1 Import

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn. feature_extraction. text import TfidfTransformer
from sklearn. feature_extraction. text import TfidfVectorizer
from sklearn. feature_extraction. text import CountVectorizer
from sklearn. metrics import confusion_matrix
from sklearn import metrics
from sklearn. metrics import roc_curve, auc
from nltk. stem. porter import PorterStemmer
import re
import string
from nltk. corpus import stopwords
from nltk. stem import PorterStemmer
from nltk. stem. wordnet import WordNetLemmatizer
from gensim. models import Word2Vec
from gensim. models import KeyedVectors
import pickle
import scipy as sp
from tqdm import tqdm
import os
from sklearn. cross_validation import train_test_split
from sklearn. neighbors import KNeighborsClassifier
from sklearn. metrics import accuracy_score
from sklearn. cross_validation import cross_val_score
from collections import Counter
from sklearn. metrics import accuracy_score
from sklearn import cross_validation
from sklearn. metrics import confusion_matrix
from sklearn. preprocessing import normalize
from sklearn import datasets, neighbors
from sklearn. metrics import roc_auc_score
from sklearn. preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn import tree
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.decomposition import TruncatedSVD

with open("C:/Python/Assignments/Preprocessing/final_sorted.txt", "rb") as file:
    sorted_data = pickle.load(file)
```

```
C:\Anaconda3\lib\site-packages\gensim\utils.py:860: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

2 [5] Assignment 11: Truncated SVD

1. **Apply Truncated-SVD on only this feature set:**
   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
   - **Procedure:**
     - Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.
     - You need to calculate the co-occurrence matrix with the selected features (Note: X.X^T doesn't give the co-occurrence matrix, it returns the covariance matrix, check these bolgs [blog-1, (https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285)](https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285) [blog-2 (https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)](https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)for more information)
     - You should choose the n_components in truncated svd, with maximum explained variance. Please search on how to choose that and implement them. (hint: plot of cumulative explained variance ratio)
     - After you are done with the truncated svd, you can apply K-Means clustering and choose the best number of clusters based on elbow method.
     - Print out wordclouds for each cluster, similar to that in previous assignment.
     - You need to write a function that takes a word and returns the most similar words using cosine similarity between the vectors(vector: a row in the matrix after truncatedSVD)

2.1 Truncated-SVD

2.1.1 Load

In [2]:

```
# Taling 100k Points
my_final = sorted_data[:100000]
X = my_final['Text_new'].values
```

2.1.2 [5.1] Taking top features from TFIDF, SET 2

In [3]:

```
#Taking top 2000 features
tf_idf_vect = TfidfVectorizer( min_df=10, max_features=2000)
X_tfidf = tf_idf_vect.fit_transform(X)
```

In [4]:

```
# List of top 2000 words
top_words = np.array(tf_idf_vect.get_feature_names())
```

2.1.3 [5.2] Calulation of Co-occurrence matrix

In [5]:

```
sent = " ".join(X)
sent_list = sent.split(" ")
m, sent_length= len(top_words), len(sent_list)
dic = dict((top_words[i],i) for i in range(m))

```

In [7]:

```
import itertools
def create(neigh,m,sent_length,dic,sent_list):
    '''
    Parameters:
    neigh =  no of neighbours
    m = length of top_words
    sent_list = Corpus of all the words in data
    Sent_length = length of corus
    dic = Dictionary of each topwords and their location in topword list
    '''
    matrix = np.zeros((m,m)) # Initializing co occurence matrix

    tmp = sent_list[0:neigh+1]  # The first window of neigh + 1 elements.
    #print(tmp)                 # Since it is first we will check all possible combinations for matrix updation
    for word in tmp:       ## First update diag ele since it is same as the no of occurence of the word
        if(word in dic):
            loc = dic[word]  ## Get the location in topword list
            matrix[loc][loc] += 1
    for w1,w2 in itertools.combinations(tmp, 2): # This will give all possible combinations (nC2)
        if((w1 in dic) & (w2 in dic) & (w1 != w2)): # Bost must be top words and Both must be different
            l1,l2 = dic[w1],dic[w2]              # Then we get the location and
                                                #Udate the matrix
            matrix[l1][l2] += 1
            matrix[l2][l1] += 1

    #print("done 1st")
    for i in tqdm(range(1,len(sent_list) - neigh)):
        tmp = sent_list[i:i+neigh+1]   # This is the window of neigh + 1 elements
        #print(tmp)
        new_word = tmp[neigh]        # Only the last added word is the new one and we have to pair this with other elements
        if(new_word in dic):         # only if it is a top word otherwise NO
            loc = dic[new_word]    # Location of the new word in topword list
            matrix[loc][loc] += 1   # Because each diag ele is same as the no of occurence of the word


            for i in range(neigh):   # pairing new word with other elements in the window
                w1, w2 = tmp[i], new_word # getting the location of both words in the topwords list
                if((w1 in dic) & (w1 != new_word)): #if w1 is a topword and not same as new word, then we

                    l1,l2 = dic[w1],  loc      #get the location in top word list
                    matrix[l1][l2] += 1    # ANd dinally update the matrix element
                    matrix[l2][l1] += 1

    return matrix.astype(int)
co_matrix = create(neigh = 5, m = len(top_words), sent_length = len(sent_list),dic=dic, sent_list= sent_list)
```

```
100%|████████████████████████████| 3733264/3733264 [00:43<00:00, 86350.13it/s]
```

