Amazon Food Reviews

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

This dataset consists of reviews of fine foods from Amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Excerpt

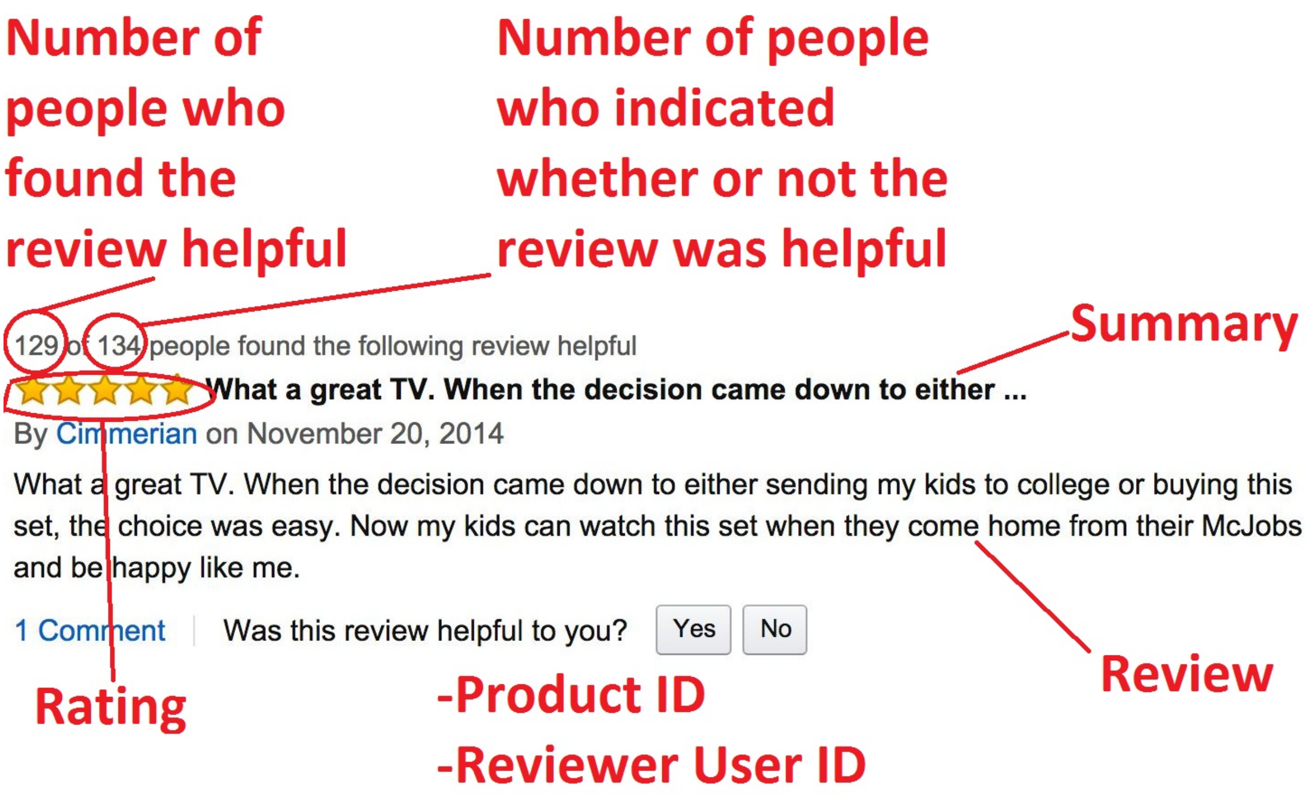
1. Defined Problem Statement
2. Performed Exploratory Data Analysis(EDA) on Amazon Fine Food Reviews Dataset plotted Word Clouds, Distplots, Histograms, etc.
3. Performed Data Cleaning & Data Preprocessing by removing unneccesary and duplicates rows and for text reviews removed html tags, punctuations, Stopwords and Stemmed the words using Porter Stemmer
4. Documented the concepts clearly
5. Plotted TSNE plots for Different Featurization of Data viz. BOW(uni-gram,bi-gram), tfidf, Avg-Word2Vec(using Word2Vec model pretrained on Google News) and tf-idf-Word2Vec

Data includes:

- Reviews from Oct 1999 - Oct 2012
- 568,454 reviews
- 256,059 users
- 74,258 products
- 260 users with > 50 reviews

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:- Review Polarity

Given a review, determine the review is positive or neagative

1.Naive Way

Naive way to do this will be the to say Score with 1 & 2 -> Negative and 4 & 5 -> positive and review with score 3 is ignored and we consider it as neutral

2. Using text review to decide the polarity

Take the summary and text of review and analyze it using NLP whether the customer feedback/review is positive or negative

In [1]:

```
#Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3 as sql
import seaborn as sns
from time import time
import gensim
import random
import warnings


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 like the
Jupyter notebook,
#directly below the code cell that produced it. The resulting plots will then also be stored in th
e notebook document.

#Pickle python objects to file
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"), protocol=4)
def openfromfile(filename):
```

```
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

First Let's do the EDA

Loading the data

In [6]:

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

con = sql.connect("database.sqlite")#Connection object that represents the database

#Using pandas functions to query from sql table
df = 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
df.head()
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	130386240(
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134697600(
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	121901760(
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	130792320(
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	135077760(

In [7]:

```
df.describe()
```

Out[7]:

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	568454.000000	568454.000000	568454.00000	568454.000000	5.684540e+05
mean	284227.500000	1.743817	2.22881	4.183199	1.296257e+09
std	164098.679298	7.636513	8.28974	1.310436	4.804331e+07
min	1.000000	0.000000	0.00000	1.000000	9.393408e+08
25%	142114.250000	0.000000	0.00000	4.000000	1.271290e+09
50%	284227.500000	0.000000	1.00000	5.000000	1.311120e+09
75%	426340.750000	2.000000	2.00000	5.000000	1.332720e+09
max	568454.000000	866.000000	923.00000	5.000000	1.351210e+09

In [8]:

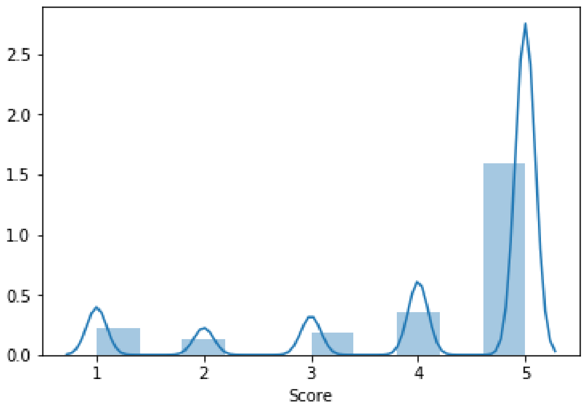
```
df.shape
df['Score'].size
```

Out[8]:

```
568454
```

In [9]:

```
sns.distplot(df['Score'],bins=10)
plt.show()
```



In [10]:

```
df['Score'].value_counts()
```

Out[10]:

```
5    363122
4     80655
1     52268
3     42640
2     29769
Name: Score, dtype: int64
```

In [11]:

```
#Using pandas functions to query from sql table
df = pd.read_sql_query("""
SELECT * FROM Reviews
WHERE Score != 3
""",con)
```

1 Naive Way

1. Naive Way

Score as positive or negative

In [12]:

```python
def polarity(x):
    if x < 3:
        return 'Negative'
    else:
        return 'Positive'
df["Score"] = df["Score"].map(polarity) #Map all the scores as the function polarity i.e. positive or negative
df.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	1303862
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	1346976
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	1219017
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	Negative	1307923
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	Positive	1350777

Using Score column now we can say either a Review is positive or negative

2.Using Text data and Natural Language Processing (NLP)

Firstly we need to perform some data cleaning and then text preprocessing and convert the texts as vectors so that we can train some model on those vectors and predict polarity of the review

1.Data Cleaning

(i) Data Deduplication

In [13]:

```
df.duplicated(subset={"UserId","ProfileName","Time","Text"}).value_counts()
```

Out[13]:

```
False    364173
True     161641
dtype: int64
```

There exist alot of duplicates wherein the different products is **reviewed by same user at the same time**
The product ID may be different but the product is similar with different variant

In [14]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display
```

Out[14]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	119957760

Geeta gave the review at the same time for multiple product which is not possible ethically, the product were same but different flavours hence counted as multiple products

In [15]:

```
#Deleting all the duplicates having the same userID, Profile, NameTime and Text all in the same co
lumn.
df1 =  df.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep="first")
```

In [16]:

```
size_diff = df1['Id'].size/df['Id'].size
```

```
size_diff = df1['Id'].size/df['Id'].size
print("%.1f %% reduction in data after deleting duplicates"%((1-size_diff)*100))
print("Size of data",df1['Id'].size," rows ")
```

```
30.7 % reduction in data after deleting duplicates
Size of data 364173  rows
```

(ii) Helpfullness Numerator Greater than Helpfullness Denominator

In [17]:

```
df2 = df1[df1.HelpfulnessNumerator <= df1.HelpfulnessDenominator]
print("Size of data",df2['Id'].size," rows ")
```

```
Size of data 364171  rows
```

Text Preprocessing

[1] HTML Tag Removal

In [18]:

```
import re #Regex (Regualar Expr Operations)
#string = r"sdfsdfd" :- r is for raw string as Regex often uses \ backslashes(\w), so they are oft
en raw strings(r'\d')

########Function to remove html tags from data
def striphtml(data):
    p = re.compile('<.*?>')#Find this kind of pattern
#     print(p.findall(data))#List of strings which follow the regex pattern
    return p.sub('',data) #Substitute nothing at the place of strings which matched the patterns

striphtml('<a href="foo.com" class="bar">I Want This <b>text!</b></a><>')
```

Out[18]:

```
'I Want This text!'
```

[2] Punctuations Removal

In [19]:

```
########Function to remove All the punctuations from the text
def strippunc(data):
    p = re.compile(r'[?|!|\'|"|#|.|,|)|(|\|/|~|%|*]')
    return p.sub('',data)
strippunc("fsd*?~,,,( sdfsdfdsvv)#")
```

Out[19]:

```
'fsd sdfsdfdsvv'
```

[3] Stopwords

Stop words usually refers to the most common words in a language are generally filtered out before or after processing of natural language data. Sometimes it is avoided to remove the stop words to support phrase search.

In [22]:

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = stopwords.words('english') #All the stopwords in English language
#excluding some useful words from stop words list as we doing sentiment analysis
excluding = ['against','not','don', "don't",'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
```

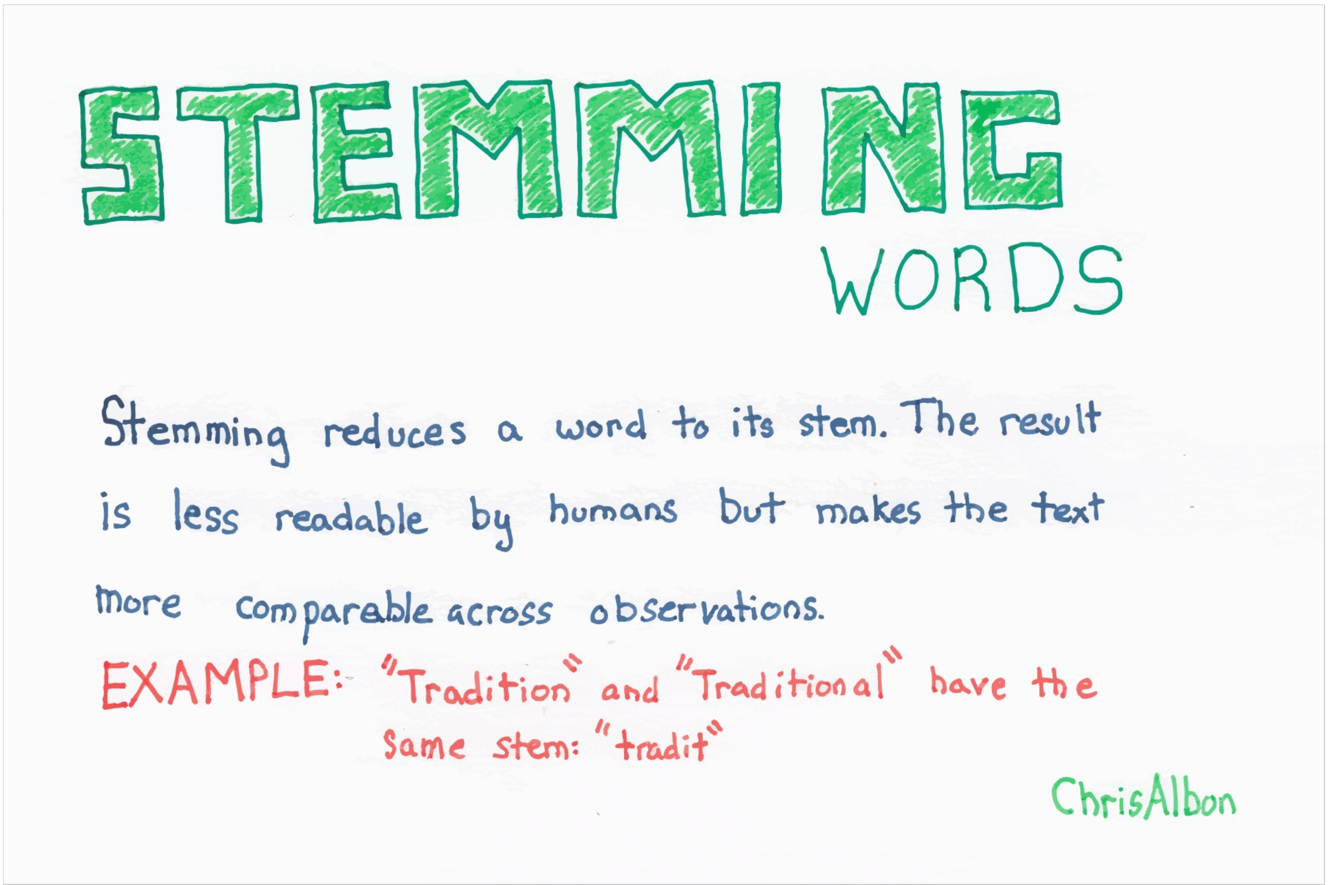
```
"didn't",
              'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't",
              'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",'shouldn', "shouldn't", '
wasn',
              "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
stop = [words for words in stop if words not in excluding]
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
"you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's",
'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', '
is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'd
id', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'between', 'into', 'through', 'during', 'before', 'after', 'ab
ove', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'fu
rther', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'eac
h', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'should', "should've", 'now', 'd', 'll', '
m', 'o', 're', 've', 'y', 'ma', 'shan', "shan't"]
```

[4] Stemming

Porter Stemmer: Most commonly used stemmer without a doubt, also one of the most gentle stemmers. Though it is also the most computationally intensive of the algorithms. It is also the oldest stemming algorithm by a large margin.

SnowBall Stemmer(Porter2): Nearly universally regarded as an improvement over porter, and for good reason. Porter himself in fact admits that it is better than his original algorithm. Slightly faster computation time than Porter, with a fairly large community around it.



In [23]:

```
from nltk.stem import SnowballStemmer
snow = SnowballStemmer('english') #initialising the snowball stemmer
print("Stem/Root words of the some of the words using SnowBall Stemmer:")
print(snow.stem('tasty'))
print(snow.stem('tasteful'))
print(snow.stem('tastiest'))
print(snow.stem('delicious'))
```

```
print(snow.stem('amazing'))
print(snow.stem('amaze'))
print(snow.stem('initialize'))
print(snow.stem('fabulous'))
print(snow.stem('Honda City'))
print(snow.stem('unpleasant'))
```

```
Stem/Root words of the some of the words using SnowBall Stemmer:
tasti
tast
tastiest
delici
amaz
amaz
initi
fabul
honda c
unpleas
```

Stemming and Lemmatization Differences

- Both lemmatization and stemming attempt to bring a canonical form for a set of related word forms.
- Lemmatization takes the part of speech in to consideration. For example, the term 'meeting' may either be returned as 'meeting' or as 'meet' depending on the part of speech.
- Lemmatization often uses a tagged vocabulary (such as Wordnet) and can perform more sophisticated normalization. E.g. transforming mice to mouse or foci to focus.
- Stemming implementations, such as the Porter's stemmer, use heuristics that truncates or transforms the end letters of the words with the goal of producing a normalized form. Since this is algorithm based, there is no requirement of a vocabulary.
- Some stemming implementations may combine a vocabulary along with the algorithm. Such an approach for example convert 'cars' to 'automobile' or even 'Honda City', 'Mercedes Benz' to a common word 'automobile'
- A stem produced by typical stemmers may not be a word that is part of a language vocabulary but lemmatizer transform the given word forms to a valid lemma.

