Loading Data

In [0]:
```
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
```

In [0]:
```
train_data=pd.read_pickle('CleanedText')
```

In [0]:
```
train_data.head()
```

Out[0]:
```
138706    witti littl book make son laugh loud recit car...
138683    rememb see show air televis year ago child sis...
417839    beetlejuic well written movi everyth excel act...
346055    twist rumplestiskin captur film star michael k...
417838    beetlejuic excel funni movi keaton hilari wack...
Name: CleanedText, dtype: object
```

TF-IDF

In [0]:
```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(train_data.head(60000))
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final
_tf_idf.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (60000, 29132)
the number of unique words including both unigrams and bigrams  29132
```

In [0]:
```
indices=(tf_idf_vect.idf_).argsort()
```

In [0]:
```
importent_features=np.take(tf_idf_vect.get_feature_names(),indices[:2000])
```

In [0]:
```
importent_features
```

Out[0]:
```
array(['tast', 'like', 'good', ..., 'cough', 'yuck', 'passion'],
      dtype='<U30')
```

In [0]:
```
text=train_data.head(60000).tolist()
```

In [0]:
```
importent_features=importent_features.tolist()
```

In [0]:
```
ctxs = text


mat = np.zeros((len(importent_features), len(importent_features)))

nei = []
nei_size = 3

for ctx in ctxs:
    words = ctx.split(' ')

    for i, _ in enumerate(words):
        nei.append(words[i])

        if len(nei) > (nei_size * 2) + 1:
            nei.pop(0)

        pos = int(len(nei) / 2)
        for j, _ in enumerate(nei):
          if nei[j]  in importent_features and words[i] in importent_features
:
            mat[importent_features.index(nei[j]), importent_features.index(wor
ds[i])] += 1

mat = pd.DataFrame(mat)
mat.index = importent_features
mat.columns = importent_features
display(mat)
```

	tast	like	good	great	love	flavor	one	product	tri	u
tast	27745.0	5527.0	3398.0	2737.0	1145.0	1540.0	1134.0	1100.0	1123.0	1203
like	2287.0	28071.0	1487.0	1078.0	1191.0	1982.0	1513.0	1194.0	1189.0	1081
good	1991.0	1684.0	21953.0	1049.0	1035.0	1430.0	969.0	1271.0	919.0	967
great	2290.0	1174.0	1036.0	20134.0	1250.0	1519.0	631.0	1942.0	708.0	1079
love	1471.0	1079.0	960.0	1285.0	18925.0	1345.0	963.0	1238.0	819.0	817
flavor	1867.0	2098.0	1388.0	1237.0	999.0	21956.0	1182.0	731.0	1085.0	953
one	1262.0	1355.0	991.0	828.0	874.0	969.0	19046.0	759.0	819.0	734
product	1476.0	1180.0	1293.0	1451.0	976.0	703.0	867.0	19354.0	864.0	1205
tri	1082.0	1310.0	843.0	744.0	986.0	1348.0	1388.0	929.0	16294.0	626
use	794.0	814.0	666.0	783.0	604.0	760.0	765.0	1151.0	507.0	18650
make	924.0	881.0	987.0	1480.0	484.0	524.0	661.0	643.0	427.0	799
get	728.0	705.0	728.0	683.0	562.0	540.0	749.0	569.0	422.0	717
buy	457.0	503.0	508.0	597.0	598.0	375.0	607.0	897.0	439.0	410
best	1475.0	507.0	419.0	458.0	427.0	655.0	345.0	412.0	737.0	540
would	659.0	1012.0	639.0	522.0	488.0	392.0	464.0	755.0	611.0	478
time	536.0	558.0	561.0	557.0	466.0	353.0	447.0	541.0	490.0	523
eat	534.0	739.0	623.0	499.0	555.0	301.0	816.0	446.0	426.0	296
realli	1079.0	1873.0	1544.0	645.0	653.0	704.0	400.0	489.0	409.0	373
find	542.0	526.0	617.0	504.0	376.0	441.0	470.0	615.0	284.0	423
also	436.0	785.0	708.0	730.0	463.0	425.0	376.0	343.0	471.0	810
dont	746.0	1492.0	418.0	306.0	291.0	399.0	375.0	416.0	352.0	584
amazon	315.0	354.0	538.0	771.0	466.0	239.0	372.0	739.0	368.0	342
much	684.0	737.0	380.0	344.0	393.0	570.0	421.0	418.0	354.0	462
even	563.0	695.0	398.0	360.0	440.0	325.0	397.0	299.0	396.0	399
littl	578.0	587.0	417.0	375.0	357.0	475.0	377.0	272.0	224.0	355
tea	1922.0	1493.0	1221.0	1046.0	978.0	1873.0	1122.0	464.0	1024.0	886
order	353.0	370.0	466.0	471.0	450.0	305.0	479.0	672.0	332.0	274
price	316.0	348.0	586.0	638.0	398.0	202.0	318.0	611.0	255.0	348
well	449.0	499.0	420.0	434.0	352.0	361.0	346.0	393.0	341.0	368
better	601.0	391.0	369.0	362.0	306.0	369.0	354.0	381.0	390.0	377
...	...	...	...	...	...	...	...	...	...	
stain	4.0	7.0	1.0	9.0	6.0	2.0	4.0	6.0	6.0	9
chunki	11.0	10.0	10.0	4.0	3.0	7.0	4.0	5.0	2.0	9
hemp	14.0	7.0	12.0	16.0	8.0	11.0	8.0	16.0	5.0	11
factori	7.0	6.0	3.0	2.0	2.0	5.0	2.0	5.0	6.0	4