In [8]:

```
co_matrix
```

Out[8]:

```
array([[3020,    1,   11, ...,   13,    9,    3],
       [   1,  377,    0, ...,    3,    1,    1],
       [  11,    0, 2616, ...,    3,   29,   13],
       ...,
       [  13,    3,    3, ...,  958,   11,    3],
       [   9,    1,   29, ...,   11, 2024,    3],
       [   3,    1,   13, ...,    3,    3,  517]])
```

In [9]:

```
with open("mat.txt",'wb') as f:
    pickle.dump(co_matrix,f)
# Saving this as a back up in case kernel is restarted
```

2.1.4 [5.3] Finding optimal value for number of components (n) to be retained.

In [10]:

```
matrix = co_matrix
svd = TruncatedSVD(n_components=2000-1)
data = svd.fit_transform(matrix)
```

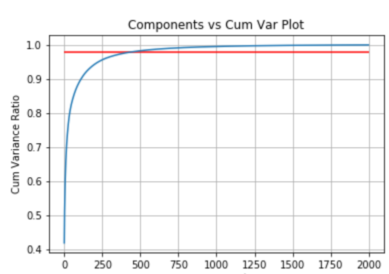
In [11]:

```
cum_var = np.cumsum(svd.explained_variance_ratio_)
```

In [12]:

```
plt.plot(cum_var)
plt.grid(1)
plt.xlabel("n_components")
plt.ylabel("Cum Variance Ratio")
plt.title("Components vs Cum Var Plot")
plt.hlines(0.98,0,2000,colors='r') #A line to show 99% coverage
plt.show
```

Out[12]: `<function matplotlib.pyplot.show(*args, **kw)>`



In [13]:

```
np.where(cum_var>0.98)[0][0:5]
```

Out[13]:

```
array([451, 452, 453, 454, 455], dtype=int64)
```

Observation: So with only 450 features I have 98% coverage of the whole data. So I will take 500 components instead of 2000

In [14]:

```
svd_new = TruncatedSVD(n_components=500)
data_new = svd_new.fit_transform(matrix)
```

2.1.5 [5.4] Applying k-means clustering

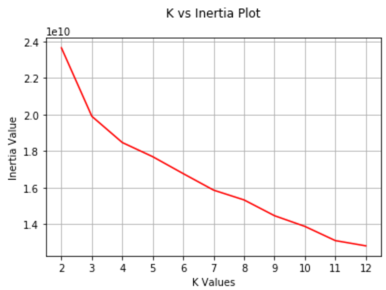
In [15]:

```
k_values = [2,3,4,5,6,7,8,9,10,11,12]
inertia = []
for k in (k_values):
    kmeans = KMeans(n_clusters=k).fit(data_new)
    inertia.append(kmeans.inertia_)

```

In [16]:

```python
plt.plot(k_values,inertia,'r')
plt.grid(True)
plt.title("K vs Inertia Plot\n")
plt.xlabel("K Values")
plt.ylabel("Inertia Value")
plt.xticks(k_values)
plt.show();
```



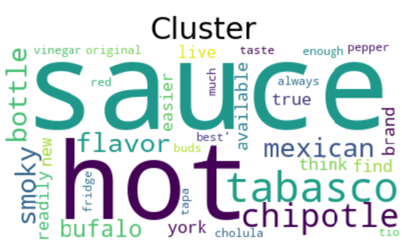
Observation: At Kvalue = 4, there is an elbow in the graph, So I will take 4 clusters for KMeans

