),
 ('aw', 0.677577793598175),
 ('appeal', 0.6592475175857544),
 ('funni', 0.6556658148765564),
 ('terribl', 0.6500604152679443),
 ('odd', 0.6400692462921143),
 ('dislik', 0.6374333500862122),
 ('real', 0.6365963220596313),
 ('nasti', 0.6296987533569336)]
```

[7.2.7] Avg W2V, TFIDF-W2V

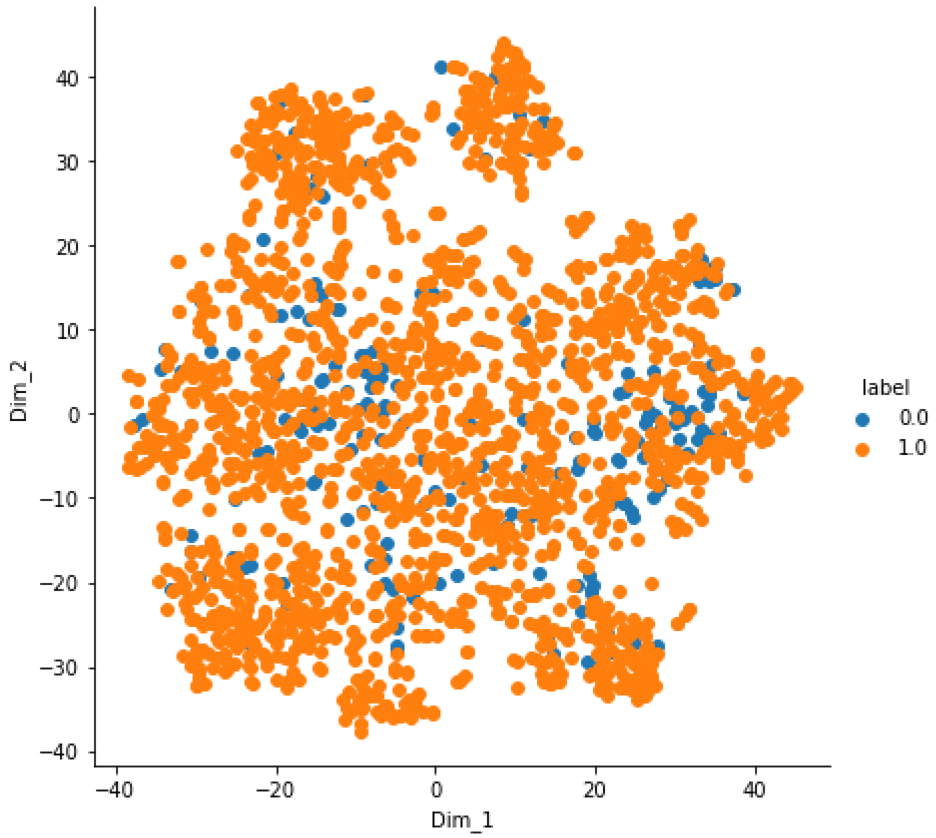
In [34]:

```
# 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_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]))
```

```
100%|██████████| 25000/25000 [01:14<00:00, 337.24it/s]

25000
50
```

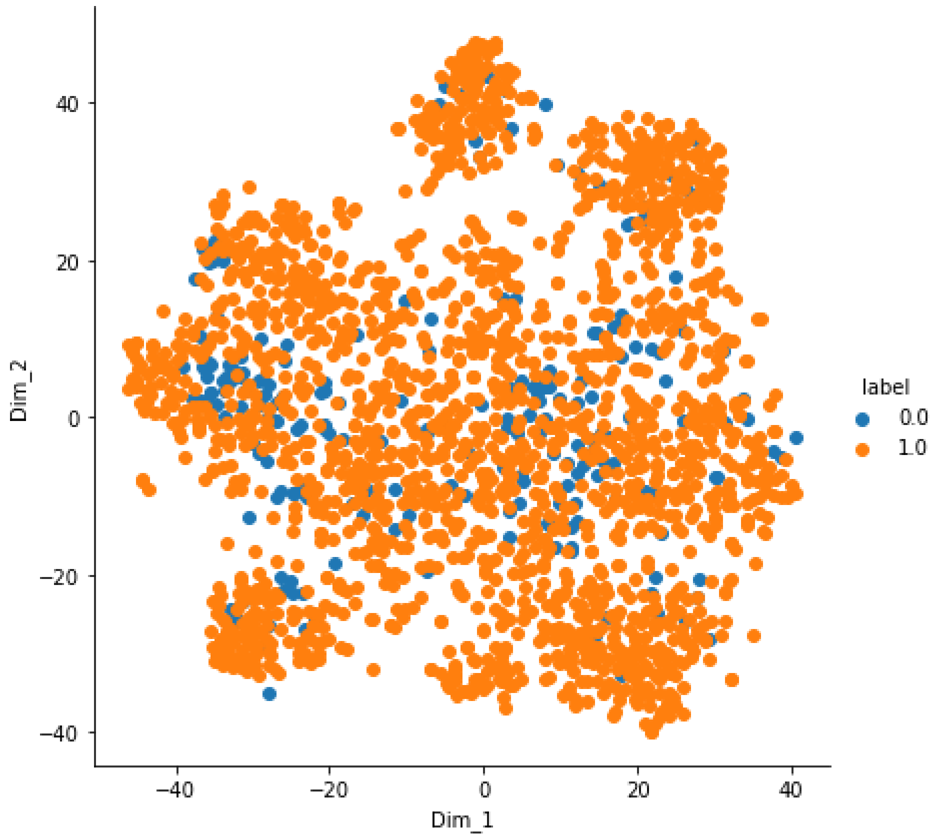
In [35]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_2000 = sent_vectors[0:2000]
labels_2000 = final["Score"][0:2000]

TSNE_model = TSNE(n_components=2, random_state=0,perplexity = 20,n_iter=500,)
# 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 = TSNE_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.title('With perplexity = 50')
plt.show()
```