Preprocessing output for one review

In [24]:

```
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 df2['Text'][2:3].values: #Running only for 2nd review
    filtered_sentence=[]
    print(sent) #Each review
    sent=striphtml(sent)# remove HTMl tags
    sent=strippunc(sent)# remove Punctuation Symbols
    print(sent.split())
    for w in sent.split():
        print("==============================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght less than 2
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall Stemmer
                print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
            else:
                print("Eliminated as it is a stopword")
                continue
        else:
            print("Eliminated as it is a numerical value or character of lenght less than 2")
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
```

```
    final_string.append(str1)
    print("*************************************************************************")
    print("Finally selected words from the review:\n",final_string)
```

```
This is a confection that has been around a few centuries.  It is a light, pillowy citrus gelatin
with nuts - in this case Filberts. And it is cut into tiny squares and then liberally coated with
powdered sugar.  And it is a tiny mouthful of heaven.  Not too chewy, and very flavorful.  I highl
y recommend this yummy treat.  If you are familiar with the story of C.S. Lewis' "The Lion, The Wi
tch, and The Wardrobe" - this is the treat that seduces Edmund into selling out his Brother and Si
sters to the Witch.
['This', 'is', 'a', 'confection', 'that', 'has', 'been', 'around', 'a', 'few', 'centuries', 'It',
'is', 'a', 'light', 'pillowy', 'citrus', 'gelatin', 'with', 'nuts', '-', 'in', 'this', 'case',
'Filberts', 'And', 'it', 'is', 'cut', 'into', 'tiny', 'squares', 'and', 'then', 'liberally',
'coated', 'with', 'powdered', 'sugar', 'And', 'it', 'is', 'a', 'tiny', 'mouthful', 'of', 'heaven',
'Not', 'too', 'chewy', 'and', 'very', 'flavorful', 'I', 'highly', 'recommend', 'this', 'yummy', 't
reat', 'If', 'you', 'are', 'familiar', 'with', 'the', 'story', 'of', 'CS', 'Lewis', 'The', 'Lion',
'The', 'Witch', 'and', 'The', 'Wardrobe', '-', 'this', 'is', 'the', 'treat', 'that', 'seduces', 'E
dmund', 'into', 'selling', 'out', 'his', 'Brother', 'and', 'Sisters', 'to', 'the', 'Witch']
=================================> This
Eliminated as it is a stopword
=================================> is
Eliminated as it is a numerical value or character of lenght less than 2
=================================> a
Eliminated as it is a numerical value or character of lenght less than 2
=================================> confection
Selected: Stem Word-> b'confect'
=================================> that
Eliminated as it is a stopword
=================================> has
Eliminated as it is a stopword
=================================> been
Eliminated as it is a stopword
=================================> around
Selected: Stem Word-> b'around'
=================================> a
Eliminated as it is a numerical value or character of lenght less than 2
=================================> few
Eliminated as it is a stopword
=================================> centuries
Selected: Stem Word-> b'centuri'
=================================> It
Eliminated as it is a numerical value or character of lenght less than 2
=================================> is
Eliminated as it is a numerical value or character of lenght less than 2
=================================> a
Eliminated as it is a numerical value or character of lenght less than 2
=================================> light
Selected: Stem Word-> b'light'
=================================> pillowy
Selected: Stem Word-> b'pillowi'
=================================> citrus
Selected: Stem Word-> b'citrus'
=================================> gelatin
Selected: Stem Word-> b'gelatin'
=================================> with
Eliminated as it is a stopword
=================================> nuts
Selected: Stem Word-> b'nut'
=================================> -
Eliminated as it is a numerical value or character of lenght less than 2
=================================> in
Eliminated as it is a numerical value or character of lenght less than 2
=================================> this
Eliminated as it is a stopword
=================================> case
Selected: Stem Word-> b'case'
=================================> Filberts
Selected: Stem Word-> b'filbert'
=================================> And
Eliminated as it is a stopword
=================================> it
Eliminated as it is a numerical value or character of lenght less than 2
=================================> is
Eliminated as it is a numerical value or character of lenght less than 2
=================================> cut
```

```
Selected: Stem Word-> b'cut'
=================================> into
Eliminated as it is a stopword
=================================> tiny
Selected: Stem Word-> b'tini'
=================================> squares
Selected: Stem Word-> b'squar'
=================================> and
Eliminated as it is a stopword
=================================> then
Eliminated as it is a stopword
=================================> liberally
Selected: Stem Word-> b'liber'
=================================> coated
Selected: Stem Word-> b'coat'
=================================> with
Eliminated as it is a stopword
=================================> powdered
Selected: Stem Word-> b'powder'
=================================> sugar
Selected: Stem Word-> b'sugar'
=================================> And
Eliminated as it is a stopword
=================================> it
Eliminated as it is a numerical value or character of lenght less than 2
=================================> is
Eliminated as it is a numerical value or character of lenght less than 2
=================================> a
Eliminated as it is a numerical value or character of lenght less than 2
=================================> tiny
Selected: Stem Word-> b'tini'
=================================> mouthful
Selected: Stem Word-> b'mouth'
=================================> of
Eliminated as it is a numerical value or character of lenght less than 2
=================================> heaven
Selected: Stem Word-> b'heaven'
=================================> Not
Selected: Stem Word-> b'not'
=================================> too
Eliminated as it is a stopword
=================================> chewy
Selected: Stem Word-> b'chewi'
=================================> and
Eliminated as it is a stopword
=================================> very
Eliminated as it is a stopword
=================================> flavorful
Selected: Stem Word-> b'flavor'
=================================> I
Eliminated as it is a numerical value or character of lenght less than 2
=================================> highly
Selected: Stem Word-> b'high'
=================================> recommend
Selected: Stem Word-> b'recommend'
=================================> this
Eliminated as it is a stopword
=================================> yummy
Selected: Stem Word-> b'yummi'
=================================> treat
Selected: Stem Word-> b'treat'
=================================> If
Eliminated as it is a numerical value or character of lenght less than 2
=================================> you
Eliminated as it is a stopword
=================================> are
Eliminated as it is a stopword
=================================> familiar
Selected: Stem Word-> b'familiar'
=================================> with
Eliminated as it is a stopword
=================================> the
Eliminated as it is a stopword
=================================> story
Selected: Stem Word-> b'stori'
=================================> of
Eliminated as it is a numerical value or character of lenght less than 2
```

```
================================> CS
Eliminated as it is a numerical value or character of lenght less than 2
================================> Lewis
Selected: Stem Word-> b'lewi'
================================> The
Eliminated as it is a stopword
================================> Lion
Selected: Stem Word-> b'lion'
================================> The
Eliminated as it is a stopword
================================> Witch
Selected: Stem Word-> b'witch'
================================> and
Eliminated as it is a stopword
================================> The
Eliminated as it is a stopword
================================> Wardrobe
Selected: Stem Word-> b'wardrob'
================================> -
Eliminated as it is a numerical value or character of lenght less than 2
================================> this
Eliminated as it is a stopword
================================> is
Eliminated as it is a numerical value or character of lenght less than 2
================================> the
Eliminated as it is a stopword
================================> treat
Selected: Stem Word-> b'treat'
================================> that
Eliminated as it is a stopword
================================> seduces
Selected: Stem Word-> b'seduc'
================================> Edmund
Selected: Stem Word-> b'edmund'
================================> into
Eliminated as it is a stopword
================================> selling
Selected: Stem Word-> b'sell'
================================> out
Eliminated as it is a stopword
================================> his
Eliminated as it is a stopword
================================> Brother
Selected: Stem Word-> b'brother'
================================> and
Eliminated as it is a stopword
================================> Sisters
Selected: Stem Word-> b'sister'
================================> to
Eliminated as it is a numerical value or character of lenght less than 2
================================> the
Eliminated as it is a stopword
================================> Witch
Selected: Stem Word-> b'witch'
***********************************************************************
Finally selected words from the review:
 [b'confect around centuri light pillowi citrus gelatin nut case filbert cut tini squar liber coat
powder sugar tini mouth heaven not chewi flavor high recommend yummi treat familiar stori lewi
lion witch wardrob treat seduc edmund sell brother sister witch']
```

Preprocessing on all the reviews

In [29]:

```
%%time
# Code takes a while to run as it needs to run on around 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=''
t0=time()
for sent in df2['Text'].values:
```

```
    filtered_sentence=[]
#     print(sent) #Each review
    sent=striphtml(sent)# remove HTMl tags
    sent=strippunc(sent)# remove Punctuation Symbols
#     print(sent.split())
    for w in sent.split():
#         print("==================================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght less t
han 2
            if(w.lower() not in stop):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall Stemmer
                                        #encoding as byte-string/utf-8
#                 print("Selected: Stem Word->",s)
                filtered_sentence.append(s)
                if (df2['Score'].values)[i] == 'Positive':
                    all_positive_words.append(s) #list of all words used to describe positive revie
ws
                if(df2['Score'].values)[i] == 'Negative':
                    all_negative_words.append(s) #list of all words used to describe negative revie
ws reviews
            else:
#                 print("Eliminated as it is a stopword")
                continue
        else:
#             print("Eliminated as it is a numerical value or character of lenght less than 2")
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
            #encoding as byte-string/utf-8

    final_string.append(str1)
#     print("***********************************************************************")
#     print("Finally selected words from the review:\n",final_string)
    i+=1
print("Preprocessing completed in ")
```

```
Preprocessing completed in
CPU times: user 9min 36s, sys: 420 ms, total: 9min 37s
Wall time: 9min 37s
```

Cleaned text Without Stemming for Google trained W2Vec{You will See Further}

In [30]:

```
%%time
# Code takes a while to run as it needs to run on around 500k sentences.
i=0
str1=' '
final_string_nostem=[]
s=''
t0=time()
for sent in df2['Text'].values:
    filtered_sentence=[]
    sent=striphtml(sent)# remove HTMl tags
    sent=strippunc(sent)# remove Punctuation Symbols
    for w in sent.split():
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght less t
han 2
            if(w.lower() not in stop):# If it is a stopword
                s=w.lower().encode('utf8') #encoding as byte-string/utf-8
            else:
                continue
        else:
            continue
    str1 = b" ".join(filtered_sentence)
    final_string_nostem.append(str1)
    i+=1
print("Preprocessing completed in ")
```

```
Preprocessing completed in
CPU times: user 1min 18s, sys: 16 ms, total: 1min 18s
Wall time: 1min 18s
```

The above code uses string as byte-string / utf-8(uses 1 byte), Python defaut stores string as Unicode / (utf16/utf32) {depends on how python was compiled}-(uses 2/4 byte) as our data is large 1 byte difference can save a lot of memory. Hence encoding the data as byte-string
For more info: https://stackoverflow.com/questions/10060411/byte-string-vs-unicode-string-python

Postive and Negative words in reviews

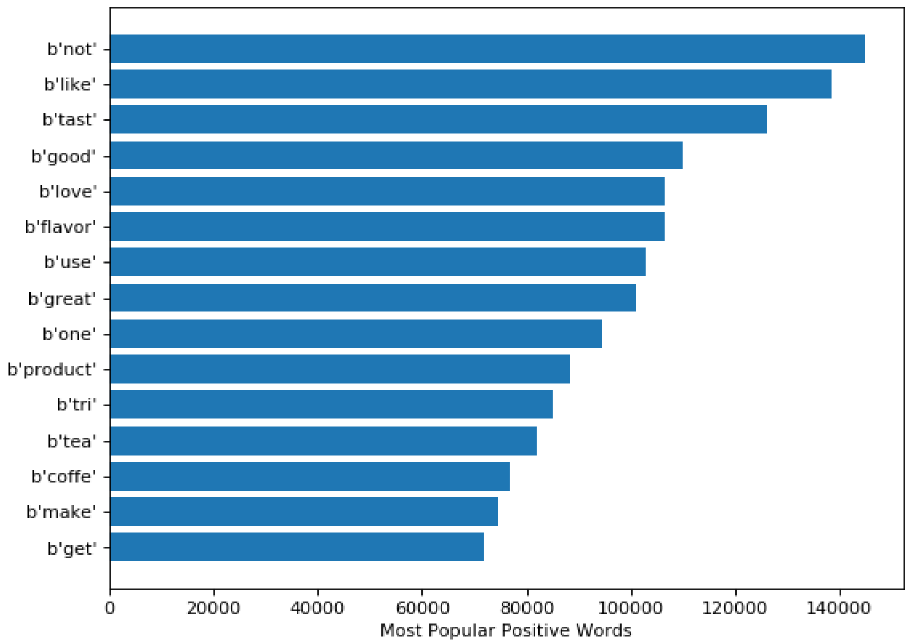
In [31]:

```
from collections import Counter
print("No. of positive words:",len(all_positive_words))
print("No. of negative words:",len(all_negative_words))
# print("Sample postive words",all_positive_words[:9])
# print("Sample negative words",all_negative_words[:9])
positive = Counter(all_positive_words)
print("\nMost Common postive words",positive.most_common(10))
negative = Counter(all_negative_words)
print("\nMost Common negative words",negative.most_common(10))
```

```
No. of positive words: 11678044
No. of negative words: 2393854

Most Common postive words [(b'not', 145019), (b'like', 138335), (b'tast', 126024), (b'good', 10983
8), (b'love', 106551), (b'flavor', 106408), (b'use', 102872), (b'great', 101125), (b'one', 94396),
(b'product', 88466)]

Most Common negative words [(b'not', 53634), (b'tast', 33828), (b'like', 32059), (b'product', 2741
1), (b'one', 20176), (b'flavor', 18898), (b'would', 17858), (b'tri', 17515), (b'use', 15148), (b'g
ood', 14616)]
```

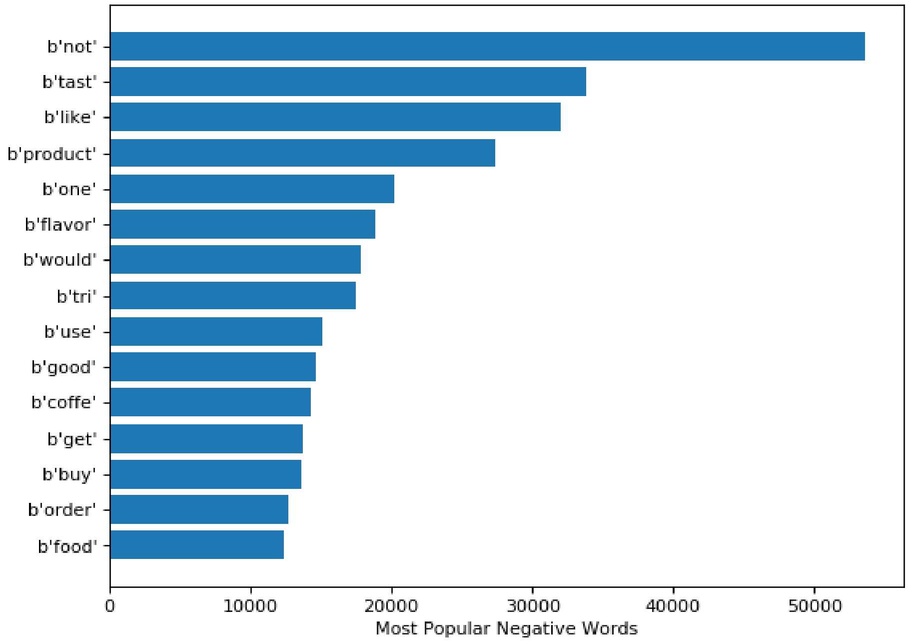
In [89]:

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
pos_words = positive.most_common(15)
pos_words.sort(key=lambda x: x[1], reverse=False)
words=[]
times=[]
for w,t in pos_words:
    words.append(w)
    times.append(t)
plt.barh(range(len(words)),times)
plt.yticks(range(len(words)),words)
plt.xlabel('Most Popular Positive Words')
plt.show()
```



In [91]:

```
neg_words = negative.most_common(15)
neg_words.sort(key=lambda x: x[1], reverse=False)
words=[]
times=[]
for w,t in neg_words:
    words.append(w)
    times.append(t)
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
plt.barh(range(len(words)),times)
plt.yticks(range(len(words)),words)
plt.xlabel('Most Popular Negative Words')
plt.show()
```



- "tast" , "like" , "flavor", "good" and "one" are some of the most common words in both negative and positve reviews
- "good" and "great" are some of the most common words in positive reviews
- "would" and "coffe" are some of the most common words in negative reviews
- tasty, good, etc are some of the words common in both **because there may be a not before it like "not tasty" , "not good"**

Storing our preprocessed data in DB

In [105]:

```
#Adding a column of CleanedText which displays the data after pre-processing of the review
df2['CleanedText']=final_string
df2['CleanedText_NoStem']=final_string_nostem
df2.head(3)
```

Out[105]:

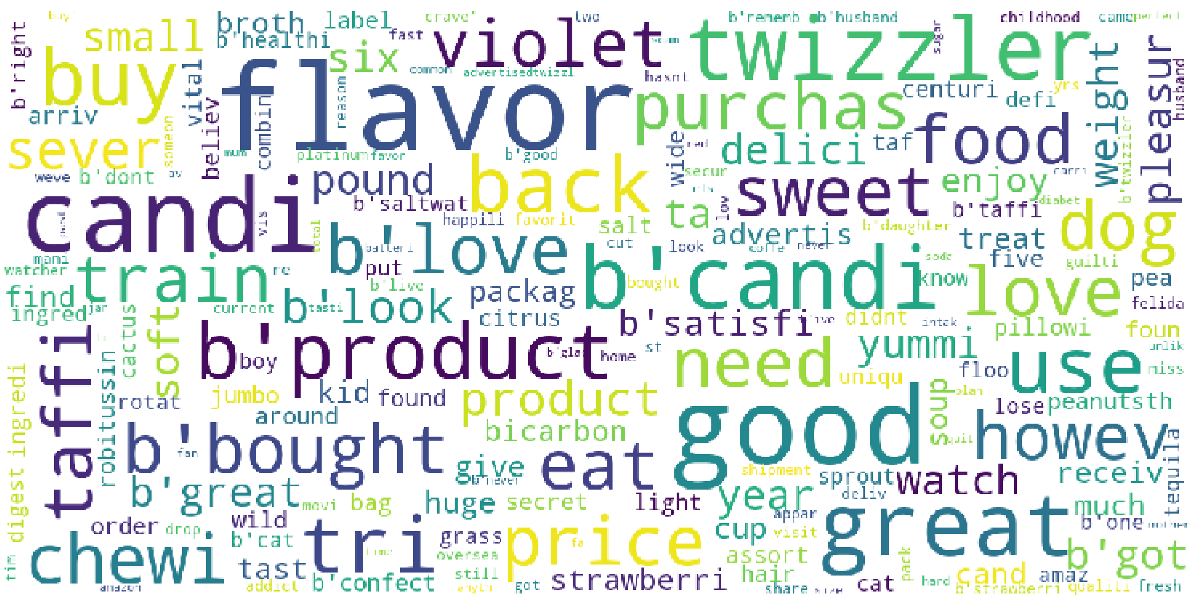
	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	Positive	1303862
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	Negative	1346976

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	Positive	1219017

Word Cloud of Whole Dataset

In [107]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

plt.rcParams['figure.figsize']=(8.0,6.0)     #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                 #10
plt.rcParams['savefig.dpi']=100              #72
plt.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2['CleanedText'])
df2.loc[df2['Score'] == 'Positive']['CleanedText']
```



