Plotting a scatter plot for Amazon review assignment

In [215]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3
import seaborn as sns
import scipy as sp
import nltk
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

d = {'color': ['b', 'r']}
```

DATA

I am taking only the required data into the dataframe , 2000 points from positive and 2000 from negative

In [216]:
```
dispos = pd.read_sql_query(""" SELECT * FROM Reviews
Where Score > 3 LIMIT 2000 """,con)
```

In [217]:
```
disneg = pd.read_sql_query(""" SELECT * FROM Reviews
Where Score < 3 LIMIT 2000 """,con)
```

In [218]:
```
dis = dispos.append(disneg,ignore_index=True)
```

Changing the values in Score with positive and negative

In [219]:
```
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

actualScore = dis['Score']
positiveNegative = actualScore.map(partition)
dis['Score'] = positiveNegative
```

Text preprocessing phase

Here , we will remove 1)Stopwords 2)all html tags 3)punctuations 4)convert it in to lowercase

In [220]:
```
# Here , we are finding the data with html tags
import re
# find sentences containing HTML tags
import re
i=0;
for sent in dis['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
8
I don't know if it's the cactus or the tequila or just the unique combination of ingredients, bu
t the flavour of this hot sauce makes it one of a kind!  We picked up a bottle once on a trip we
were on and brought it back home with us and were totally blown away!  When we realized that we
simply couldn't find it anywhere in our city we were bummed.<br /><br />Now, because of the magi
c of the internet, we have a case of the sauce and are ecstatic because of it.<br /><br />If you
love hot sauce..I mean really love hot sauce, but don't want a sauce that tastelessly burns your
throat, grab a bottle of Tequila Picante Gourmet de Inclan.  Just realize that once you taste i
t, you will never want to use any other sauce.<br /><br />Thank you for the personal, incredible
service!
```

In [221]:
```
# functions for clearing html tags and punctuations
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 special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [222]:
```
#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 dis['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 (dis['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive re
views
                    if(dis['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative re
views 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
```

In [223]:
```
dis['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-pr
ocessing of the review
dis['CleanedText']=dis['CleanedText'].str.decode("utf-8")
```

In [224]:
```
dis.head(3) #below the processed review can be seen in the CleanedText Column


# store final table into an SQlLite table for future.
con = sqlite3.connect('final.sqlite')
c=con.cursor()
con.text_factory = str
dis.to_sql('Reviews', con,  schema=None, if_exists='replace', index=True, index_label=None, chunksiz
e=None, dtype=None)
```

Bag of words

here for each sentece we will count , how many times does the word occur

for this, we are directly using countvectorizer() function

In [225]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
final_counts = count_vect.fit_transform(dis['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])
```

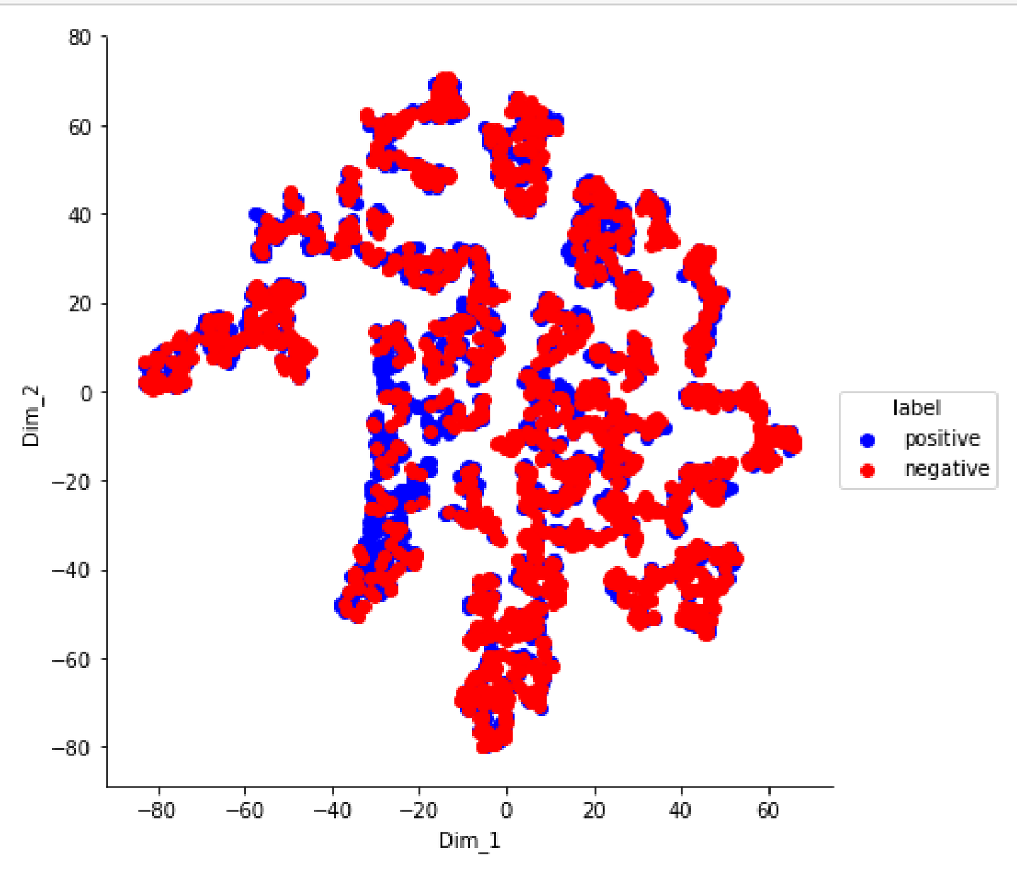
```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4000, 8013)
the number of unique words  8013
```

In [226]:
```
S = dis['Score']
Score = S.head(4000)