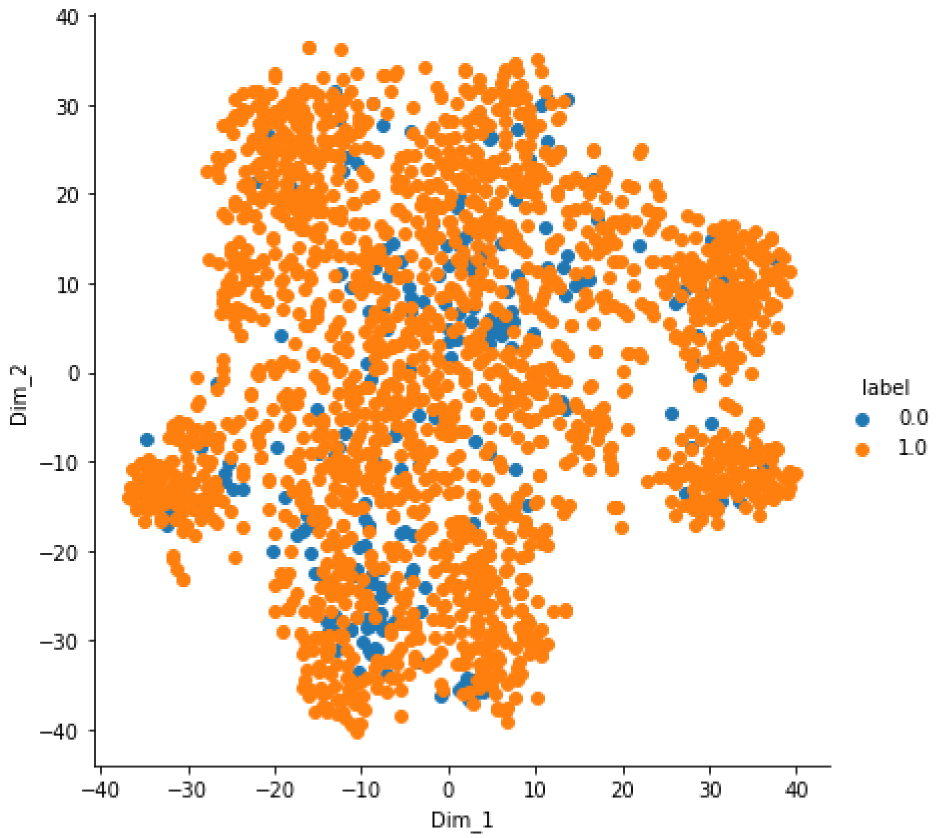
In [36]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_2000 = sent_vectors[0:2000]
labels_2000 = final["Score"][0:2000]

TSNE_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 = TSNE_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.title('With perplexity = 50')
plt.show()
```



In [37]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_2000 = sent_vectors[0:2000]
labels_2000 = final["Score"][0:2000]

TSNE_model = TSNE(n_components=2, random_state=0,perplexity = 40,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 = TSNE_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.title('With perplexity = 50')
plt.show()
```