Word Cloud of only Positive Reviews

In [109]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

plt.rcParams['figure.figsize']=(8.0,6.0)     #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                 #10
plt.rcParams['savefig.dpi']=100              #72
plt.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2.loc[df2['Score'] == 'Positive']['CleanedText'])
```

Word Cloud of only Negative Reviews

In [110]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

plt.rcParams['figure.figsize']=(8.0,6.0)     #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                 #10
plt.rcParams['savefig.dpi']=100              #72
plt.rcParams['figure.subplot.bottom']=.1
```

```python
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2.loc[df2['Score'] == 'Negative']['CleanedText'])
```

In [90]:

```python
### Storing dataframe in sqlite3
import sqlite3

con = sqlite3.connect('final.sqlite')
con.text_factory = str #To store the string as byte strings only
df2.to_sql('Reviews', con,if_exists='replace')
```

In [6]:

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

con = sql.connect("final.sqlite")#Loading Cleaned/ Preprocesed text that we did in Text
Preprocessing

#Using pandas functions to query from sql table
df2 = pd.read_sql_query("""
SELECT * FROM Reviews
""",con)
```

Some Key NLP Terms:

Natural Language Processing (NLP)

A Computer Science field connected to Artificial Intelligence and Computational Linguistics which focuses on interactions between computers and human language and a machine's ability to understand, or mimic the understanding of human language. Examples of NLP applications include Siri and Google Now.

Information Extraction

The process of automatically extracting structured information from unstructured and/or semi-structured sources, such as text documents or web pages for example.

Sentiment Analysis

The use of Natural Language Processing techniques to extract subjective information from a piece of text. i.e. whether an author is being subjective or objective or even positive or negative. (can also be referred to as Opinion Mining). As in this case we doing sentiment analysis of reviews of users from Amazon.

Data Corpus or Corpora

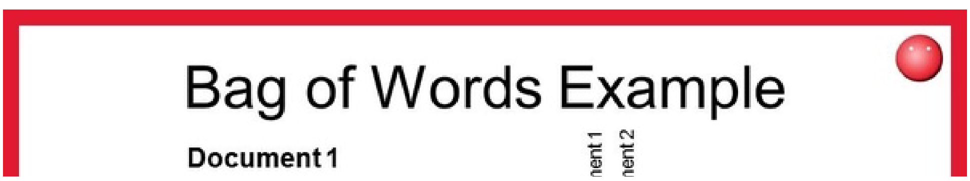
A usually large collection of documents that can be used to infer and validate linguistic rules, as well as to do statistical analysis and hypothesis testing.eg. The Amazon Fine Food Review dataset is a corpus.

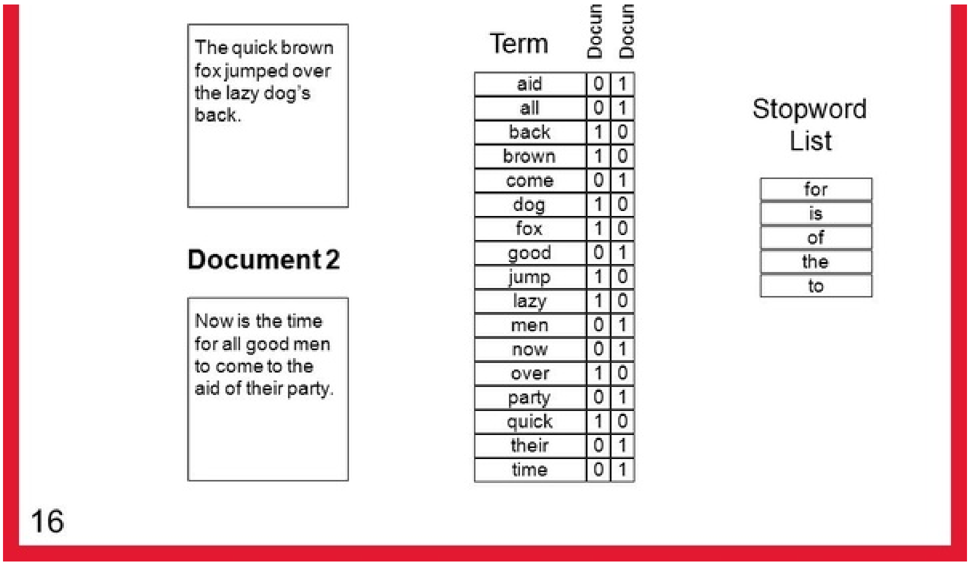
Document

A "document" is a distinct text, you could treat an individual paragraph or even sentence as a "document".
In our case our each review is a document

Bag of Words (BoW)

A commonly used model in methods of Text Classification. As part of the BOW model, a piece of text (sentence or a document) is represented as a bag or multiset of words, disregarding grammar and even word order and the frequency or occurrence of each word is used as a feature for training a classifier.
OR
Simply,Converting a collection of text documents to a matrix of token counts





Ways to convert text to vector

1. Uni-gram BOW

In [8]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [29]:

```
%%time
uni_gram = CountVectorizer() #in scikit-learn
uni_gram_vectors = uni_gram.fit_transform(df2['CleanedText'].values)
```

```
CPU times: user 17.9 s, sys: 120 ms, total: 18.1 s
Wall time: 18.1 s
```

In [30]:

```
#Saving the variable to access later without recomputing
savetofile(uni_gram_vectors,"uni_gram")
```

In [16]:

```
#Loading the variable from file
uni_gram_vectors = openfromfile("uni_gram")
```

In [31]:

```
uni_gram_vectors.shape[1]
```

Out[31]:

```
209129
```

In [11]:

```
uni_gram_vectors[0]
```

Out[11]:

```
<1x209129 sparse matrix of type '<class 'numpy.int64'>'
 with 20 stored elements in Compressed Sparse Row format>
```

In [12]:

```
type(uni_gram_vectors)
```

Out[12]:

```
scipy.sparse.csr.csr_matrix
```

In [95]:

```
%%time
from sklearn.decomposition import TruncatedSVD

tsvd_uni = TruncatedSVD(n_components=1000)#No of components as total dimensions
tsvd_uni_vec = tsvd_uni.fit_transform(uni_gram_vectors)
```

```
CPU times: user 30min 43s, sys: 29.7 s, total: 31min 13s
Wall time: 9min
```

In [96]:

```
savetofile(tsvd_uni,"tsvd_uni")
savetofile(tsvd_uni_vec,"tsvd_uni_vec")
```

In [6]:

```
tsvd_uni = openfromfile("tsvd_uni")
tsvd_uni_vec = openfromfile("tsvd_uni_vec")
```

In [27]:

```
tsvd_uni.explained_variance_ratio_[:].sum()
```

Out[27]:

```
0.82439113222951488
```

In [28]:

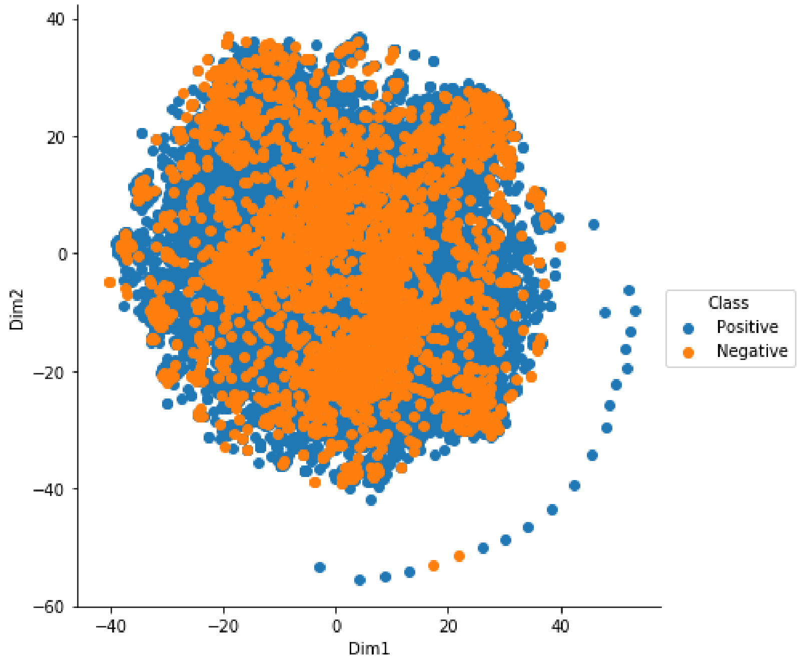
```
%%time
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 20000
sample_cols = random.sample(range(1, tsvd_uni_vec.shape[0]), n_samples)
sample_features = tsvd_uni_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
```

```
(20000, 1000) (20000, 1)
TSNE done in 1716.121s.
(20000, 3)
CPU times: user 27min 30s, sys: 1min 5s, total: 28min 36s
Wall time: 28min 36s
```

In [29]:

```
#Perplexity = 30
```

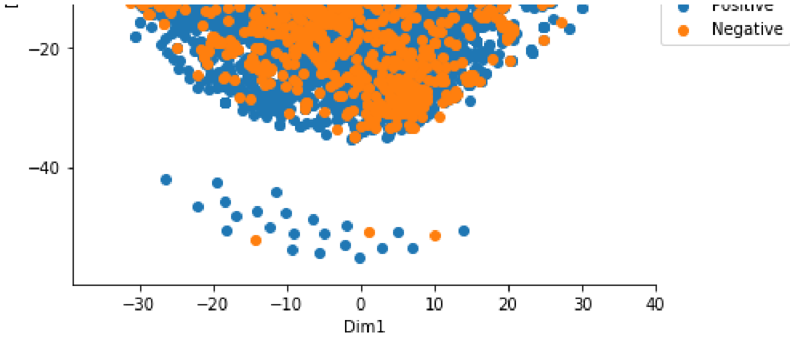
```
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```



In [32]:

```
%%time
#Perplexity = 40
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 20000
sample_cols = random.sample(range(1, tsvd_uni_vec.shape[0]), n_samples)
sample_features = tsvd_uni_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=40)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(20000, 1000) (20000, 1)
TSNE done in 1952.465s.
(20000, 3)
```

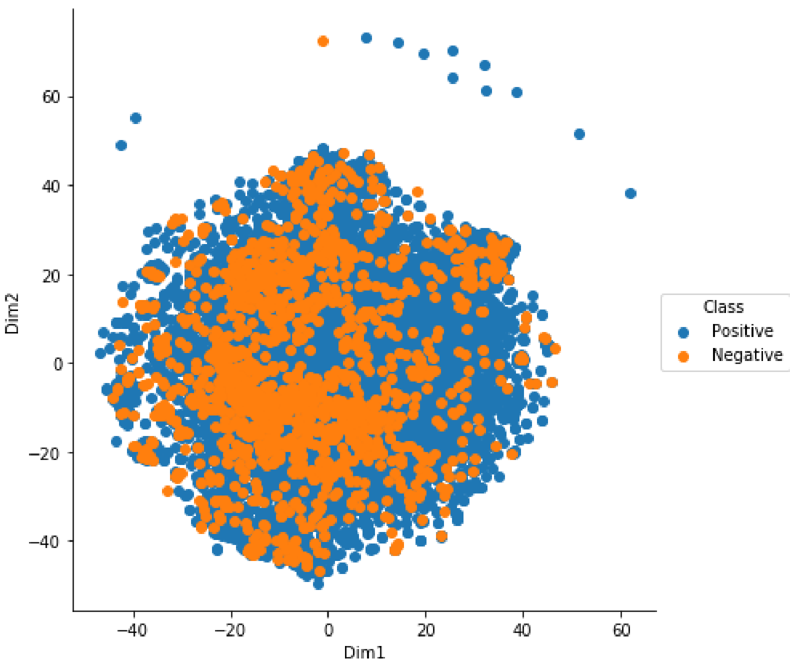



```
CPU times: user 31min 27s, sys: 1min 5s, total: 32min 33s
Wall time: 32min 33s
```

In [7]:

```
%%time
#Perplexity = 30 wiht 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_uni_vec.shape[0]), n_samples)
sample_features = tsvd_uni_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 1000) (10000, 1)
TSNE done in 608.475s.
(10000, 3)
```



```
CPU times: user 9min 43s, sys: 25.4 s, total: 10min 9s
Wall time: 10min 9s
```

2. Bi-gram BOW

In [9]:

```
%%time
#taking one words and two consecutive words together
bi_gram = CountVectorizer(ngram_range=(1,2))
bi_gram_vectors = bi_gram.fit_transform(df2['CleanedText'].values)
```

```
CPU times: user 58.2 s, sys: 728 ms, total: 59 s
Wall time: 59 s
```

In [10]:

```
#Saving the variable to access later without recomputing
savetofile(bi_gram_vectors,"bi_gram")
```

In [11]:

```
#Loading the variable from file
bi_gram_vectors = openfromfile("bi_gram")
```

In [12]:

```
bi_gram_vectors.shape
```

Out[12]:

```
(364171, 3404647)
```

In [13]:

```
bi_gram_vectors[0]
```

Out[13]:

```
<1x3404647 sparse matrix of type '<class 'numpy.int64'>'
 with 42 stored elements in Compressed Sparse Row format>
```

In [14]:

```
type(bi_gram_vectors)
```

Out[14]:

```
scipy.sparse.csr.csr_matrix
```

In [17]:

```
print("bi-gram is %.2f times more than uni-gram"%((bi_gram_vectors.shape[1]/uni_gram_vectors.shape
[1])))#Dividing boths columns
```

```
bi-gram is 16.28 times more than uni-gram
```

In [18]:

```
%%time
from sklearn.decomposition import TruncatedSVD
sample_points = df2.sample(20000)

bi_gram = CountVectorizer(ngram_range=(1,2))
bi_gram_vectors = bi_gram.fit_transform(sample_points['CleanedText'])
```

```
bi_gram_vectors = bi_gram.fit_transform(sample_points['CleanedText'])
tsvd_bi = TruncatedSVD(n_components=2500)#No of components as total dimensions
tsvd_bi_vec = tsvd_bi.fit_transform(bi_gram_vectors)
```

```
CPU times: user 1h 2min 47s, sys: 1min 4s, total: 1h 3min 51s
Wall time: 16min 54s
```

In [21]:

```
savetofile(tsvd_bi,"tsvd_bi")
savetofile(tsvd_bi_vec,"tsvd_bi_vec")
```

In [24]:

```
tsvd_bi = openfromfile("tsvd_bi")
tsvd_bi_vec = openfromfile("tsvd_bi_vec")
```

In [25]:

```
tsvd_bi.explained_variance_ratio_[:].sum()
```

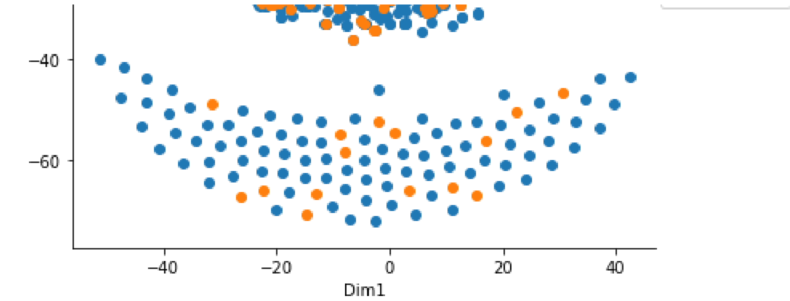
Out[25]:

```
0.72117200409365134
```

In [36]:

```
%%time
#Perplexity = 30 with 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_bi_vec.shape[0]), n_samples)
sample_features = tsvd_bi_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2500) (10000, 1)
TSNE done in 924.456s.
(10000, 3)
```

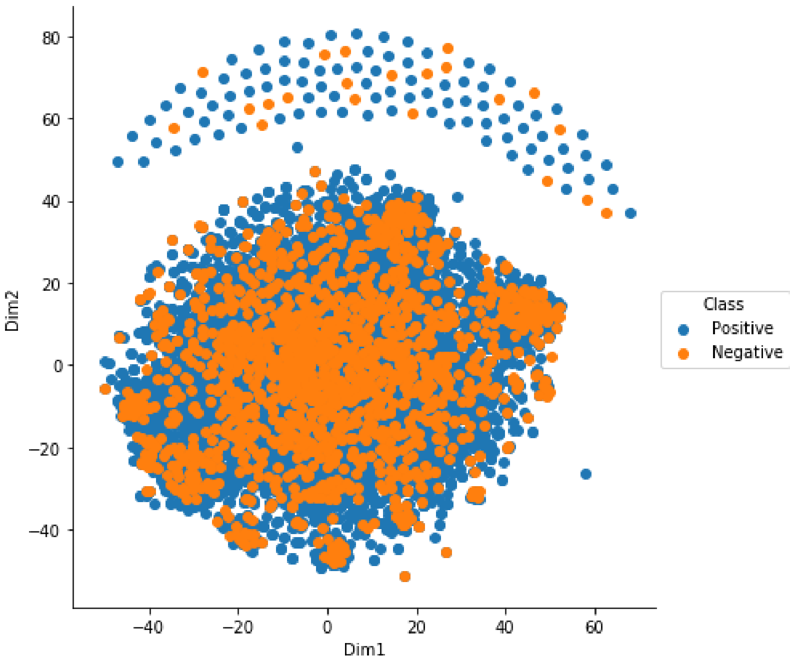



```
CPU times: user 14min 53s, sys: 31.7 s, total: 15min 25s
Wall time: 15min 25s
```

In [39]:

```
%%time
#Perplexity = 20 with 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_bi_vec.shape[0]), n_samples)
sample_features = tsvd_bi_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2500) (10000, 1)
TSNE done in 870.144s.
(10000, 3)
```

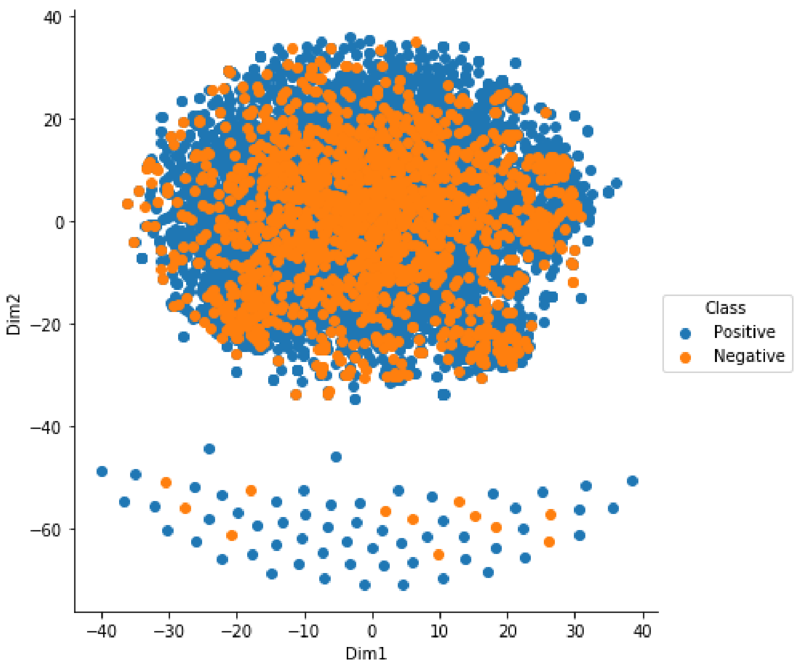


```
CPU times: user 14min, sys: 30.8 s, total: 14min 30s
Wall time: 14min 30s
```

In [40]:

```
%%time
#Perplexity = 40 with 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_bi_vec.shape[0]), n_samples)
sample_features = tsvd_bi_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=40)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2500) (10000, 1)
TSNE done in 1505.767s.
(10000, 3)
```




```
CPU times: user 24min 31s, sys: 34.6 s, total: 25min 6s
Wall time: 25min 6s
```

3. tf-idf

TFIDF = TF x IDF

Term Frequency: This summarizes how often a given word appears within a document.
Inverse Document Frequency: This downscales words that appear a lot across documents in the corpus.