	tast	like	good	great	love	flavor	one	product	tri	u
rope	3.0	9.0	6.0	11.0	9.0	4.0	11.0	3.0	3.0	3
starch	3.0	8.0	2.0	1.0	2.0	13.0	3.0	3.0	0.0	8
fault	6.0	4.0	4.0	4.0	2.0	3.0	4.0	23.0	3.0	5
mash	6.0	10.0	4.0	4.0	2.0	4.0	1.0	5.0	4.0	11
nestl	9.0	7.0	11.0	4.0	3.0	5.0	7.0	12.0	5.0	4
bug	4.0	9.0	4.0	1.0	3.0	1.0	10.0	3.0	4.0	5
assur	5.0	2.0	10.0	1.0	3.0	6.0	3.0	19.0	3.0	4
plump	4.0	7.0	12.0	5.0	2.0	14.0	1.0	3.0	1.0	4
truth	5.0	7.0	6.0	1.0	2.0	7.0	0.0	6.0	2.0	5
ensur	8.0	3.0	6.0	4.0	6.0	3.0	1.0	19.0	2.0	7
ketchup	21.0	17.0	11.0	8.0	8.0	11.0	11.0	7.0	7.0	8
access	6.0	4.0	7.0	4.0	7.0	4.0	10.0	7.0	3.0	6
swiss	7.0	9.0	12.0	3.0	1.0	1.0	6.0	6.0	4.0	7
graham	14.0	14.0	3.0	6.0	7.0	4.0	4.0	2.0	1.0	5
north	4.0	4.0	2.0	4.0	3.0	1.0	2.0	7.0	4.0	2
reserv	5.0	9.0	6.0	8.0	7.0	6.0	5.0	6.0	6.0	4
petit	1.0	8.0	3.0	4.0	7.0	9.0	3.0	8.0	8.0	2
master	9.0	9.0	2.0	8.0	5.0	10.0	9.0	1.0	8.0	4
lemonad	16.0	18.0	6.0	10.0	8.0	18.0	11.0	5.0	6.0	2
avid	2.0	5.0	2.0	5.0	8.0	3.0	4.0	2.0	8.0	3
concept	4.0	9.0	7.0	6.0	3.0	5.0	10.0	6.0	3.0	3
quarter	6.0	6.0	5.0	7.0	3.0	4.0	6.0	6.0	3.0	5
focus	2.0	2.0	3.0	1.0	1.0	8.0	8.0	9.0	5.0	7
cough	13.0	9.0	13.0	8.0	7.0	5.0	6.0	5.0	8.0	7
yuck	17.0	16.0	8.0	8.0	4.0	9.0	3.0	8.0	7.0	9
passion	10.0	11.0	11.0	5.0	2.0	24.0	4.0	1.0	4.0	2