In [38]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# 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_)))
```

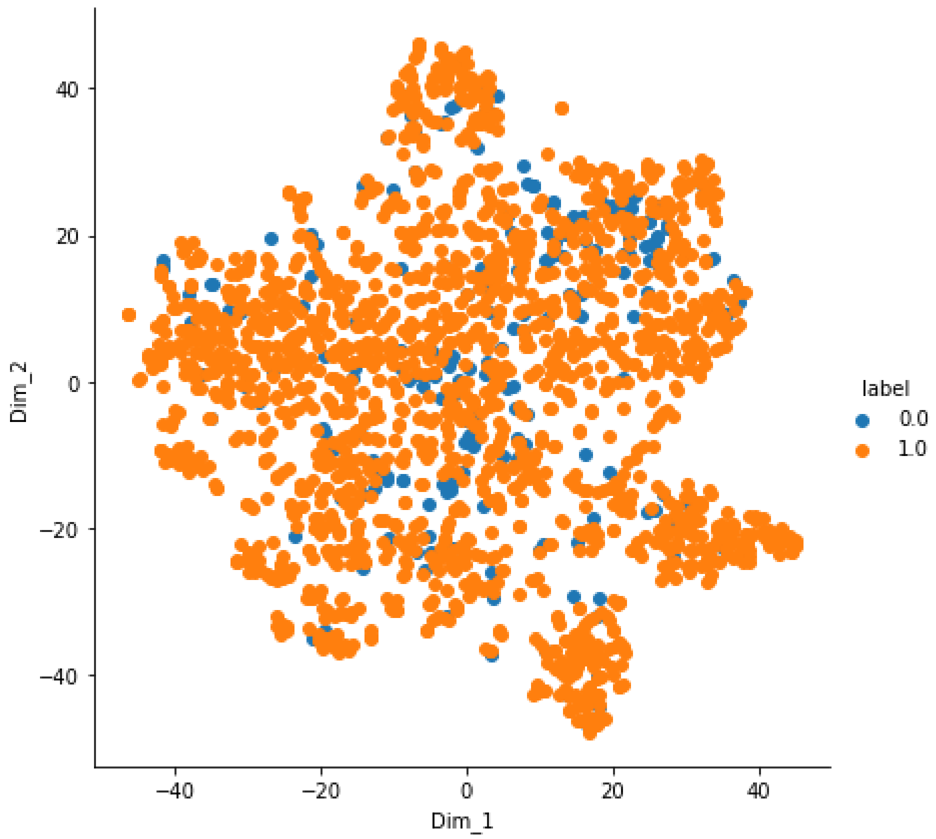
In [39]:

```
# 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_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
        if word in w2v_words:
            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%|██████████| 25000/25000 [00:47<00:00, 528.52it/s]
```

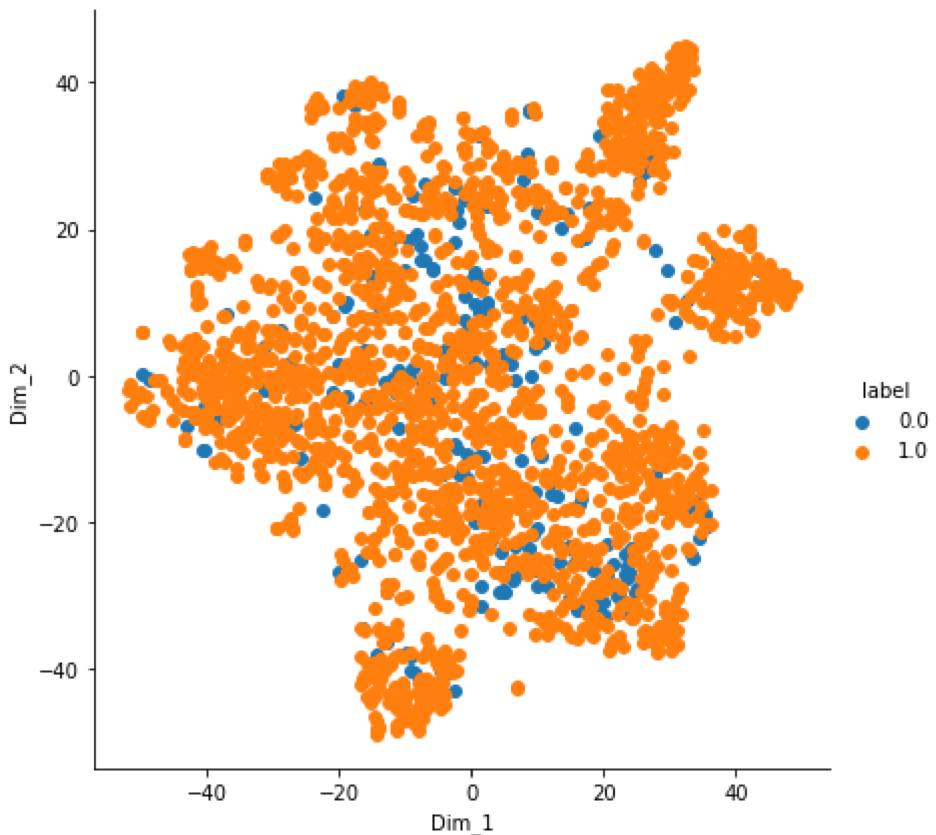
In [40]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_1000 = tfidf_sent_vectors[0:2000]
labels_1000 = final["Score"][0:2000]

TSNE_model = TSNE(n_components=2, random_state=0,perplexity = 20,n_iter=500,)
# 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 = TSNE_model.fit_transform(data_1000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).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.title('With perplexity = 50')
plt.show()
```