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. Tf-idf is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use tf-idf.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

In [10]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec = tfidf.fit_transform(df2['CleanedText'])
```

```
CPU times: user 1min, sys: 788 ms, total: 1min
Wall time: 1min
```

In [11]:

```
#Saving the variable to access later without recomputing
savetofile(tfidf_vec,"tfidf")
```

In [12]:

```
#Loading the variable from file
tfidf_vec = openfromfile("tfidf")
```

In [13]:

```
tfidf_vec.shape
```

Out[13]:

```
(364171, 3404647)
```

tf-idf came up with 2.9 million features for the data corpus

In [14]:

```
print(tfidf_vec[2])
```

```
  (0, 1995684)	0.0230577620001
  (0, 634637)	0.0962976136656
  (0, 148485)	0.0491621357676
  (0, 483299)	0.095130835222
  (0, 1680739)	0.0511254495732
  (0, 2212861)	0.131631500755
  (0, 556420)	0.0781934548627
  (0, 1232676)	0.0865863526019
  (0, 2018759)	0.0553755411213
  (0, 465022)	0.0506518591503
  (0, 1093493)	0.119601669622
  (0, 737841)	0.0569454130211
  (0, 3064216)	0.128558300715
  (0, 2798119)	0.0741999555212
  (0, 1673827)	0.0960763467562
  (0, 575573)	0.0613536698434
```

```
  (0, 2271789)	0.0542982037779
  (0, 2885378)	0.0416697262506
  (0, 1920053)	0.0562561162586
  (0, 1397444)	0.070424363201
  (0, 516953)	0.0606048623527
  (0, 1125576)	0.027927961642
  (0, 1414824)	0.040096342878
  (0, 2423531)	0.0377442598615
  (0, 3396938)	0.0571169892128
  :	:
  (0, 576437)	0.132967109652
  (0, 2273524)	0.0936229482509
  (0, 2888619)	0.126621307635
  (0, 3064745)	0.124419662051
  (0, 1920733)	0.129355981494
  (0, 1397906)	0.117978967687
  (0, 1997120)	0.09759836373
  (0, 517278)	0.108210127217
  (0, 1128708)	0.0911014244865
  (0, 1416273)	0.0498816191539
  (0, 2427136)	0.125107119153
  (0, 3398129)	0.102164881838
  (0, 3112449)	0.144089290953
  (0, 1052035)	0.144089290953
  (0, 2851106)	0.148687090651
  (0, 1673525)	0.148687090651
  (0, 1703206)	0.132967109652
  (0, 3330091)	0.134481298135
  (0, 3256500)	0.148687090651
  (0, 3114021)	0.148687090651
  (0, 2595216)	0.148687090651
  (0, 933619)	0.148687090651
  (0, 2607918)	0.148687090651
  (0, 379436)	0.113896691731
  (0, 2684850)	0.148687090651
```

Returns all the features which is non-zero for a particular review from the sparse matrix

In [15]:

```
features = tfidf.get_feature_names()
features[190000:190010]
```

Out[15]:

```
['babi health',
 'babi healthi',
 'babi healthier',
 'babi healthiest',
 'babi healthyp',
 'babi healtyplus',
 'babi heart',
 'babi heat',
 'babi heimlich',
 'babi help']
```

Some of the feature of the tf-idf

In [16]:

```
def top_tfidf_features(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature names.'''
    topn_ind = np.argsort(row)[::-1][:top_n]
    #Sorting and getting the indexes using argsort and reversing to get descending wise and taking
the top n values
    top_feats = [(features[i], row[i]) for i in topn_ind]
    df = pd.DataFrame(top_feats,columns = ['feature', 'tfidf'])
    return df
top_tfidfs = top_tfidf_features(tfidf_vec[3000,:].toarray()[0],features,20)#top 20 tfidf features
of 3000th review
top_tfidfs
```

Out[16]:

	feature	tfidf
0	stretch strong	0.399925
1	cup stretch	0.378784
2	delici mocha	0.357643
3	delici hot	0.268651
4	chocol best	0.262176
5	best cup	0.256560
6	coffe delici	0.251700
7	stretch	0.229628
8	delici	0.220825
9	strong coffe	0.208862
10	mocha	0.199555
11	hot chocol	0.192724
12	strong	0.128024
13	hot	0.122785
14	chocol	0.115835
15	cup	0.114143
16	coffe	0.097874
17	best	0.094951
18	flavorless dark	0.000000
19	flavorless cup	0.000000

Top 20 tfidf features of 3000th review in the data corpus

In [ ]:

```
%%time
from sklearn.decomposition import TruncatedSVD

tsvd_tfidf = TruncatedSVD(n_components=100)#No of components as total dimensions
tsvd_tfidf_vec = tsvd_tfidf.fit_transform(tfidf_vec)
```

In [ ]:

```
savetofile(tsvd_tfidf,"tsvd_tfidf")
savetofile(tsvd_tfidf_vec,"tsvd_tfidf_vec")
```

In [27]:

```
tsvd_tfidf_vec = openfromfile("tsvd_tfidf_vec")
tsvd_tfidf = openfromfile("tsvd_tfidf")
```

In [41]:

```
tsvd_tfidf.explained_variance_ratio_[:].sum()
```

Out[41]:

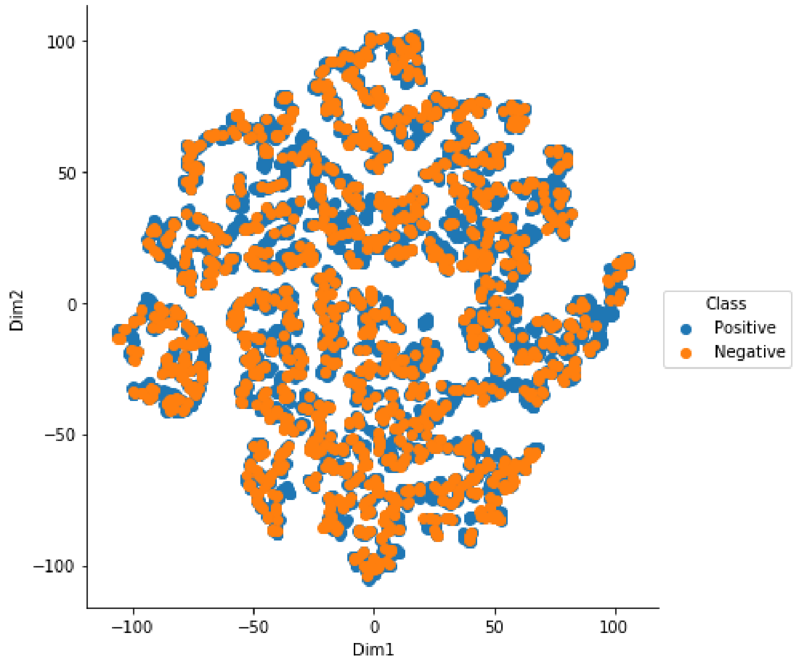
```
0.0030303842146799662
```

In [43]:

```
%%time
#Perplexity = 20 with 10k points
from sklearn.manifold import TSNE
```

```
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_tfidf_vec.shape[0]), n_samples)
sample_features = tsvd_tfidf_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2) (10000, 1)
TSNE done in 248.691s.
(10000, 3)
```



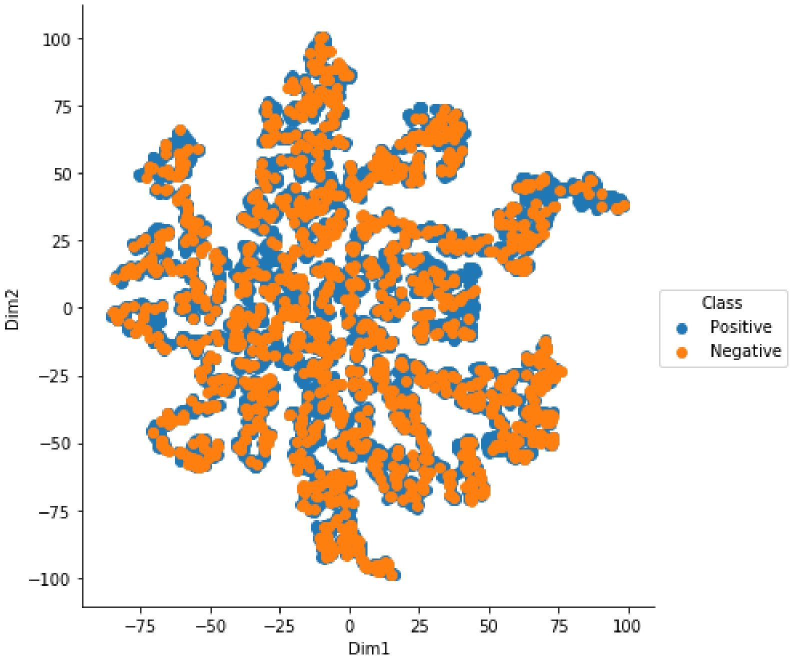

```
CPU times: user 3min 37s, sys: 32 s, total: 4min 9s
Wall time: 4min 9s
```

In [44]:

```
%%time
#Perplexity = 30 with 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_tfidf_vec.shape[0]), n_samples)
sample_features = tsvd_tfidf_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)
# print(sample_features,sample_class)
```

```
t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2) (10000, 1)
TSNE done in 283.880s.
(10000, 3)
```

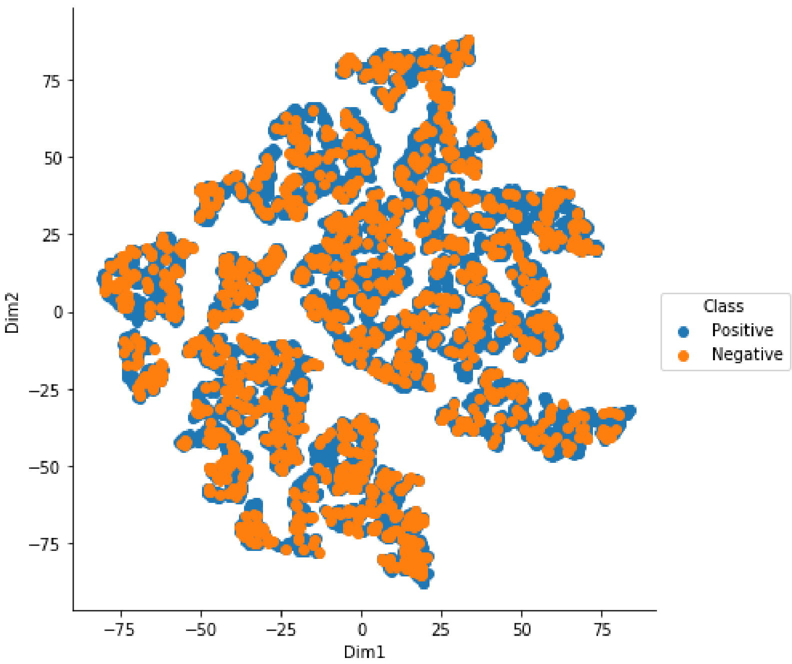



```
CPU times: user 4min 13s, sys: 31.2 s, total: 4min 44s
Wall time: 4min 44s
```

In [45]:

```
%%time
#Perplexity = 40 with 10k points
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 10000
sample_cols = random.sample(range(1, tsvd_tfidf_vec.shape[0]), n_samples)
sample_features = tsvd_tfidf_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=40)
# print(sample_features,sample_class)

t0 = time()
embedded_data = model.fit_transform(sample_features)
print("TSNE done in %0.3fs." % (time() - t0))
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])

sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 2) (10000, 1)
TSNE done in 312.785s.
(10000, 3)
```



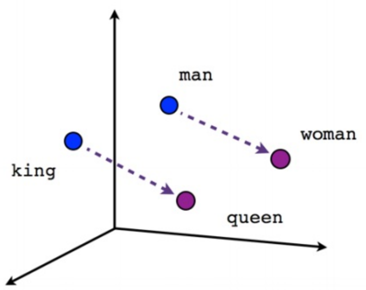

```
CPU times: user 4min 41s, sys: 32.2 s, total: 5min 13s
Wall time: 5min 13s
```

Gensim

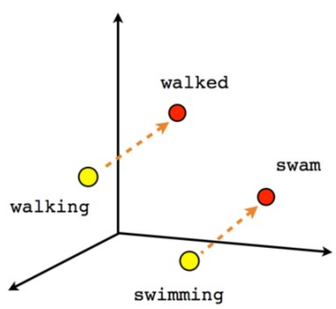
Gensim is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. It uses NumPy, SciPy and optionally Cython for performance. Gensim is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms, which differentiates it from most other scientific software packages that only target batch and in-memory processing.

4. Word2Vec

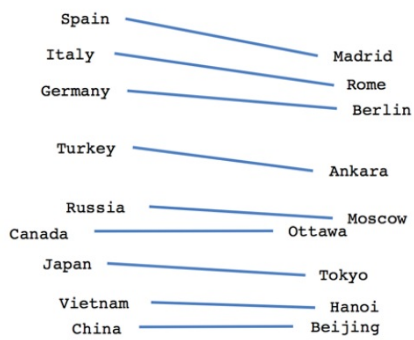
[Refer Docs] :https://radimrehurek.com/gensim/models/word2vec.html



Male-Female



Verb tense



Country-Capital

In [92]:

```
final_string = []
for sent in df2['CleanedText'].values:
    sent = str(sent)
    sentence=[]
#     print(sent)
    for word in sent.split():
```

```
#         print(word)
        sentence.append(word)
#         print(sentence)
    final_string.append(sentence)
```

In [93]:

```
%%time
# Train your own Word2Vec model using your own text corpus
import gensim

w2v_model=gensim.models.Word2Vec(final_string,min_count=5,size=50, workers=-1)
#min-count: Ignoring the words which occurs less than 5 times
#size:Creating vectors of size 50 for each word
#workers: Use these many worker threads to train the model (faster training with multicore machine
s)
```

```
CPU times: user 5.28 s, sys: 0 ns, total: 5.28 s
Wall time: 5.28 s
```

In [30]:

```
w2v_model.save('w2vmodel')#Persist/Saving the model to a file in the disk
```

In [31]:

```
w2v_model = gensim.models.Word2Vec.load('w2vmodel') #Loading the model from file in the disk
```

In [32]:

```
w2v_vocub = w2v_model.wv.vocab
len(w2v_vocub)
```

Out[32]:

```
34906
```

In [33]:

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

Out[33]:

```
[('compar', 0.5214215517044067),
 ('anywaya', 0.5147097706794739),
 ('dryer', 0.5107567310333252),
 ('hadiv', 0.5099674463272095),
 ('vivani', 0.49352583289146423),
 ('vancouv', 0.4796876013278961),
 ('vomit', 0.47908806800842285),
 ('mirin', 0.47721731662750244),
 ("peet'", 0.47637227177619934),
 ('downthi', 0.47575598895515442)]
```

In [34]:

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

Out[34]:

```
[('porch', 0.5215961933135986),
 ('maisi', 0.5133664608001709),
 ('unstick', 0.49281394481658936),
 ('complianc', 0.48115843534469604),
 ("b'pricey", 0.47690099477767944),
 ('mightili', 0.47623634338378906),
 ("b'also", 0.47145384550094604),
 ('vow', 0.4705711007118225),
 ('hexan', 0.46026793122291565),
 ('ment', 0.45899584889411926)]
```

In [35]:

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

Out[35]:

```
[('cain', 0.5610529184341431),
 ('finea', 0.5418595671653748),
 ('therapi', 0.5118728280067444),
 ('sasha', 0.5085721015930176),
 ('anton', 0.5078103542327881),
 ("soil'", 0.5062910914421082),
 ('discern', 0.5059114694595337),
 ("awsom'", 0.5021228790283203),
 ("b'newborn", 0.49307799339294434),
 ("larg'", 0.4898505210876465)]
```

4.a Avg Word2Vec

- One of the most naive but good ways to convert a sentence into a vector
- Convert all the words to vectors and then just take the avg of the vectors the resulting vector represent the sentence

In [62]:

```
%%time
avg_vec = [] #List to store all the avg w2vec's
for sent in final_string[0:1]:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(50) #Initializing with zeroes
    print("sent:",sent)
    for word in sent:
        try:
            wvec = w2v_model.wv[word] #Vector of each using w2v model
            print("wvec:",wvec)
            sent_vec += wvec #Adding the vectors
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
    print("sent_vec:",sent_vec)
    a_vec =sent_vec / cnt #Taking average of vectors sum of the particular review
    print("avg_vec:",a_vec)
    avg_vec.append(a_vec) #Storing the avg w2vec's for each review
    print("**********************************************************************")
```

```
sent: ["b'bought", 'sever', 'vital', 'can', 'dog', 'food', 'product', 'found', 'good', 'qualiti',
'product', 'look', 'like', 'stew', 'process', 'meat', 'smell', 'better', 'labrador', 'finicki', 'a
ppreci', 'product', "better'"]
wvec: [ -4.75264713e-03  -2.68976251e-03   3.45981750e-03   8.61182716e-03
  -3.45326052e-03  -6.07945665e-04  -3.81294428e-03   7.70723587e-03
  -6.99552661e-03  -2.04596203e-03  -4.63803299e-03  -7.98908077e-05
  -5.44417766e-04  -6.50190702e-03   4.06994065e-03   3.06597492e-03
  -7.19741359e-03  -8.84887390e-03  -8.55807401e-03   5.74907893e-03
   6.90606283e-03   3.70534114e-03   4.26333630e-03  -9.67295747e-03
  -5.33873122e-03   2.21293815e-03   5.71956998e-03   4.67022089e-03
  -5.25551289e-03   7.90084433e-03  -8.64620041e-03   1.24186510e-03
   6.92145852e-03   7.15341698e-03   1.75182312e-03   2.55550840e-03
   6.01556059e-03   1.02293317e-03  -7.80893781e-04   7.74320262e-03
  -5.13905776e-04  -1.03073404e-03   3.13923252e-03   1.53249697e-04
   2.41127494e-03   1.59304694e-03  -1.53104786e-03  -1.76926993e-03
   1.10834360e-03  -3.19925486e-03]
wvec: [-0.00556279  0.00752297  0.00460804 -0.00926204  0.0089932  -0.00559665
  0.00586318  0.00810189 -0.00954244  0.00968478 -0.00038469  0.00798686
 -0.00578657 -0.00694925  0.00127955 -0.00781901  0.0009803  -0.00737002
  0.0079879   0.00804264 -0.00775681 -0.00258346  0.00218481 -0.0004461
  0.001299   -0.00633101 -0.00729357 -0.00945852 -0.00238822  0.00085551
 -0.00566007  0.00693674 -0.00608156 -0.00298047  0.00115315  0.00528906
  0.0086888   0.00480924 -0.00241633 -0.00990093  0.00664739  0.00239386
 -0.00260268  0.00731526  0.00220745 -0.00478192  0.0060298  -0.00606644
 -0.00726027  0.00715353]
wvec: [ -2.65350752e-03   8.72655073e-04  -1.96381388e-05   9.93598066e-03
   9.54756513e-03   7.44436914e-03  -6.42555533e-03  -6.34707650e-03
```

```
  -4.50171949e-03   4.37499769e-03   5.31097967e-03  -4.01081610e-03
  -2.46817060e-03  -9.80363041e-03  -7.01672351e-03   6.63593784e-03
   6.25140127e-03   2.56444886e-03   6.31519733e-03   9.01319738e-03
   6.31709117e-03  -3.79921519e-03  -3.58444848e-03   8.41936748e-03
   2.53804983e-03   4.67862701e-03  -9.71257780e-03  -2.95251654e-03
  -3.97092057e-03   4.12182324e-03  -5.48095861e-03   5.65639790e-03
   1.67198631e-03  -6.95487391e-03   3.38459993e-03  -6.14847289e-03
  -2.78812542e-04   5.04260790e-03  -5.34548657e-03  -1.56506547e-03
  -8.12477153e-03   3.02348426e-03  -9.22218524e-03  -6.05887827e-03
   2.63877749e-03  -1.79424955e-04   4.02882975e-03   7.08158128e-03
   1.97588373e-03  -8.27535708e-03]
wvec: [-0.00072441 -0.00164973  0.00859921 -0.00267348 -0.00662391 -0.00917224
  0.00280171  0.00495424 -0.00053926  0.00385878 -0.00493861  0.00475864
  0.00932392  0.00718789 -0.00712787 -0.00271706 -0.00859092  0.00461992
  0.00067061  0.00324616  0.00925345 -0.00330868  0.00301088  0.0058722
  0.00149901 -0.00317736  0.00207728  0.00522385  0.0098709   0.00566704
  0.00512577 -0.00888478  0.00239988  0.00740758 -0.00856608  0.00364283
 -0.00766531 -0.00271859 -0.00355986  0.00204862  0.00109249  0.0013948
 -0.00472202 -0.00464104 -0.00835998 -0.00263224  0.00479626  0.00930953
 -0.00382798 -0.00013601]
wvec: [  3.84374172e-03  -4.76932805e-03  -3.14358692e-03   3.18449456e-03
   9.03673936e-03   4.16682893e-03   8.95305444e-03  -4.20769211e-03
   1.92374410e-03   9.60189570e-03   2.62396573e-03   2.51849112e-03
  -6.46916963e-03  -4.13086353e-04  -9.44136083e-03   8.02234933e-03
  -3.88453924e-03  -6.68661436e-03   7.86746896e-05  -5.94623666e-03
   8.33817758e-03   4.19990486e-03  -6.77729258e-03   8.21753033e-03
   1.19540619e-03   2.03565392e-03   7.88750127e-03  -1.84832039e-04
  -2.64141755e-03  -7.35277589e-03   1.39770412e-03   7.44905602e-03
  -2.63712485e-03  -5.96492458e-03   6.41079212e-04  -9.04554781e-03
  -3.10306263e-04   3.34001007e-03  -7.31774094e-03  -7.79963145e-03
  -5.66320727e-03   5.12925349e-03  -3.69603105e-04  -6.66710222e-03
  -6.04024436e-03   6.54786732e-03  -5.64918667e-03  -8.66167806e-03
   6.66423375e-03   7.56110903e-03]
wvec: [ -8.98641255e-03   5.30437101e-03  -7.78613705e-03   4.52958886e-03
  -2.02551577e-03   5.54468203e-03  -9.39768832e-03  -5.21024165e-04
  -5.65697066e-03   8.03776830e-03  -9.93325375e-03   8.65310151e-03
  -6.03862712e-03  -6.44331565e-04  -6.77965395e-03  -7.76847266e-03
  -9.80872568e-03   3.33832926e-03  -3.34202382e-03  -8.43661651e-03
  -4.27562371e-03   4.54779994e-03  -1.57508429e-03  -8.44068732e-03
   4.46273433e-03  -9.27662849e-03  -6.73664408e-03   7.73029868e-03
  -4.82420233e-04   8.36976105e-05   7.57969858e-04   6.45056320e-03
  -1.64581172e-03   6.25526765e-03  -6.10177219e-03  -2.01769616e-03
   1.83258753e-03  -9.34200175e-03  -7.38788210e-03   7.72404857e-03
   3.58597375e-03  -4.69172606e-03   6.09861035e-03   4.75727255e-04
   2.78948154e-03  -4.95554507e-03   9.31141339e-03  -7.43280677e-03
  -5.65345865e-03   8.38269014e-03]
wvec: [-0.00519954  0.00820932  0.00080061  0.00137945  0.00622367 -0.00934624
 -0.00724957 -0.00932915  0.0023706   0.00865911  0.00970238  0.00327066
  0.0011805  -0.00481626 -0.00857859  0.00761682  0.00322645  0.00828408
 -0.00867775  0.0001234   0.00376284 -0.0058005  -0.00149229  0.0014703
  0.00156599  0.00642583 -0.00796384 -0.00880892  0.00590896  0.00697395
  0.00311856  0.00201523 -0.00430622 -0.00678094 -0.00686245 -0.00372184
  0.00162162 -0.00036233  0.00421439  0.00234655  0.00766121  0.00152749
 -0.00230008  0.00227202 -0.00612525  0.00326373 -0.00284388 -0.00362739
 -0.00754044 -0.00163635]
wvec: [  3.55861476e-03   3.87572311e-03  -4.92203329e-03  -1.89245155e-03
  -6.46027410e-03  -9.06284712e-03  -5.24346530e-03   9.07435175e-03
  -8.26285779e-03   3.72582697e-03  -8.08339193e-03   2.47897953e-03
  -3.48098017e-03  -2.50451267e-03   2.94162255e-05   5.41871239e-04
   1.68376754e-03  -8.25294666e-03  -7.29853287e-03   9.91709251e-03
  -9.64085478e-03   7.34770298e-03  -8.31694528e-03   5.22464886e-03
  -3.66182392e-03   6.73329632e-04  -3.08873248e-03  -5.19602187e-03
   9.51020233e-03  -7.67551363e-03  -7.41763925e-03   9.24272370e-03
  -2.15016468e-03   5.73959854e-03  -9.58729628e-03  -4.38289624e-03
   3.34230601e-03  -5.83191495e-03   2.76562292e-03   9.64506250e-03
  -4.40086517e-03  -6.95919793e-04   7.25256652e-03  -2.95769772e-03
   8.75304639e-03   7.65092811e-03  -2.95318710e-03  -4.25403798e-03
  -5.81071666e-03   7.65157724e-03]
wvec: [ 0.0002832  -0.0021516   0.00353971  0.0089048   0.00940409  0.00621347
 -0.00121681  0.00092658 -0.00260386  0.00332681 -0.00754494  0.00376701
 -0.00030736 -0.00169808  0.00158546  0.00966423  0.00045933 -0.00542138
  0.00395646 -0.00041367  0.00830161 -0.00903824  0.00721408  0.00429652
  0.00455682  0.00209897  0.00623397  0.00084557  0.00857033  0.00552175
 -0.00464948 -0.00802782 -0.00817764  0.00515473  0.00187538 -0.00732329
  0.00198899 -0.00566979 -0.00950923 -0.00550869 -0.00670781  0.00946056
  0.00961993  0.00278376  0.00110024  0.00914957 -0.00091544 -0.00916331
  0.00339898 -0.00054255]
```

```
wvec: [  6.79438002e-03   5.31875482e-03  -9.88891814e-03   2.66075181e-03
  -6.43195491e-03  -9.71881021e-03   3.17339925e-03   9.55363456e-03
   5.64443413e-04  -9.06833666e-05  -9.41581186e-03   5.61945559e-03
   5.18637570e-03  -5.89944143e-03  -2.65461812e-03   7.11631170e-03
   8.46402021e-04  -7.74611067e-03   2.00310536e-03  -5.46753360e-03
   3.18827783e-03   3.22063104e-03  -9.69805103e-03   2.32008356e-03
   1.55467031e-04  -3.69938090e-03  -8.42885114e-03  -6.38264697e-04
  -5.03472402e-04  -8.16413760e-03   6.05225004e-03  -3.55701754e-03
   2.29440560e-03   9.55795124e-03  -6.26058597e-03  -7.73757091e-03
   3.17392149e-03  -2.16309418e-04  -4.96296259e-03  -6.31377613e-03
  -9.32387821e-03  -3.28598823e-03   5.54829976e-03  -9.12261254e-04
  -6.65239664e-03  -7.73076841e-04   2.09365762e-03   4.83063562e-03
   7.97148049e-03   7.88849778e-03]
wvec: [-0.00519954  0.00820932  0.00080061  0.00137945  0.00622367 -0.00934624
 -0.00724957 -0.00932915  0.0023706   0.00865911  0.00970238  0.00327066
  0.0011805  -0.00481626 -0.00857859  0.00761682  0.00322645  0.00828408
 -0.00867775  0.0001234   0.00376284 -0.0058005  -0.00149229  0.0014703
  0.00156599  0.00642583 -0.00796384 -0.00880892  0.00590896  0.00697395
  0.00311856  0.00201523 -0.00430622 -0.00678094 -0.00686245 -0.00372184
  0.00162162 -0.00036233  0.00421439  0.00234655  0.00766121  0.00152749
 -0.00230008  0.00227202 -0.00612525  0.00326373 -0.00284388 -0.00362739
 -0.00754044 -0.00163635]
wvec: [ -1.40334666e-03  -9.70786251e-03  -1.20477506e-03  -2.82296585e-03
   1.73216220e-04  -5.21411747e-03  -9.59732104e-03   1.42392598e-03
   4.31668665e-03   6.87380321e-03  -3.03984177e-03  -2.36374419e-03
   2.75995862e-03   2.42810906e-03   8.42210464e-03  -8.17770895e-04
  -4.13977448e-03   6.32358642e-05   1.87723245e-03   8.05592909e-03
   4.23823763e-03   9.29800607e-03   7.83644558e-04   4.82490472e-03
  -4.73826192e-03   8.58431961e-03  -7.22278608e-03  -7.63412798e-03
  -1.07291527e-03  -3.32138641e-03  -3.64550157e-03   5.22187352e-03
   1.35345419e-03  -5.82535565e-03  -7.60531460e-04  -6.77055854e-04
  -8.02347064e-03   4.16282797e-03  -1.62410041e-04   6.60922518e-03
  -3.29095381e-03   3.80169135e-03  -7.47973472e-03   1.63638731e-03
  -8.04807711e-03  -8.35837796e-03   5.34077501e-03  -5.87049406e-03
   7.64531433e-04  -8.18004180e-03]
wvec: [  4.37464099e-03  -2.74888389e-05   4.29435633e-03  -3.81085160e-03
   2.71993899e-03   4.95806383e-03   1.61968032e-03  -6.24609413e-03
  -4.69881482e-03  -8.90790485e-03  -6.93539763e-03   1.80425367e-03
  -2.13310053e-03  -8.24733730e-03   4.40433016e-03   1.51420000e-03
   4.02007764e-03   5.30163944e-03   8.22434761e-03   3.44689167e-03
  -6.14278438e-03   1.42024690e-03   1.14813633e-03  -7.14709610e-03
   4.01085336e-03   4.64583514e-04   9.45342705e-03   8.44747387e-03
   3.63319507e-03  -7.39431707e-03   6.49299566e-03   6.96398318e-03
   3.01549537e-03   2.66427803e-03   2.80674570e-03   5.29679283e-03
  -7.54928915e-03   3.73655022e-03  -7.10964017e-03   2.30305782e-03
   1.56713161e-03   8.67715292e-03   5.73663414e-03   7.13814079e-05
  -6.59671426e-03  -7.88433570e-03   8.76835175e-03   6.64421497e-03
  -7.17785396e-03   1.89421093e-03]
wvec: [ 0.00554495  0.00667121  0.00602769  0.00419587 -0.00334847 -0.0029082
 -0.00188135  0.00831611  0.00413762 -0.0063552  -0.00614774  0.00872094
  0.00742496 -0.00228342 -0.00310339  0.00913101 -0.00168977 -0.00864125
 -0.00256638  0.00844739 -0.00673473 -0.00700487 -0.00120524  0.00730555
  0.00578113 -0.0070179  -0.00228103 -0.00619523 -0.00655603 -0.00848421
  0.00815881 -0.00355752  0.00080518  0.00916929 -0.00632163 -0.00637335
 -0.00903294  0.00329785 -0.00107092  0.00952503 -0.00363083  0.00745544
  0.0083261   0.00725124 -0.00467164 -0.00249669 -0.006509    0.00419803
 -0.00286134 -0.00270736]
wvec: [-0.00218973 -0.00643705  0.00855845  0.00011338  0.00352613  0.00190064
  0.00762735 -0.00285855 -0.00258389 -0.00636494  0.00683942 -0.00755396
  0.00510394  0.00966507 -0.00470806 -0.00400613  0.00847305 -0.00733965
  0.00110871  0.00655002 -0.0016291  -0.00486845  0.00618962 -0.00911336
  0.00554575  0.00185513  0.00242254 -0.0048668   0.00119299 -0.00348067
 -0.00731566  0.0072178   0.00513821  0.00837473  0.00552174  0.00080857
  0.00337771  0.00609202  0.00608416  0.00913736  0.00018939  0.0026093
 -0.00081795 -0.00721983  0.00107509  0.00735365  0.00356334 -0.00074606
  0.00015824  0.0079576 ]
wvec: [  9.57474764e-03  -5.96104935e-03   1.77405635e-03   1.15405780e-03
  -1.29721942e-03  -4.15798771e-04  -7.52500212e-03  -9.42120887e-03
   3.80579110e-07   3.41701205e-03  -9.86613426e-03   9.04329680e-03
  -1.11154537e-03   8.71823635e-03   1.70977646e-03   9.76117421e-03
  -3.32920998e-03  -9.04311426e-04   1.34755333e-03   9.85828578e-04
  -8.81465618e-03  -5.16002625e-03   4.50679474e-03   6.28248788e-03
  -7.37731485e-03  -3.80313676e-03   6.93051377e-03  -1.49843562e-03
  -5.31920139e-03  -6.81551779e-03  -4.34450619e-03  -8.82079545e-03
  -2.29339092e-03  -3.85187077e-03   5.33103477e-03   6.31664367e-03
  -5.76211186e-03  -3.89432441e-03   2.63126427e-03   2.68897245e-04
  -9.43794847e-03   4.81350534e-03   9.33599193e-03  -1.65486918e-03
```