2000 rows × 2000 columns

In [0]:
```
mat.shape
```

Out[0]:
```
(2000, 2000)
```

In [0]:
```
from sklearn.decomposition import TruncatedSVD
```

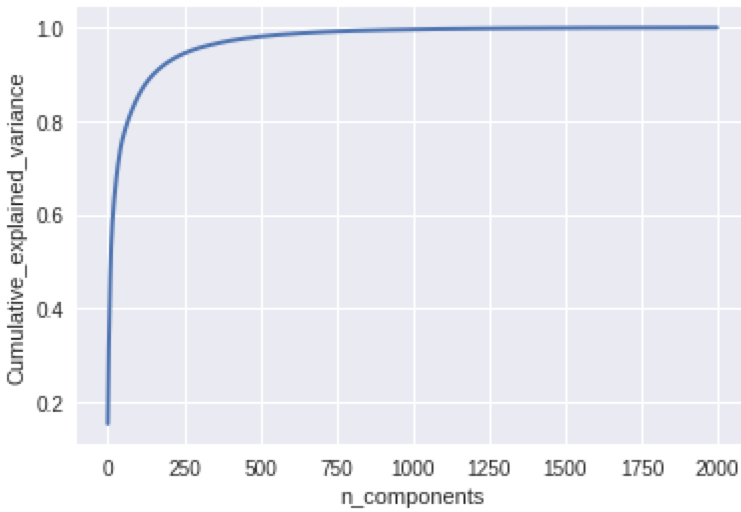
In [0]:
```
svd=TruncatedSVD(n_components=1999, algorithm='randomized', n_iter=5, random_s
tate=None, tol=0.0)
svd_data = svd.fit_transform(np.array(mat))
```