#final_counts = final_counts.toarray()
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
final_counts1=svd.fit_transform(final_counts)
```

In [227]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(final_counts1)
tsne_data = np.vstack((tsne_data.T, Score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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



TFIDF

Here, we will have 2 things TF-Term Frequency.. (no. of times a word occured/no. of words in a sentance) IDF-Inverse document frequency .. log(total no. of sentences/in how many sentences does the word appear) We are doing it both for bigram and unigram here..

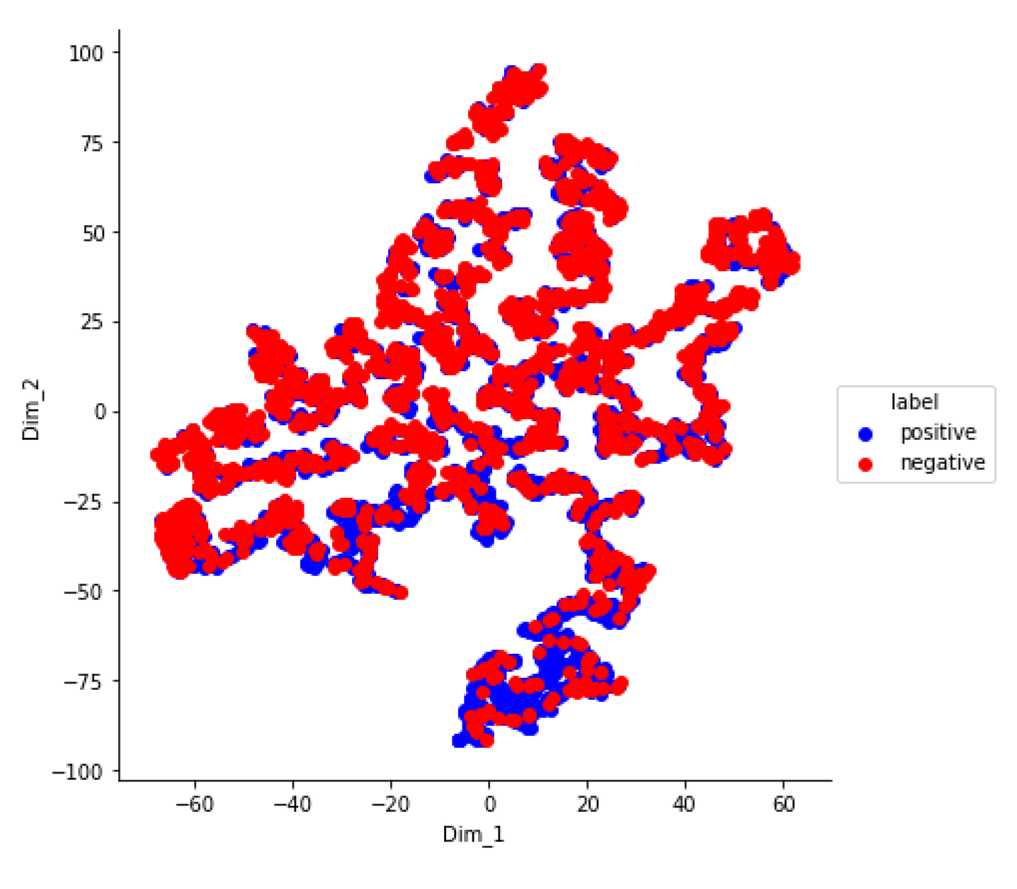
In [228]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(dis['CleanedText'].values)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1
])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4000, 106488)
the number of unique words including both unigrams and bigrams  106488
```

In [229]:
```
#dimentionality reduction
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
final_tf_idf1=svd.fit_transform(final_tf_idf)
```

In [230]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(final_tf_idf1)
tsne_data = np.vstack((tsne_data.T, Score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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



Word to Vector

It is word to vector trained by us for our data set..