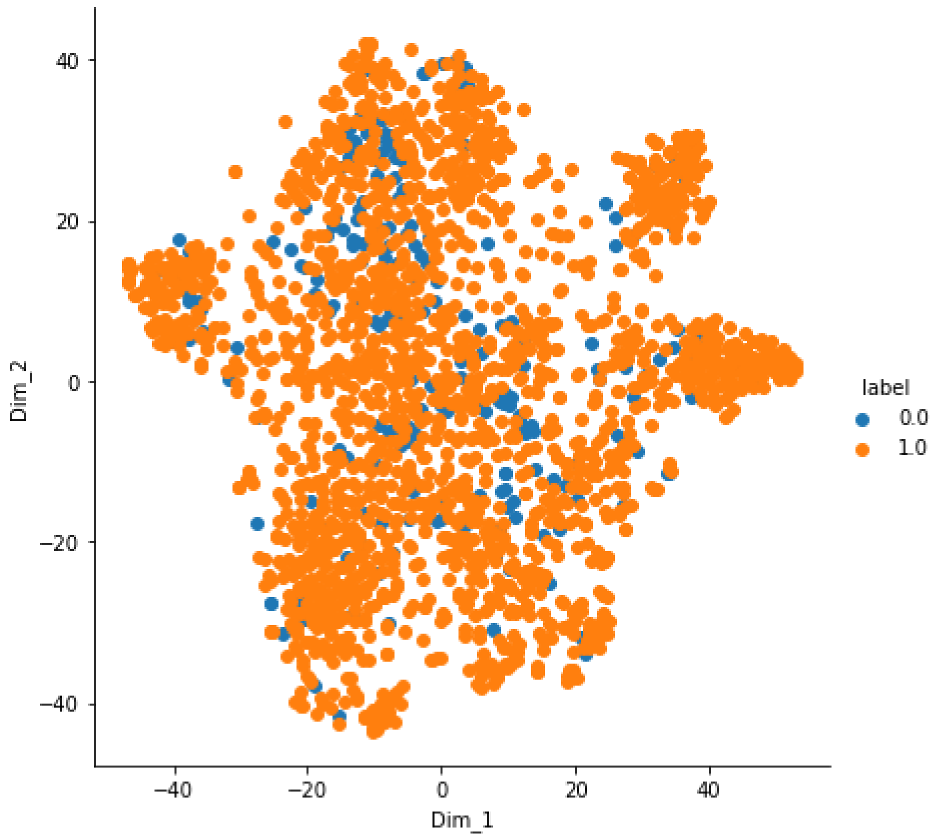
In [41]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_1000 = tfidf_sent_vectors[0:2000]
labels_1000 = final["Score"][0:2000]

TSNE_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 = TSNE_model.fit_transform(data_1000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).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.title('With perplexity = 50')
plt.show()
```



In [42]:

```
# TSNE

from sklearn.manifold import TSNE
import seaborn as sn

# Picking the top 1000 points as TSNE takes a lot of time for 364K points
data_1000 = tfidf_sent_vectors[0:2000]
labels_1000 = final["Score"][0:2000]

TSNE_model = TSNE(n_components=2, random_state=0,perplexity = 40,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 = TSNE_model.fit_transform(data_1000)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).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.title('With perplexity = 50')
plt.show()
```



Observations

tsne plot analysis

1.Bow:by observing above plots we conclude that as the perplexity and number of iterations increases positive and negative classes are overlapping and they are unreadable.

2.Tf-Idf:observing above plots we conclude that as the perplexity and number of iterations increases the overlapping of both the classes increases and also the density of classes around the plot tend to decrease and then Increased in later increase of perplexity and iterations.

3.Avg W2v:observing above plots we conclude that as the perplexity and number of iterations increases the area of covered by the classes on the plot decreased.

4.Tf-idf W2v:observing above plots we conclude that as the perplexity and number of iterations increases the overlapping of both the classes alsp increases

In [ ]:

```
```