```
   4.64740256e-03  -1.73448399e-03   9.11988085e-04   8.29079282e-03
   2.65207374e-03  -9.01211612e-03]
wvec: [ -5.40116662e-03  -3.71513912e-03   3.42057296e-03  -3.48039041e-03
  -3.67651432e-04  -2.22379621e-03   2.05016040e-04  -2.94897798e-03
  -2.14935979e-03  -6.94960635e-03   9.87674430e-05   4.10280842e-03
  -2.34019151e-03  -5.97970141e-03   6.78456062e-03  -2.07797647e-03
   9.63194575e-03   2.19477504e-03  -3.66186863e-03  -2.67664390e-03
   3.85940220e-04  -8.76907725e-03  -2.67255073e-03   2.11719089e-04
   5.52469026e-03   2.31486047e-03   2.26594927e-03  -7.96192978e-03
   5.93873579e-03   2.85317795e-03   8.07061419e-03  -8.19576532e-03
   3.97988968e-03   9.33136325e-03   1.23492011e-03  -6.61105337e-03
   8.57632142e-03  -9.37940553e-03  -8.52983911e-03   3.63595109e-03
   1.05707138e-03  -7.77897146e-03   9.90957767e-03  -6.08324166e-03
  -5.79759991e-03   3.83967697e-03  -9.61536262e-03  -7.92068802e-03
   9.88458283e-03   8.10713880e-03]
wvec: [-0.00136407 -0.00066948 -0.00917306 -0.00398661  0.00896962  0.00201081
 -0.00029433 -0.00817395  0.00741933  0.00968601  0.0030921  -0.0031994
 -0.00484868 -0.00900722 -0.00115062 -0.00595361  0.00971398  0.00892655
  0.00267286 -0.00516936  0.006984    0.00140385 -0.00469618 -0.00507542
 -0.00044839  0.00333304  0.00600726  0.00240941  0.00235289 -0.00624719
  0.00558264  0.00053688 -0.00171738 -0.00308625 -0.00367739 -0.00393415
 -0.00567036 -0.00066342  0.00911124 -0.00189401 -0.00424348  0.00894313
 -0.00990681  0.00406902 -0.00951388  0.00260886  0.00827608  0.00354053
 -0.00476287  0.00961952]
wvec: [ 0.00292008 -0.00635679 -0.00339769  0.00691066 -0.00226122 -0.00604684
 -0.00509598 -0.00801856 -0.00978493 -0.00886562 -0.00508334  0.0035298
  0.00513794 -0.00682852  0.00606038  0.00460681  0.00996257  0.00830258
  0.00295034 -0.00013771  0.00904674  0.00758617 -0.00665344 -0.00616821
  0.00267256 -0.00915069 -0.00477755 -0.0089586  -0.00575272 -0.00079089
  0.00539337 -0.00363525 -0.00408065 -0.00608692 -0.00053478 -0.00048703
 -0.00822013 -0.00040869  0.00571409 -0.0069341  -0.00123694  0.00294487
  0.00862745  0.00248825  0.00580114  0.00518899 -0.00865493 -0.001284
 -0.00607004  0.00394245]
wvec: [ -2.45365151e-03  -4.21992596e-03  -7.32518313e-03   2.57910229e-03
   1.74926582e-03   9.51135065e-03   2.61446531e-03   4.11828887e-03
  -2.59721396e-03  -4.42388700e-03  -2.04893225e-03  -4.23021469e-04
  -3.89131065e-03  -2.25622440e-03  -1.50464335e-03  -7.11465534e-03
   4.21745563e-03   9.85432853e-05   8.63815751e-03   1.45158148e-03
   5.20212809e-03   1.24292099e-03  -6.02635555e-03   1.54202944e-03
  -8.54585785e-03   1.72592781e-03  -4.72888537e-03   2.34178663e-03
   6.66621421e-03  -5.88805415e-04   3.67244938e-03  -4.85268841e-03
   3.54953413e-03   3.46508948e-03   6.84242230e-03  -4.47322207e-04
  -3.42017825e-04   1.69649709e-03   7.74058222e-04   9.16015636e-03
  -8.33399687e-03  -7.27116922e-03  -7.22841453e-03   7.10933888e-03
  -7.23335240e-03   7.69400969e-03  -1.08113373e-03  -5.07630408e-03
   6.62370585e-03   6.94320723e-03]
wvec: [ 0.00629745  0.0015162  -0.00011148 -0.00498442  0.0059824  -0.00114104
 -0.00663411 -0.00766132 -0.0068877   0.00242965  0.00680042 -0.00096716
  0.00606777  0.00054741  0.0026987  -0.00389879  0.00749534 -0.00388142
  0.00570045 -0.00837387  0.00575863  0.00370453 -0.00359292  0.00759651
  0.00036745 -0.00875196 -0.00191046  0.00255992  0.00853816 -0.0006748
 -0.00555078 -0.00953534  0.0025527  -0.00567672  0.00647809  0.00500213
 -0.0030063  -0.0077187  -0.00168382 -0.0037725  -0.00764509  0.00382244
 -0.00932595  0.00695462  0.00059332 -0.00163174 -0.00627625 -0.00303161
 -0.00933513 -0.00313016]
wvec: [-0.00519954  0.00820932  0.00080061  0.00137945  0.00622367 -0.00934624
 -0.00724957 -0.00932915  0.0023706   0.00865911  0.00970238  0.00327066
  0.0011805  -0.00481626 -0.00857859  0.00761682  0.00322645  0.00828408
 -0.00867775  0.0001234   0.00376284 -0.0058005  -0.00149229  0.0014703
  0.00156599  0.00642583 -0.00796384 -0.00880892  0.00590896  0.00697395
  0.00311856  0.00201523 -0.00430622 -0.00678094 -0.00686245 -0.00372184
  0.00162162 -0.00036233  0.00421439  0.00234655  0.00766121  0.00152749
 -0.00230008  0.00227202 -0.00612525  0.00326373 -0.00284388 -0.00362739
 -0.00754044 -0.00163635]
wvec: [-0.00694911 -0.00897014 -0.00031501  0.00613629  0.00133191 -0.00038062
 -0.00660661 -0.00264945  0.00426468  0.0024483  -0.00345829 -0.0019306
  0.0077408  -0.00526361 -0.00534773 -0.00881806  0.00574405 -0.00428891
 -0.00752039  0.00568492  0.00117445  0.00967407 -0.00688154 -0.00146223
 -0.00130197  0.00384016  0.0093052  -0.00571343  0.0076428   0.00977523
 -0.00748814  0.00069932  0.00900444  0.0027872  -0.00866302  0.00413508
  0.00961896 -0.00415781 -0.00799312  0.0048361   0.00117751  0.00936286
  0.00073475  0.00147025 -0.00036064  0.00527647  0.00740858  0.00656593
  0.00519378  0.00630302]
sent_vec: [-0.01484765 -0.00161551 -0.0006038   0.03014196  0.04783563 -0.0387774
 -0.05262202 -0.0328651  -0.03706586  0.04943917 -0.02764561  0.05226702
  0.01286704 -0.06018233 -0.03752621  0.03191881  0.04051867 -0.00911924
 -0.0054489   0.03433927  0.04138874 -0.00458235 -0.03685561  0.01899837
```

```
  0.01289452  0.00188694 -0.0217694  -0.05345692  0.04770048 -0.00328929
 -0.00013868  0.00459592  0.00098426  0.01629029 -0.03403945 -0.03330434
 -0.00438104 -0.01788741 -0.02810653  0.03598764 -0.03425311  0.0536603
  0.01575358  0.0123996  -0.04963305  0.03126643  0.0088119  -0.02169761
 -0.02898515  0.04331266]
avg_vec: [ -6.45550124e-04  -7.02397302e-05  -2.62519819e-05   1.31052003e-03
   2.07980980e-03  -1.68597392e-03  -2.28791379e-03  -1.42891745e-03
  -1.61155906e-03   2.14952897e-03  -1.20198303e-03   2.27247908e-03
   5.59436345e-04  -2.61662311e-03  -1.63157420e-03   1.38777421e-03
   1.76168134e-03  -3.96488769e-04  -2.36908870e-04   1.49301193e-03
   1.79951061e-03  -1.99232493e-04  -1.60241786e-03   8.26015979e-04
   5.60631365e-04   8.20407092e-05  -9.46495647e-04  -2.32421391e-03
   2.07393383e-03  -1.43012576e-04  -6.02941019e-06   1.99822625e-04
   4.27939357e-05   7.08273551e-04  -1.47997622e-03  -1.44801476e-03
  -1.90479973e-04  -7.77713451e-04  -1.22202320e-03   1.56468021e-03
  -1.48926568e-03   2.33305663e-03   6.84938376e-04   5.39113246e-04
  -2.15795888e-03   1.35940996e-03   3.83126291e-04  -9.43374537e-04
  -1.26022395e-03   1.88315919e-03]
*******************************************************************
CPU times: user 48 ms, sys: 0 ns, total: 48 ms
Wall time: 43.8 ms
```

In [63]:

```
%%time
np.seterr(divide='ignore', invalid='ignore')
avg_vec = [] #List to store all the avg w2vec's
for sent in final_string:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(50) #Initializing with zeroes
    for word in sent:
        try:
            wvec = w2v_model.wv[word] #Vector of each using w2v model
            sent_vec += wvec #Adding the vectors
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
    sent_vec /= cnt #Taking average of vectors sum of the particular review
    avg_vec.append(sent_vec) #Storing the avg w2vec's for each review
    #print("*******************************************************************")
    # Average Word2Vec
```

```
CPU times: user 1min 26s, sys: 0 ns, total: 1min 26s
Wall time: 1min 26s
```

In [ ]:

```
#Saving the variable to access later without recomputing
savetofile(avg_vec,"avg_w2v_vec")
```

In [4]:

```
#Loading the variable from file
avg_vec = openfromfile("avg_w2v_vec")
```

In [40]:

```
avg_vec = np.array(avg_vec)
avg_vec.shape
```

Out[40]:

```
(364171, 50)
```

In [33]:

```
from sklearn import preprocessing
avg_vec_norm = preprocessing.normalize(avg_vec)
```

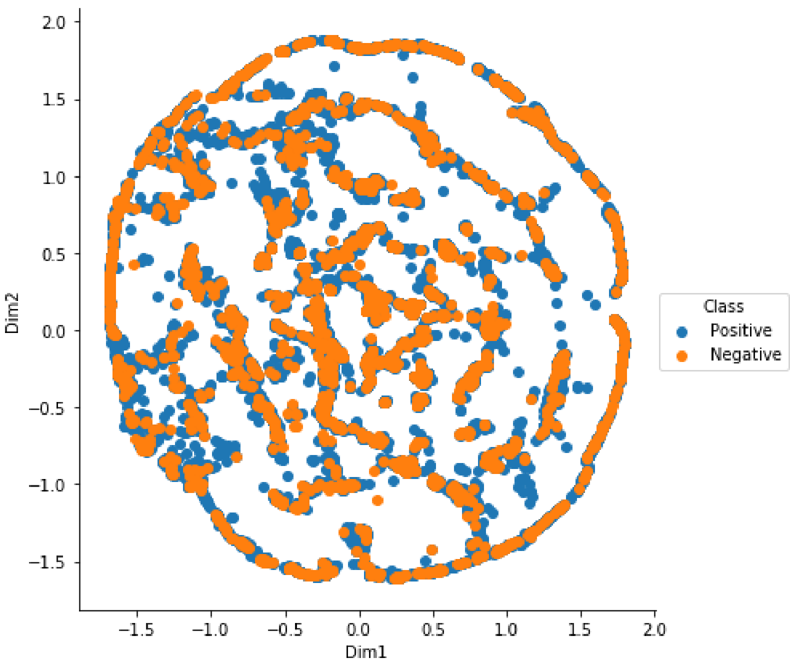
In [21]:

```
%%time
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 20000
sample_cols = random.sample(range(1, avg_vec.shape[0]), n_samples)
sample_features = avg_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
```

```
(20000, 50) (20000, 1)
TSNE done in 767.050s.
(20000, 3)
CPU times: user 11min 46s, sys: 1min, total: 12min 47s
Wall time: 12min 47s
```

In [22]:

```
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```



In [30]:

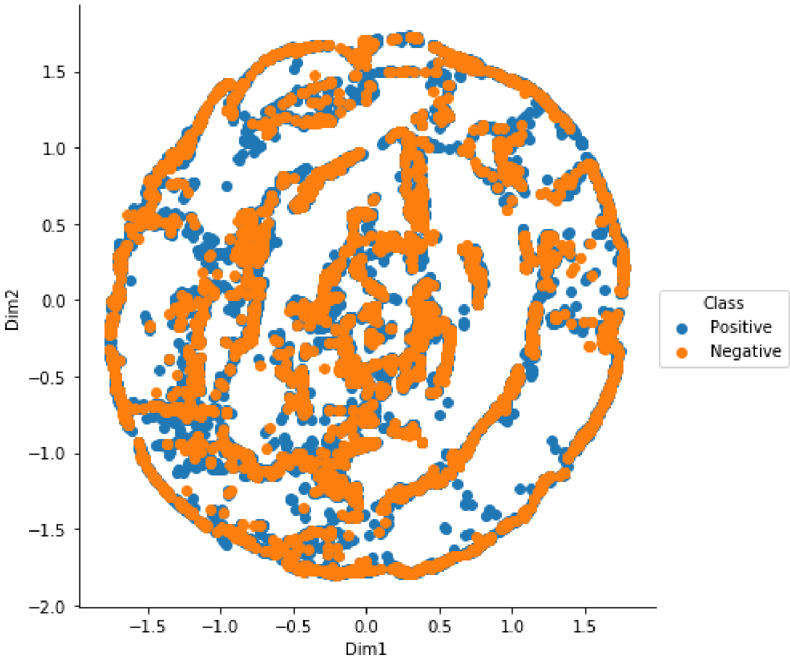
```
%%time
from sklearn.manifold import TSNE
from time import time
import random

n_samples = 40000
sample_cols = random.sample(range(1, avg_vec.shape[0]), n_samples)
sample_features = avg_vec[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)
```

```
embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
```

```
(40000, 50) (40000, 1)
(40000, 3)
CPU times: user 26min 9s, sys: 1min 50s, total: 27min 59s
Wall time: 28min
```

In [31]:

```
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```



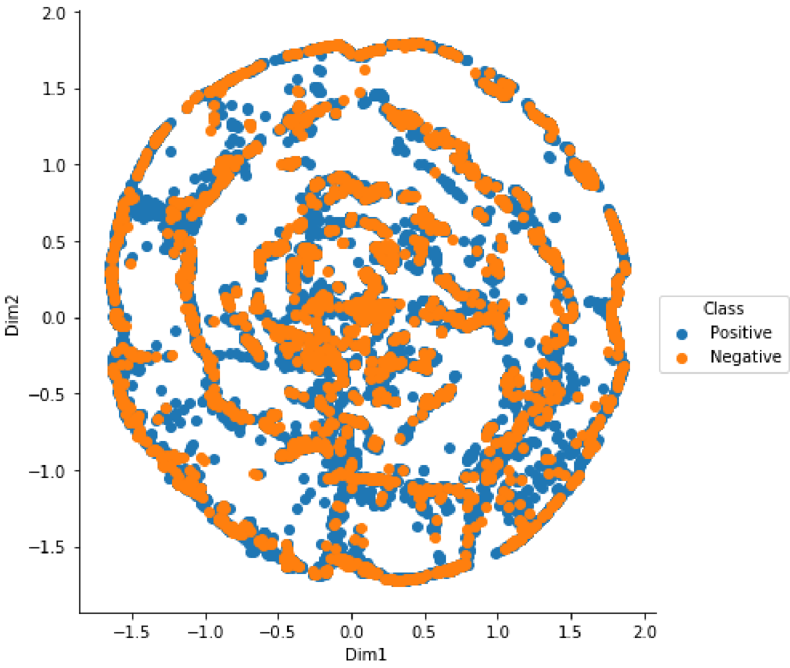
In [38]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 20000
sample_cols = random.sample(range(1, avg_vec_norm.shape[0]), n_samples)
sample_features = avg_vec_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
```

```
(20000, 50) (20000, 1)
(20000, 3)
CPU times: user 8min 32s, sys: 47.3 s, total: 9min 19s
Wall time: 9min 19s
```

In [39]:

```
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```



4.b Using Google's Trained W2Vec on Google News

In [3]:

```
from gensim.models import KeyedVectors

w2v_model_google = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=
True) #Loading the model from file in the disk
```

In [4]:

```
w2v_vocub = w2v_model_google.wv.vocab
len(w2v_vocub)
```

Out[4]:

```
3000000
```

In [5]:

```
w2v_model_google.wv.most_similar('like')
```

Out[5]:

```
[('really', 0.5752447843551636),
 ('weird', 0.5676319599151611),
 ('crazy', 0.5382447838783264),
 ('kind', 0.5310239195823669),
 ('maybe', 0.5220045447349548),
 ('loooove', 0.5187614560127258),
 ('anymore', 0.5177680253982544),
 ('Kinda_reminds', 0.5151872634887695),
 ('definitely', 0.5117843151092529),
 ('kinda_fishy', 0.5090124607086182)]
```

In [39]:

```
w2v_model_google.wv.most_similar('taste')
```

Out[39]:

```
[('tastes', 0.6838272213935852),
 ('flavor', 0.6630197763442993),
 ('tasted', 0.6162090301513672),
```

```
 ('Harry_Potter_butterbeer', 0.5894586443901062),
 ('tasting', 0.5604724884033203),
 ('tangy_taste', 0.5567916035652161),
 ('aftertaste', 0.5558385252952576),
 ('bitter_taste', 0.5491952300071716),
 ('carbonated_cough_syrup', 0.5455324053764343),
 ('taste_buds', 0.5368086695671082)]
```

In [50]:

```
w2v_model_google.wv["word"].size
```

Out[50]:

```
300
```

In [107]:

