Type *Markdown* and LaTeX: α^2

In [2]:

```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer



# using the SQLite Table to read data.
con = sqlite3.connect('C:\\Users\\ashwani\\Desktop\\tab\\Applied course\\Amazon Reviews\\da



#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3
""", con)




# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

In [3]:

```
filtered_data.head()
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [4]:

```
#now we will do deduplication. for that we need to find out if there are any duplicate entr
#first sorting is required.
sorted_data=filtered_data.sort_values("ProductId",axis=0,ascending=True,kind='quicksort',ir
sorted_data.head()
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [5]:

```
deduplicated_data= sorted_data.drop_duplicates(subset={"UserId","Profile Name","Time","Text
```

In [6]:

```
deduplicated_data.head()
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [7]:

```
sorted_data.shape
```

Out[7]:

```
(525814, 10)
```

In [8]:

```
deduplicated_data.shape
```

Out[8]:

```
(364133, 10)
```

In [9]:

```
#checking invalid entries for helpfullness denom and nr
disp= pd.read_sql_query('''
select *
From Reviews
where HelpfulnessNumerator > HelpfulnessDenominator
''',con)
```

In [10]:

```
disp
```

Out[10]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenc
0	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	
1	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	

In [11]:

```
final= deduplicated_data[deduplicated_data["HelpfulnessNumerator"]<=deduplicated_data["Help
final.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [12]:

```
type(final)
```

Out[12]:

```
pandas.core.frame.DataFrame
```

In [13]:

```
final.shape
```

Out[13]:

```
(364131, 10)
```

In [14]:

```
#checking if the dtaset is balanced or not
final["Score"].value_counts()
```

Out[14]:

```
positive    307026
negative     57105
Name: Score, dtype: int64
```

In [15]:

```
#not balanced
```

Type *Markdown* and LaTeX: α^2

In [16]:

```
import re
count=0
for sent in final["Text"]:

        if (re.search('<.*?>',sent)):
            #print("\n"+sent)
            count+=1
print(count)     #number of text sentences containing html tag
```

```
93144
```

In [17]:

```
#remove html tag
import string
from nltk.corpus import stopwords
import nltk
#stop words
nltk.download('stopwords')


stop= set(nltk.corpus.stopwords.words("english"))
print(stop)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ashwani\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
{'mustn', 'theirs', 'you', 'did', 'does', 'him', 'it', "it's", 'off', 'ain',
"didn't", 'but', 'be', 'an', 'that', 'then', "doesn't", "isn't", 'your', 'su
ch', 'which', 'its', 'with', 'those', 'same', "you'd", 'should', 's', 'dow
n', 'd', 've', 'hasn', "mightn't", 'were', 'there', 'this', 'very', 'hadn',
"wasn't", 'myself', 'don', 'whom', 're', 'over', 'own', 'his', 'isn', 'were
n', 'doesn', 'me', 'we', "weren't", 'was', "don't", 'above', 'nor', 'of', 'd
o', 'until', "you've", 'from', 'each', "you're", 'yourselves', 'between', 'm
ore', 'had', 'some', 'can', 'doing', "hadn't", 'wasn', 'won', "haven't", 'mi
ghtn', 'll', 'a', "that'll", 'while', 'shouldn', 'about', 'ma', 't', 'thes
e', "won't", 'hers', 'herself', 'they', 'i', 'wouldn', 'why', 'further', 'th
em', 'into', 'against', 'what', 'out', 'been', 'for', 'under', 'yours', 'wil
l', 'on', 'too', 'because', 'after', 'how', "hasn't", "aren't", 'couldn', 't
hrough', 'aren', 'most', 'other', "needn't", 'to', "shouldn't", 'again', 'i
n', 'haven', "she's", 'when', 'are', 'below', 'o', 'now', 'm', 'once', 'did
n', 'so', "mustn't", 'y', "you'll", 'himself', 'our', 'itself', 'if', "would
n't", 'by', 'than', 'before', 'any', 'is', 'both', "couldn't", 'yourself',
'ourselves', 'and', 'not', 'ours', 'no', 'the', 'needn', 'my', 'their', 'sh
e', 'themselves', 'only', 'here', 'or', 'have', 'just', 'few', 'who', 'wher
e', 'am', 'having', "should've", 'during', 'being', 'has', 'he', 'all', 'a
t', "shan't", 'up', 'shan', 'her', 'as'}
```

In [18]:

```
#stemming
snow = nltk.stem.SnowballStemmer('english')
snow.stem('beauty')
```

Out[18]:

```
'beauti'
```

In [19]:

```
#cleaning html

def cleanhtml(sentence): #function to clean the word of any html-tags
    #cleanr = re.compile('<.*?>') #regex
    cleantext = re.sub('<.*?>', ' ', sentence)
    return cleantext

#s=cleanhtml("this is <\br> not my kind")
#print(s)

#clean punctuation
def cleanpunc(sentence): #function to clean the word of any punctuation or special characte
    cleaned = re.sub(r'[?|!|\|"|#|.|,|)|(|/]',r' ',sentence)
    #cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
s="this is <\br> not, my? (kind)!"
a=cleanhtml(s)
b=cleanpunc(a)
print(b)
```

```
this is   not  my   kind
```

#now we will do text preprocessing, omly "text" column is being used i this. Begin by removing the html tags Remove any punctuations or limited set of special characters like , or . or # etc. Check if the word is made up of english letters and is not alpha-numeric Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters) Convert the word to lowercase Remove Stopwords Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

In [20]:

```
#taking only first 100000 entries(system issue)
final_string=[]
s=" "
str1=" "
for sent in final["Text"].iloc[:100000] :    #["this,is <\br> beautiful","core? not <\h> ta
    filtered_sent=[]
    #clean html
    sent=cleanhtml(sent)
    #cleanpunctation
    sent=cleanpunc(sent)
    #check alpha num
    for word in sent.split():
        #print(word)
        if (word.isalpha()) & (len(word)>2):
            #print(word)
            if word.lower() not in stop:
                s=snow.stem(word.lower()).encode('utf8')
                print(s)
                filtered_sent.append(s)
            else: continue
        else: continue

        str1 = b" ".join(filtered_sent) #final string of cleaned words

    final_string.append(str1)
```

```
b'witti'
b'littl'
b'book'
b'make'
b'son'
b'laugh'
b'loud'
b'recit'
b'car'
b'drive'
b'along'
b'alway'
b'sing'
b'refrain'
b'learn'
b'whale'
b'india'
b'droop'
b'love'
```

In [21]:

```
len(final_string)
```

Out[21]:

```
100000
```

In [22]:

```
final_red= pd.DataFrame()
final_red=final[:100000]

final_red.head()
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [23]:

```
final_red["Cleaned Text"]= final_string
final_red.head(2)
```

```
c:\users\ashwani\appdata\local\programs\python\python36-32\lib\site-packag
es\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

Out[23]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
138706	150524	0006641040	ACITT7DI6IDDL	shari	0	0

In [24]:

```
#writing the cleaned table to sqlite
conn= sqlite3.connect('final.sqlite')
cur= conn.cursor()
conn.text_factory = str
final_red.to_sql('AMReviews', conn, flavor=None, schema=None, if_exists='replace', index=Tr
```

```
c:\users\ashwani\appdata\local\programs\python\python36-32\lib\site-packages
\pandas\core\generic.py:1534: UserWarning: The spaces in these column names
will not be changed. In pandas versions < 0.14, spaces were converted to und
erscores.
  chunksize=chunksize, dtype=dtype)
```

BAG OF WORDS

In [25]:

```
count_bow= CountVectorizer()
final_count= count_bow.fit_transform(final_red["Text"].values)
```

In [26]:

```
final_count.get_shape()
```

Out[26]:

```
(100000, 59978)
```

In [27]:

```
type(final_count)
```

Out[27]:

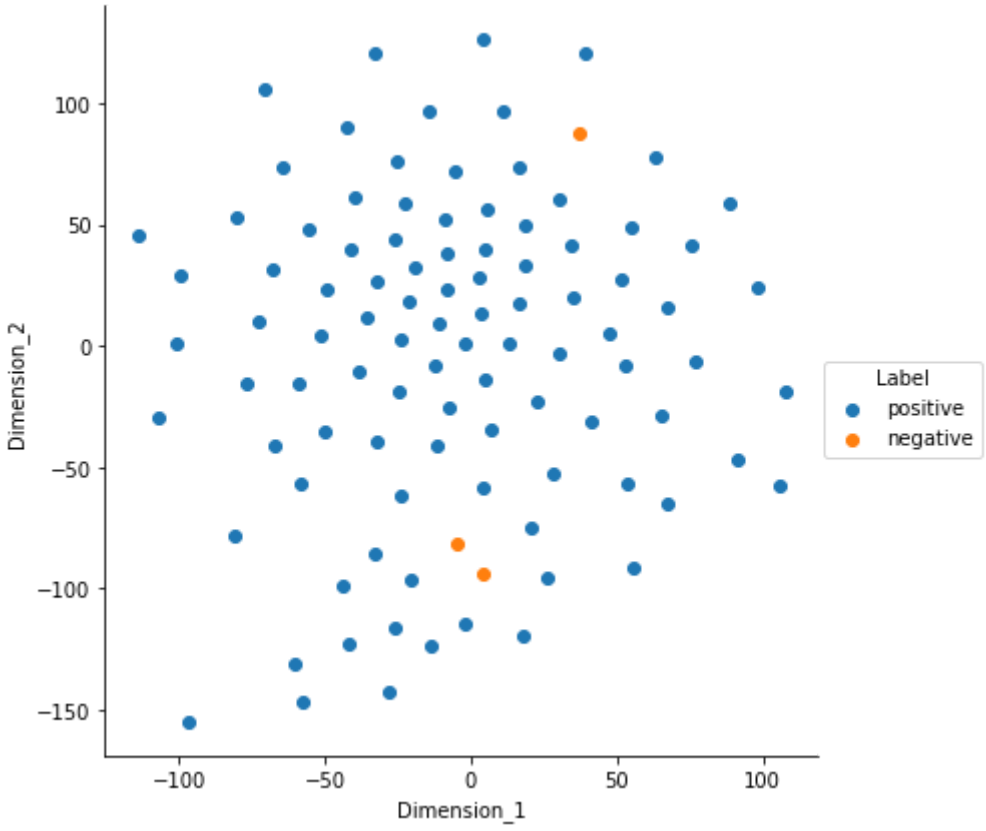
```
scipy.sparse.csr.csr_matrix
```

In [ ]:

```

```

In [28]:

```
#tSNE for bag of words
# TSNE

from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data= final_count[0:100,].toarray()
labels_1000 = final_red["Score"].iloc[0:100]

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

tsne_data = model.fit_transform(data)


# creating a new data frame which help us in ploting the result data
tsne_data = np.vstack((tsne_data.T, labels_1000)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dimension_1", "Dimension_2", "Label"))

# Ploting the result of tsne
sns.FacetGrid(tsne_df, hue="Label", size=6).map(plt.scatter, 'Dimension_1', 'Dimension_2').
plt.show()
```



In [ ]:

```

```