2.1.6 [5.5] Wordclouds of clusters obtained in the above section

In [17]:

```python
opt_k = 4
kmeans = KMeans(n_clusters=opt_k).fit(data_new)
clust = [ [] for i in range(opt_k) ]  # this is the list of clusters
for i in range(kmeans.labels_.shape[0]):
    clust[kmeans.labels_[i]].append(X[i])

from wordcloud import WordCloud
# https://www.geeksforgeeks.org/generating-word-cloud-python/
def cloud(cluster):
    wordcloud = WordCloud(collocations=False, background_color ='white',
                    min_font_size = 10).generate(str(list(cluster)))
    plt.figure(figsize = (6,6), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.title("Cluster",size= 30,)
    plt.show()
for cl in clust:
    cloud(cl)
```



Observation:

1. Cluster 1 is about tea, flavor, chocolate, Product related items.
2. Cluster 2 is about sauce, tabasco, mexican foods etc.

3. Cluster 3 is about coffee, tea, flavours etc.

4. Cluster 4 is about beer, british food, tea, root etc.

2.1.7 [5.6] Function that returns most similar words for a given word.

In [18]:
```
from sklearn.metrics.pairwise import cosine_similarity
similar = cosine_similarity(data_new,data_new) # Cosine Similarity Matrix
similar.shape
```

Out[18]:
```
(2000, 2000)
```

In [19]:
```
def get_similar_word(word,n,):
    tmp = np.where(top_words == word)[0] #Since 2000 unique words, we will get one index only
    if(len(tmp)>0):
        index = tmp[0]
        df = pd.DataFrame()
        df['word'] = top_words
        df['vect'] = similar[index,:]
        df = df.sort_values(by = 'vect',kind = 'quicksort',ascending= False)
        return list(df.head(n)['word'])
    else:
        print("Word not present")
        return None

```

In [22]:
```
get_similar_word(top_words[6],5)
```

Out[22]:
```
['across', 'came', 'come', 'upon', 'running']
```

3 [6] Conclusions

1. I found that 450 components were covering 98% of the data, So I took 500 components instead of 2000.
2. After applying k-means, I found that 4 clusters were otimal for the data.
3. I think I will get better and more clear clusters if I take more data points.

4 Check the Co Occurence Matrix Function

Courpus: "abc def ijk pqr", "pqr klm opq", "lmn pqr xyz abc def pqr abc"

top_words: "abc", "pqr", "def"

window_size: 2

Co_occurace matrix

```
          abc    pqr    def

    abc    3.0    3.0    3.0

    pqr    3.0    4.0    2.0

    def    3.0    2.0    2.0
```

It has to be printed

In [23]:
```
X1 = ["abc def ijk pqr", "pqr klm opq", "lmn pqr xyz abc def pqr abc"]
top_words1 = ["abc", "pqr", "def"]
sent1 = " ".join(X1)
sent_list1 = sent1.split(" ")
m = len(top_words1)
dic1 = dict((top_words1[i],i) for i in range(m))
co_matrix = create(neigh = 2, m = len(top_words1), sent_length = len(sent_list1),dic=dic1, sent_list= sent_list1)
co_matrix
```

```
100%|██████████████████████████████████████████████████| 11/11 [00:00<?, ?it/s]
```

Out[23]:
```
array([[3, 3, 3],
       [3, 4, 2],
       [3, 2, 2]])
```

One More Example for Cross Checking

Now, let us take an example corpus to calculate a co-occurrence matrix.

Corpus = He is not lazy. He is intelligent. He is smart.

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

In [24]:

```
X1 = ['He is not lazy','He is intelligent','He is smart']
top_words1 = ['He','is','not','lazy','intelligent','smart']
sent1 = " ".join(X1)
sent_list1 = sent1.split(" ")
m = len(top_words1)
dic1 = dict((top_words1[i],i) for i in range(m))
co_matrix = create(neigh = 2, m = len(top_words1), sent_length = len(sent_list1),dic=dic1, sent_list= sent_list1)
co_matrix
```

```
100%|██████████████████████████████████████████████████| 7/7 [00:00<?, ?it/s]
```

Out[24]:

```
array([[3, 4, 2, 1, 2, 1],
       [4, 3, 1, 2, 2, 1],
       [2, 1, 1, 1, 0, 0],
       [1, 2, 1, 1, 0, 0],
       [2, 2, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 1]])
```