```
%%time
avg_vec_google = [] #List to store all the avg w2vec's
no_datapoints = 364170
sample_cols = random.sample(range(1, no_datapoints), 20001)
for sent in df2['CleanedText_NoStem'].values[sample_cols]:
    cnt = 0 #to count no of words in each reviews
    sent_vec = np.zeros(300) #Initializing with zeroes
#     print("sent:",sent)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#             print(word)
            wvec = w2v_model_google.wv[word] #Vector of each using w2v model
#             print("wvec:",wvec)
            sent_vec += wvec #Adding the vectors
#             print("sent_vec:",sent_vec)
            cnt += 1
        except:
            pass #When the word is not in the dictionary then do nothing
#     print(sent_vec)
    sent_vec /= cnt #Taking average of vectors sum of the particular review
#     print("avg_vec:",sent_vec)
    avg_vec_google.append(sent_vec) #Storing the avg w2vec's for each review
#     print("*******************************************************************")
# print(avg_vec_google)
avg_vec_google = np.array(avg_vec_google)
```

```
CPU times: user 9.89 s, sys: 4 ms, total: 9.89 s
Wall time: 9.89 s
```

In [108]:

```
#Saving the variable to access later without recomputing
savetofile(avg_vec_google,"avg_w2v_vec_google")
```

In [109]:

```
#Loading the variable from file
avg_vec_google = openfromfile("avg_w2v_vec_google")
```

In [110]:

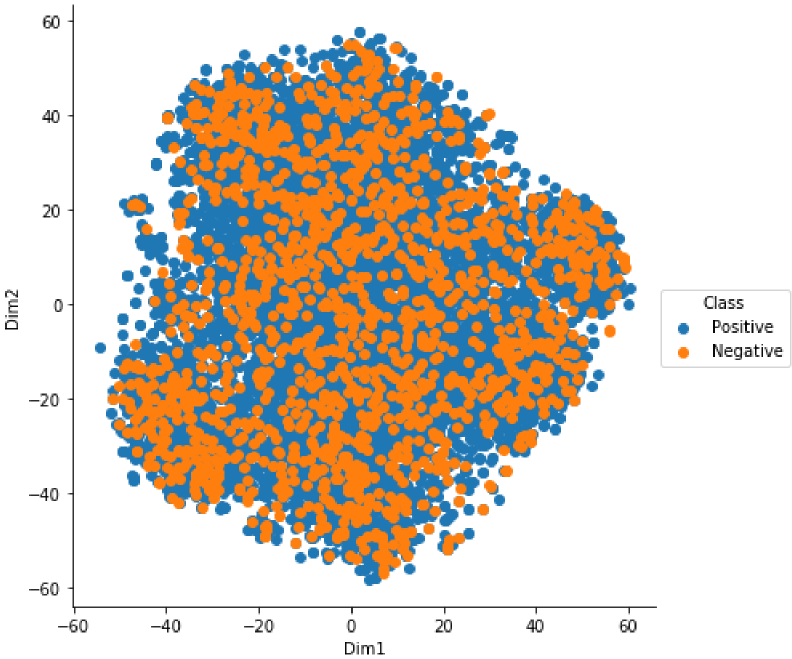
```
from sklearn import preprocessing

avg_vec_google_norm = preprocessing.normalize(avg_vec_google)
```

In [115]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
```

```
n_samples = 10000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```

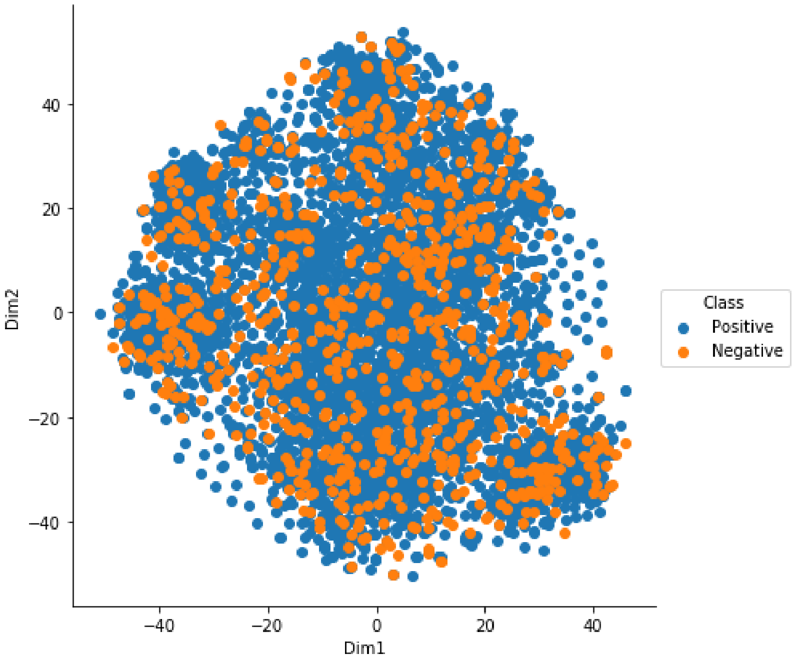



```
CPU times: user 6min 32s, sys: 31.6 s, total: 7min 4s
Wall time: 7min 3s
```

In [121]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 5000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(5000, 300) (5000, 1)
(5000, 3)
```

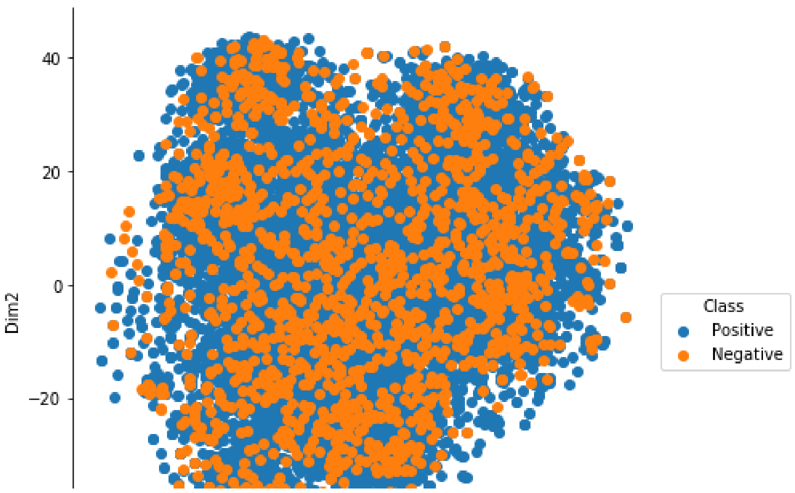


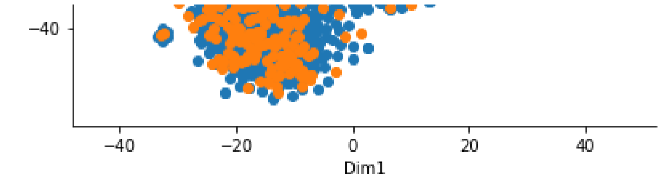

```
CPU times: user 2min 43s, sys: 16.4 s, total: 2min 59s
Wall time: 2min 59s
```

In [117]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```



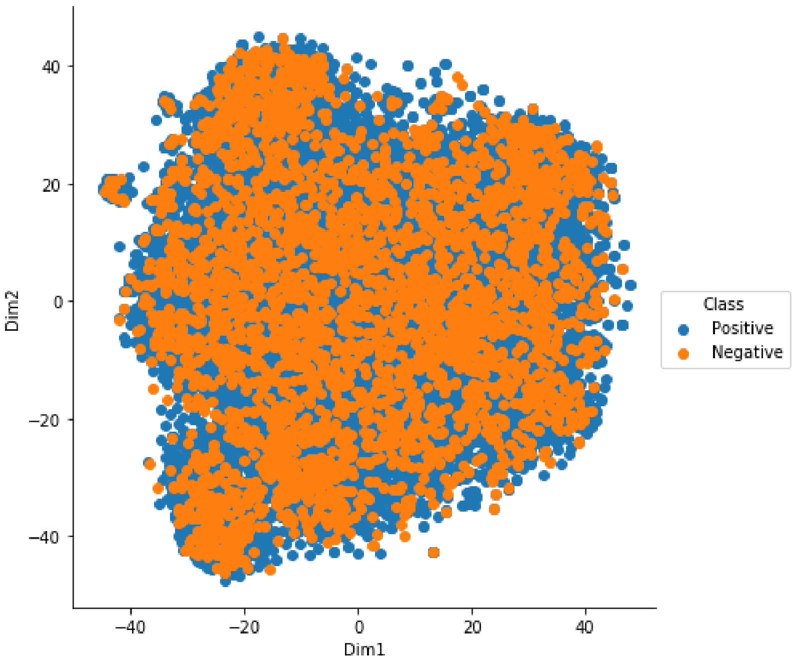



```
CPU times: user 7min 14s, sys: 33 s, total: 7min 47s
Wall time: 7min 47s
```

In [118]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 20000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=30)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(20000, 300) (20000, 1)
(20000, 3)
```



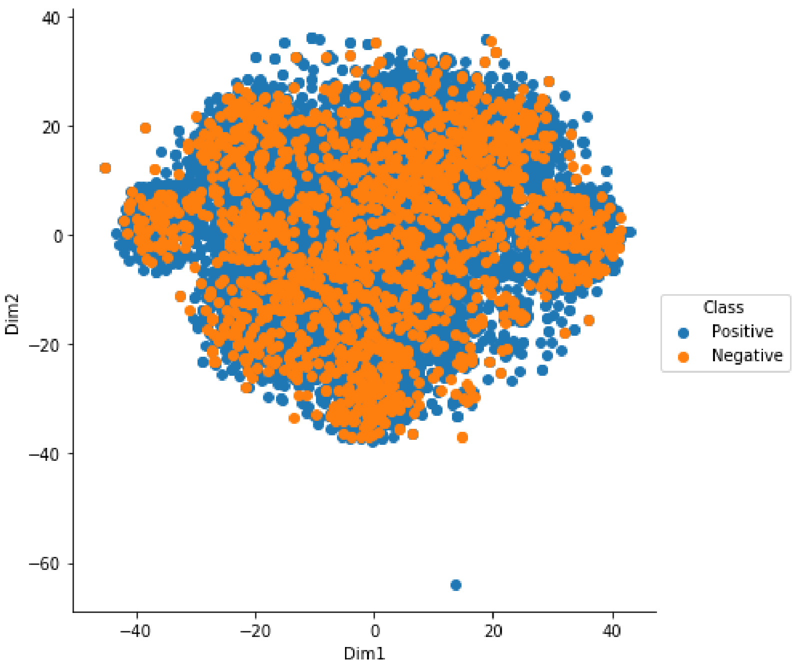

```
CPU times: user 27min 48s, sys: 1min 7s, total: 28min 55s
Wall time: 28min 55s
```

In [119]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
```

```
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=35)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```

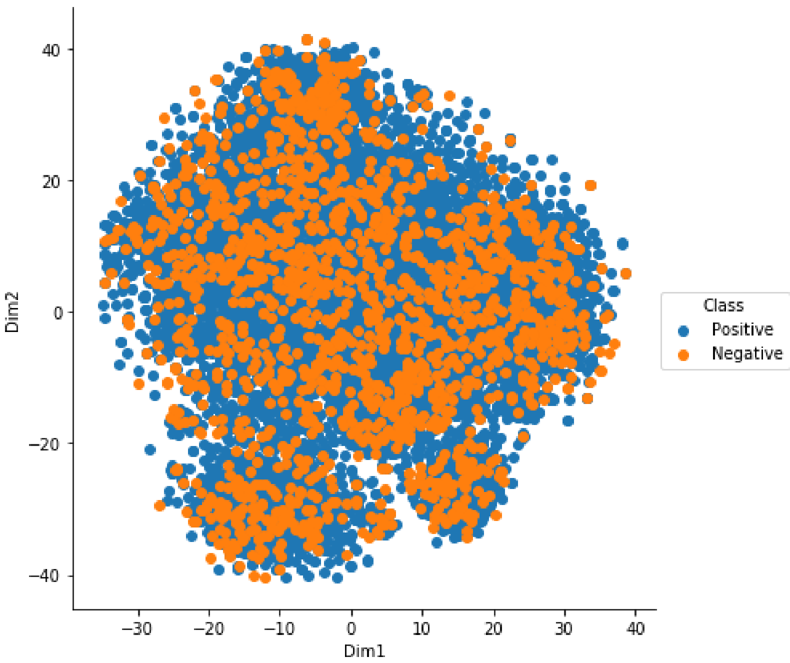



```
CPU times: user 7min 44s, sys: 32.9 s, total: 8min 16s
Wall time: 8min 16s
```

In [120]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, avg_vec_google.shape[0]), n_samples)
sample_features = avg_vec_google[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=40)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```




```
CPU times: user 8min 13s, sys: 32 s, total: 8min 45s
Wall time: 8min 44s
```

5. Tf-idf W2Vec

- Another way to covert sentence into vectors
- Take weighted sum of the vectors divided by the sum of all the tfidf's
  i.e. (tfidf(word) x w2v(word))/sum(tfidf's)

In [19]:

```
#Taking Sample of 20k points
no_datapoints = 364170
sample_cols = random.sample(range(1, no_datapoints), 20001)
```

In [20]:

```
%%time
###tf-idf with No Stemming
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
tfidf_vec_ns = tfidf.fit_transform(df2['CleanedText_NoStem'].values[sample_cols])

#Saving the variable to access later without recomputing
# savetofile(tfidf_vec,"tfidf")

#Loading the variable from file
# tfidf_vec = openfromfile("tfidf")

print(tfidf_vec_ns.shape)

# tf-idf came up with 2.9 million features for the data corpus
from sklearn.decomposition import TruncatedSVD

tsvd_tfidf_ns = TruncatedSVD(n_components=300)#No of components as total dimensions
tsvd_tfidf_vec_ns = tsvd_tfidf_ns.fit_transform(tfidf_vec_ns)
print(tsvd_tfidf_ns.explained_variance_ratio_[:].sum())
features = tfidf.get_feature_names()
```

```
(20001, 492431)
0.110114446613
CPU times: user 3min 56s, sys: 5.88 s, total: 4min 2s
Wall time: 58 s
```

In [21]:

```
%%time
tfidf_w2v_vec_google = []
review = 0

for sent in df2['CleanedText_NoStem'].values[sample_cols]:
    cnt = 0
    weighted_sum  = 0
    sent_vec = np.zeros(300)
    sent = sent.decode("utf-8")
    for word in sent.split():
        try:
#             print(word)
            wvec = w2v_model_google.wv[word] #Vector of each using w2v model
#             print("w2vec:",wvec)
#             print("tfidf:",tfidf_vec_ns[review,features.index(word)])
            tfidf = tfidf_vec_ns[review,features.index(word)]
#             print(tfidf)
            sent_vec += (wvec * tfidf)
            weighted_sum += tfidf
        except:
            pass
    sent_vec /= weighted_sum
    tfidf_w2v_vec_google.append(sent_vec)
    review += 1
```

```
CPU times: user 3h 20min 51s, sys: 1.32 s, total: 3h 20min 52s
Wall time: 3h 20min 53s
```

In [22]:

```
len(tfidf_w2v_vec_google)
```

Out[22]:

```
20001
```

In [23]:

```
len(tfidf_w2v_vec_google[0])
```

Out[23]:

```
300
```

In [24]:

```
tfidf_w2v_vec_google[5]
```

Out[24]:

```
array([  6.30765966e-03,   2.62348772e-02,  -1.28013094e-02,
         1.66244870e-01,  -4.49297504e-02,   2.66082968e-02,
         8.06051421e-02,  -8.27518457e-02,   7.73820410e-03,
         1.28079768e-01,   4.09113582e-03,  -1.26601291e-01,
        -3.10158627e-02,   3.02259649e-02,  -1.17419000e-01,
         1.19921784e-01,   3.36843358e-02,   1.50723015e-01,
         4.26849213e-02,  -3.13014550e-02,  -7.91879333e-02,
         5.21383487e-02,  -1.22338065e-02,  -1.99465251e-03,
         7.86969992e-02,  -8.04239890e-03,  -4.38397773e-02,
         5.46432782e-02,  -7.09601715e-03,   4.12984907e-02,
        -6.55848419e-02,   1.60784459e-03,   2.39487639e-02,
         2.04896547e-02,   3.50686609e-02,   4.94815506e-04,
         4.68977209e-02,  -8.77567649e-02,  -1.13872285e-02,
         1.18984474e-01,   9.81944115e-02,  -9.27291092e-02,
         1.36931440e-01,  -1.44343861e-02,  -2.90870869e-02,
        -9.79914776e-02,  -3.98462383e-02,  -2.20645862e-02,
         6.52701948e-03,  -2.17011543e-03,  -4.47150526e-02,
         8.24235868e-03,   4.96269734e-03,  -1.31958030e-02,
        -2.56885586e-04,   2.93922949e-02,  -1.72273889e-02,
        -3.99719652e-02,   3.77720190e-02,  -7.03517832e-02,
        -1.83692687e-02,   1.05203373e-01,  -6.79548702e-02,
         1.20978545e-02,   2.00720254e-02,  -2.57170686e-02,
```