In [231]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
i=0
list_of_sent=[]
for sent in dis['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [232]:
```
print(dis['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
bought sever vital can dog food product found good qualiti product look like stew process meat s
mell better labrador finicki appreci product better
*****************************************************************
['bought', 'sever', 'vital', 'can', 'dog', 'food', 'product', 'found', 'good', 'qualiti', 'produ
ct', 'look', 'like', 'stew', 'process', 'meat', 'smell', 'better', 'labrador', 'finicki', 'appre
ci', 'product', 'better']
```

In [233]:
```
# 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 [234]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  2826
```

Computing avg word to vect

computing word to vec for each word and adding and computing the avg

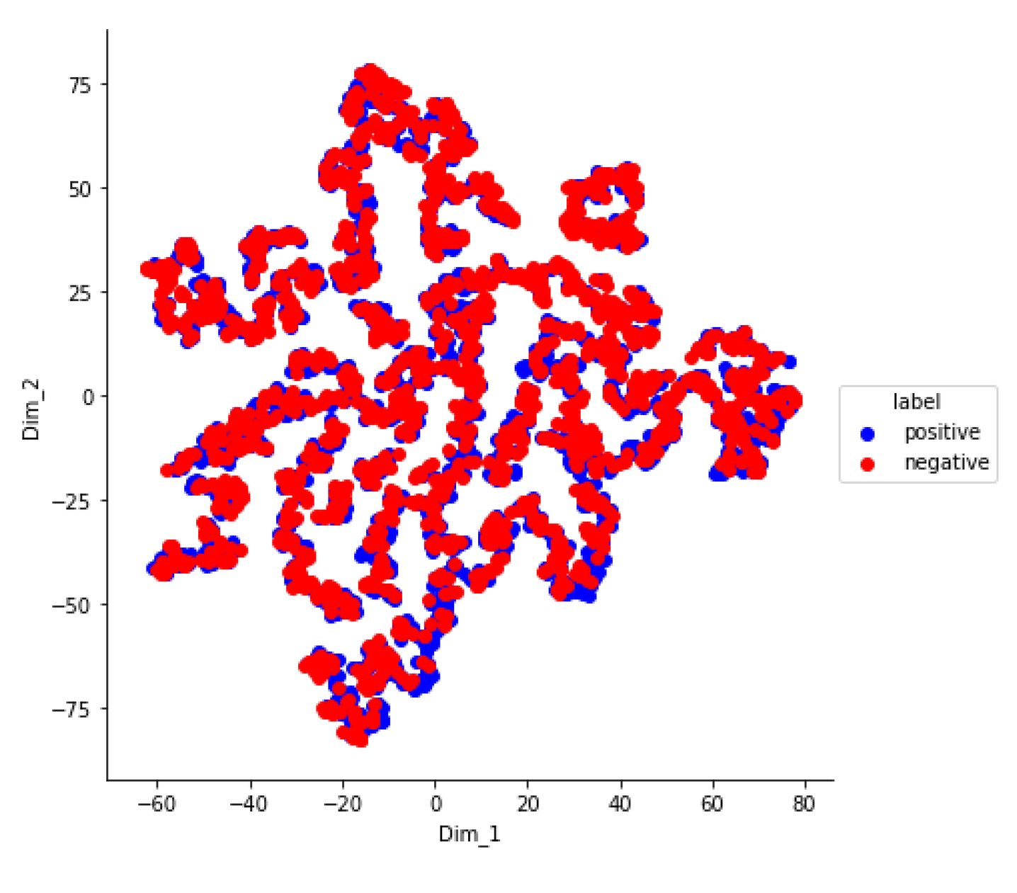
In [235]:
```
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word 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]))
```

```
4000
50
```

In [236]:
```
#dimentionality reduction
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
sent_vectors=svd.fit_transform(sent_vectors)
```

In [237]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(sent_vectors)
tsne_data = np.vstack((tsne_data.T, Score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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



TFIDF Weighted Word2vec

computing tfidf on top of w2v

In [238]:
```
# 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
        if word in w2v_words:
            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
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [239]:
```
#dim reduction
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
tf_idf_sent_vectors1=svd.fit_transform(tfidf_sent_vectors)
```

In [240]:
```
from sklearn.manifold import TSNE
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(tf_idf_sent_vectors1)
tsne_data = np.vstack((tsne_data.T, Score)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

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