```
-1.04653269e-01,  1.09477380e-02, -1.10814938e-02,
 3.58060937e-02,  5.00074362e-02, -1.68985071e-02,
 7.69999519e-02,  3.58910598e-02, -1.79429434e-01,
-8.14919239e-02, -3.09866506e-02,  1.86764700e-02,
 1.23387173e-02,  8.39024587e-02, -2.43708528e-02,
-7.07842955e-02,  5.96689909e-02,  9.32226986e-03,
-1.34524508e-01, -6.27895249e-02, -5.73845887e-02,
 1.56383474e-01,  1.06397445e-02,  1.48438283e-03,
 2.26006845e-03,  5.94836433e-02, -4.01468413e-02,
-4.94019327e-02, -7.73990009e-02, -8.28149211e-02,
 4.55678354e-02,  7.98633764e-03, -3.77175618e-03,
-1.42583838e-03, -4.15781523e-02, -1.98997212e-02,
 3.87199972e-02,  4.62500780e-02,  1.30744853e-03,
-9.77887918e-02,  8.19120729e-03, -3.97796475e-02,
-5.00299838e-02, -1.15073477e-01, -4.50036113e-02,
-1.88494720e-02,  2.04927493e-02, -3.89075545e-02,
 1.02195050e-01, -2.22327215e-02,  1.84280325e-02,
-4.10772902e-02,  3.64501879e-02,  1.85647640e-02,
-3.17008827e-02,  2.03754272e-02, -1.44728444e-02,
 4.26561627e-02, -3.85413204e-02, -5.12846338e-03,
-1.48223576e-02, -2.59871538e-02,  3.98556216e-02,
 7.85663341e-02, -1.13784874e-01, -9.24403775e-02,
-4.79589881e-02,  6.66433462e-02, -4.22474434e-02,
 3.55220688e-02,  1.97905783e-02,  6.28166382e-03,
 1.72767084e-02,  1.05971750e-01,  3.86894163e-02,
-8.53412200e-02, -3.02475745e-02, -3.61943713e-02,
 3.69913128e-02,  1.11016871e-02,  1.47543993e-02,
-6.10027457e-02, -1.18417849e-01, -8.47019800e-02,
 1.17743947e-01,  8.63746347e-02, -1.34610903e-01,
 1.02051052e-01, -5.70805444e-02, -3.03150640e-03,
-9.63756876e-02, -1.21393920e-01, -9.56149872e-02,
-7.31430816e-03, -2.23404034e-02,  1.00559661e-01,
 4.38631475e-02,  8.85652425e-02,  2.95088167e-02,
-8.00676853e-02,  2.59932720e-02, -3.91580068e-02,
-3.80113929e-04, -2.56918011e-02, -1.59347885e-01,
 1.38829789e-02, -2.24706463e-02,  1.35188281e-03,
-1.05755187e-01, -9.07731286e-04,  1.00735886e-01,
-2.53611876e-03, -9.98537248e-03,  7.41369657e-02,
-1.02420457e-01, -7.81483222e-02, -6.00523651e-03,
 4.45330631e-02,  1.15630051e-02, -4.62334250e-02,
 1.13043760e-02,  3.12486471e-02,  1.31223589e-02,
 2.73339873e-02, -1.86421712e-02,  2.96892040e-02,
 3.29694047e-02, -2.53753550e-02,  2.21723200e-03,
 1.00622369e-01,  4.17252668e-02,  1.43646585e-02,
-6.63928885e-02, -9.86401491e-02,  1.34527153e-02,
 5.02574390e-02, -7.46830584e-02,  8.79070400e-03,
-6.93407360e-03, -2.08331295e-02, -5.54051442e-02,
-3.00332546e-02, -8.14759126e-04, -2.51196201e-02,
-6.94586798e-02,  4.05398320e-02, -5.10815142e-02,
 9.32799142e-02, -3.32990062e-02,  2.60039576e-02,
 9.18909223e-02, -2.57231292e-03, -4.04243322e-02,
 8.76672893e-03, -3.58291529e-03,  7.30099091e-02,
-7.91123210e-03,  2.16569243e-02,  4.95658906e-02,
-2.20735234e-02, -6.51081920e-03,  1.20510640e-02,
-1.24278356e-02, -5.68246277e-03,  2.92287371e-02,
-3.20580019e-02,  2.72158927e-02,  4.54193312e-02,
 5.74600210e-02, -5.85809826e-02,  5.02565618e-03,
-8.40504219e-02,  9.05609831e-02, -1.82824764e-02,
 7.36574106e-02,  3.77906419e-02,  4.03374567e-03,
-8.66875252e-02,  6.04039318e-03,  3.74840825e-02,
-1.82967071e-02,  4.17376491e-02,  1.09054267e-02,
-7.05850356e-02,  2.71768480e-02,  5.79108278e-02,
 1.28444185e-01,  5.39408911e-02,  6.82060182e-02,
-1.01226069e-01, -3.37983553e-02, -5.94959429e-02,
-4.71564813e-02, -5.33480337e-02, -5.77451520e-02,
-1.24424558e-02, -8.77421416e-02,  2.99121135e-02,
-1.13357725e-02,  2.40719855e-02, -8.88173492e-05,
 1.27712088e-02, -3.13910613e-02,  8.81666745e-03,
 6.15059880e-02,  8.98287157e-02,  1.25739757e-01,
 2.16616150e-02,  8.84198771e-02, -1.17471983e-01,
-8.62687478e-02, -1.92099273e-02, -4.58803753e-02,
-7.02773713e-02,  6.30421877e-03,  1.22709289e-02,
 2.98380458e-02,  5.46972339e-02, -1.53963893e-02,
-6.30442083e-02, -7.03832803e-02,  5.20401001e-02,
 2.40175645e-02,  1.15646546e-01, -7.09699871e-02,
 2.06148266e-02, -8.14377215e-02, -2.05277963e-02,
 2.38067638e-02, -7.02949280e-03, -6.84436059e-03,
```

```
-3.07543165e-02,   2.34512462e-02,  -1.67175623e-02])
```

In [25]:

```
savetofile(tfidf_w2v_vec_google,"tfidf_w2v_vec_google")
```

In [3]:

```
tfidf_w2v_vec_google = openfromfile("tfidf_w2v_vec_google")
```

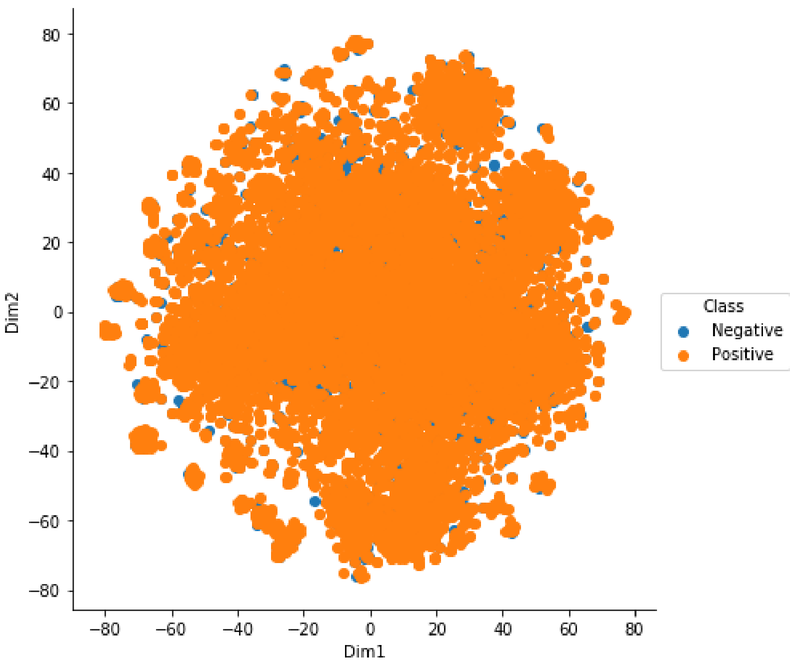
In [4]:

```
from sklearn import preprocessing
tfidf_w2v_vec_google_norm = preprocessing.normalize(tfidf_w2v_vec_google)
```

In [28]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=20)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```

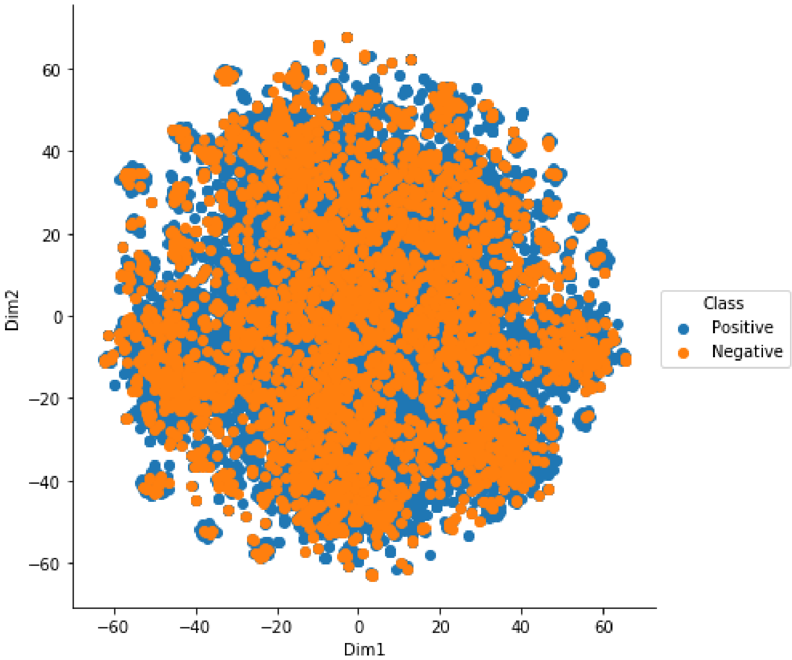



```
CPU times: user 6min 7s, sys: 32.6 s, total: 6min 40s
Wall time: 6min 40s
```

In [30]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 20000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=35)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(20000, 300) (20000, 1)
(20000, 3)
```



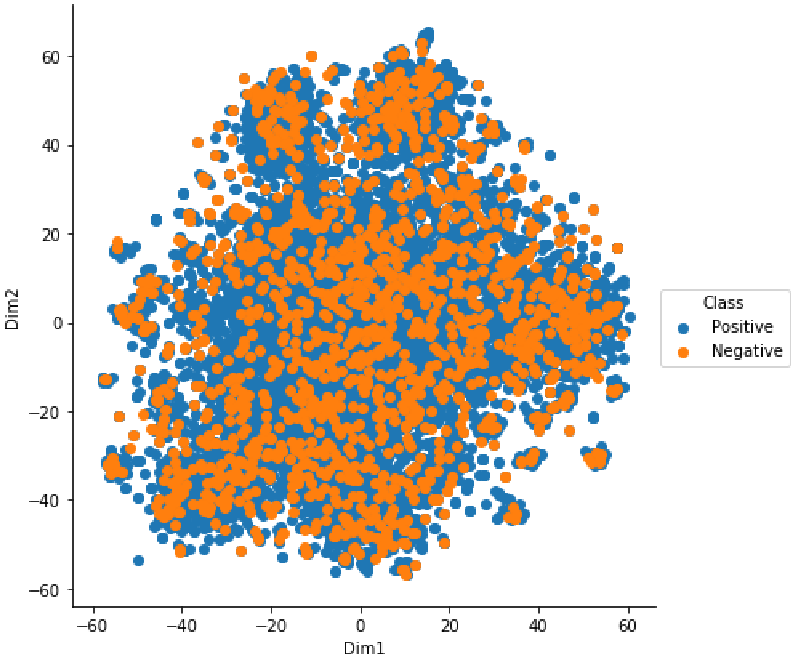

```
CPU times: user 17min 6s, sys: 1min 3s, total: 18min 9s
Wall time: 18min 9s
```

In [31]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=40)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
```

```
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```

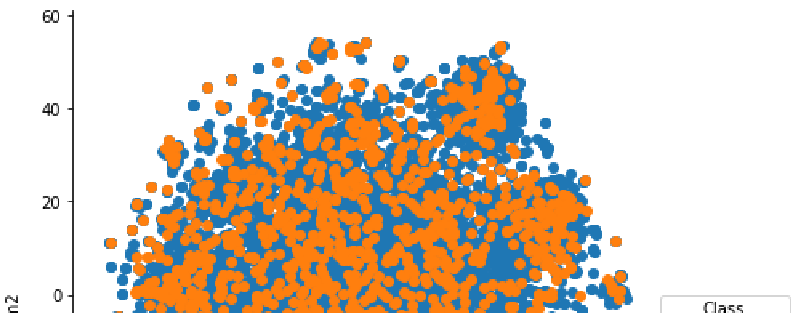


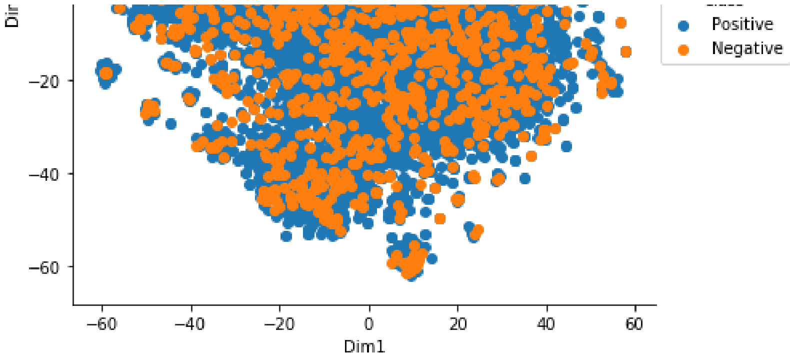

```
CPU times: user 12min 10s, sys: 33.6 s, total: 12min 43s
Wall time: 12min 43s
```

In [32]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=45)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```



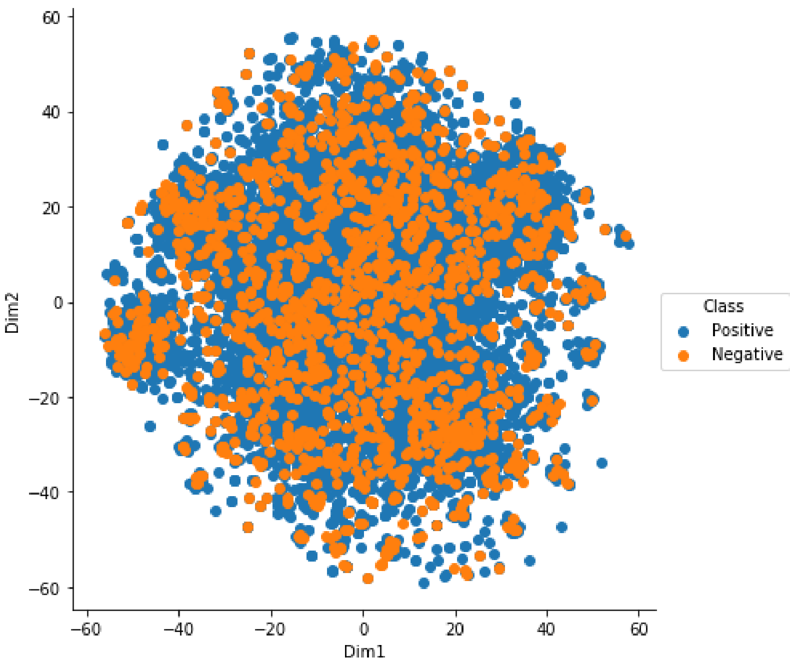



```
CPU times: user 7min 46s, sys: 32.3 s, total: 8min 18s
Wall time: 8min 18s
```

In [34]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=50)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```

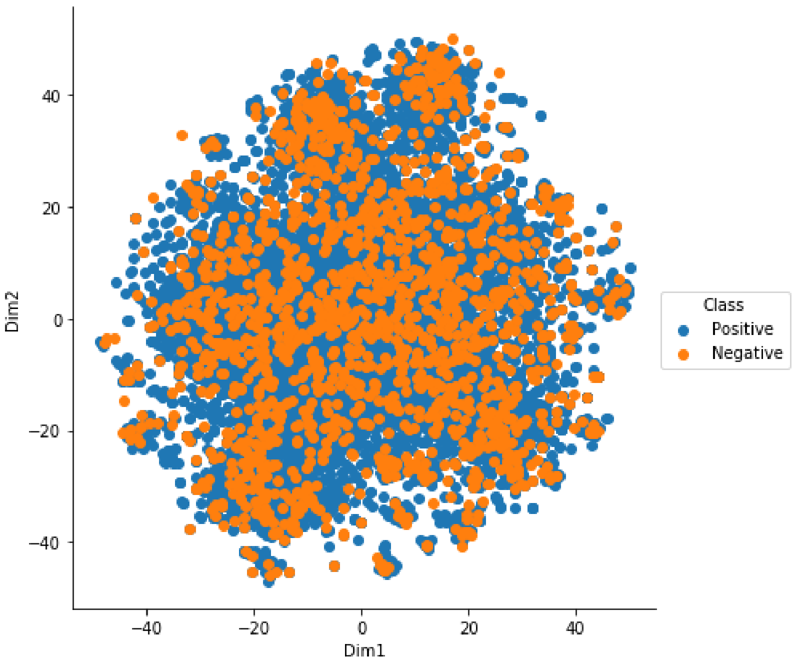



```
CPU times: user 8min 47s, sys: 32.5 s, total: 9min 20s
Wall time: 9min 20s
```

In [7]:

```
%%time
from sklearn.manifold import TSNE
import random

n_samples = 10000
sample_cols = random.sample(range(1, tfidf_w2v_vec_google_norm.shape[0]), n_samples)
sample_features = tfidf_w2v_vec_google_norm[sample_cols]
# sample_features = df
sample_class = df2['Score'][sample_cols]
sample_class = sample_class[:,np.newaxis]
print(sample_features.shape,sample_class.shape)
model = TSNE(n_components=2,random_state=0,perplexity=70)

embedded_data = model.fit_transform(sample_features)
# print(embedded_data.shape,sample_class.shape)
final_data = np.concatenate((embedded_data,sample_class),axis=1)
print(final_data.shape)
newdf = pd.DataFrame(data=final_data,columns=["Dim1","Dim2","Class"])
sns.FacetGrid(newdf,hue="Class",size=6).map(plt.scatter,"Dim1","Dim2").add_legend()
plt.show()
```

```
(10000, 300) (10000, 1)
(10000, 3)
```




```
CPU times: user 10min 1s, sys: 24.7 s, total: 10min 26s
Wall time: 10min 26s
```

Conclusions from TSNE plots

Most of TSNE plot shows that data is quite overlapping hence we can't be sure that data is linearly sepearable but as TSNE is an approximation algorithm we can't be sure fo this claim

Hence we need to make models and test it ourselves

If the data from the TSNE plots would had been seen to seperable it would have easily seperable using any linear model

References:

(1) http://blog.aylien.com/10-common-nlp-terms-explained-for-the-text/
(2) https://en.wikipedia.org/
(3) https://buhrmann.github.io/tfidf-analysis.html