In [0]:
```
percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_vari
ance_);

cum_var_explained = np.cumsum(percentage_var_explained)


plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [0]:
```
svd=TruncatedSVD(n_components=250, algorithm='randomized', n_iter=5, random_st
ate=0, tol=0.0)
svd_data = svd.fit_transform(np.array(mat))
```

In [0]:
```
svd_data.shape
```

Out[0]: (2000, 250)

In [0]:
```
word_representation=pd.DataFrame(svd_data,index=importent_features)
```

In [0]:
```
word_representation.head()
```

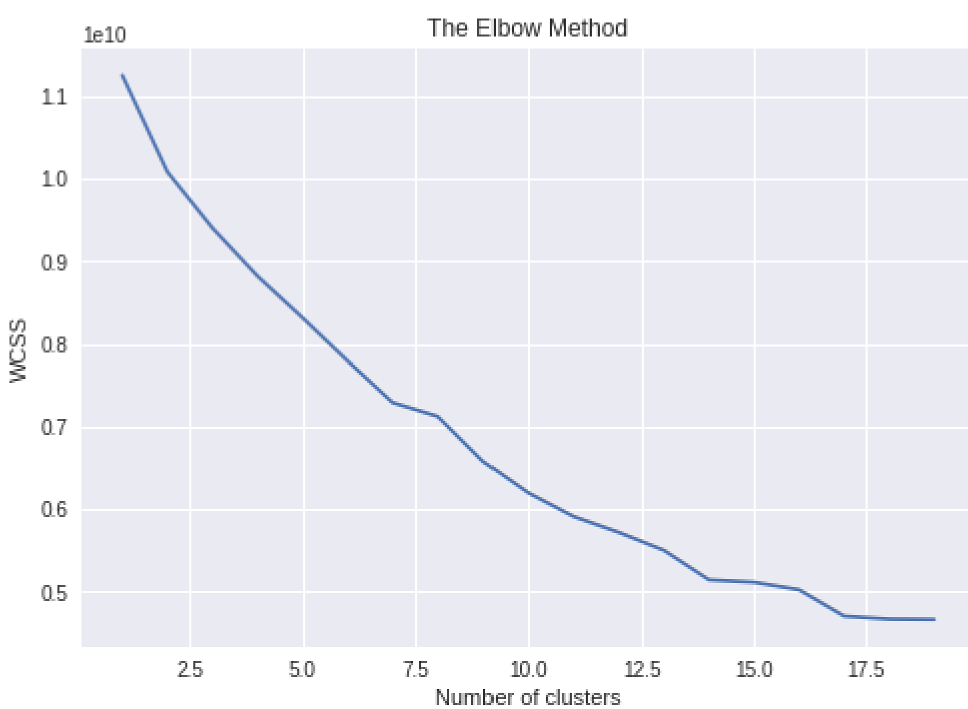
Out[0]:

	0	1	2	3	4	5	
tast	19867.851188	-9392.463216	-9185.686962	16733.948963	1333.374574	-1613.922750	2(
like	18176.675242	-11587.669187	19070.585302	-1226.902196	-1960.026390	141.428067	-2(
good	11117.953339	72.133948	-5681.124213	-3315.888183	-4671.581243	17627.393415	-32
great	9708.345237	663.042864	-5817.894817	-3243.333243	-886.272399	-7775.811348	-10(
love	8162.171842	1946.523810	-2351.628411	-4474.893086	-904.163180	-3123.765228	24

5 rows × 250 columns

In [0]:
```
from sklearn.cluster import KMeans
```

In [0]:
```
wcss=[]
for i in range(1, 20):
  kmeans = KMeans(n_clusters = i, init ='k-means++', random_state = 0)
  kmeans.fit(svd_data)
  wcss.append(kmeans.inertia_)
plt.plot(range(1, 20), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [0]:
```
kmeans = KMeans(n_clusters = 4, init ='k-means++', random_state = None)
kmeans.fit(svd_data)
```

Out[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [0]:
```
word_representation['cluster']=kmeans.labels_
```

In [0]:
```
word_representation['cluster'].value_counts()
```

Out[0]:
```
0    1987
3      11
2       1
1       1
Name: cluster, dtype: int64
```

In [0]:
```
from wordcloud import WordCloud, STOPWORDS
for j in [0,3,1]:
  DF = word_representation[word_representation['cluster']==j]
  comment_words=' '
  for val in DF.index:
    comment_words = comment_words + val + ' '
  print("Word cloud for cluster "+ str(j))
  wordcloud = WordCloud(width = 800, height = 800, background_color ='black',
min_font_size =5,max_words =200).generate(comment_words)
  plt.figure(figsize = (8, 8), facecolor = None)
  plt.imshow(wordcloud)
  plt.axis("off")
  plt.tight_layout(pad = 0)
  plt.show()
```

Word cloud for cluster 0

Word cloud for cluster 3

Word cloud for cluster 1

In [0]:
```python
def compute_similarity(Word):
  dataSetI=word_representation.loc[Word].values
  similarities=[]
  for val in word_representation.index:
    dataSet2=word_representation.loc[val].values
    result = spatial.distance.cosine(dataSetI, dataSet2)
    similarities.append(result)
  indices=np.array(similarities).argsort()
  index=indices[-10:]
  return np.take((word_representation.index).tolist(),index)
```

In [0]:
```python
compute_similarity("love")
```

Out[0]:
```
array(['contain', 'juic', 'sea', 'fat', 'listmania', 'virgin', 'tran',
       'hydrogen', 'fructos', 'satur'], dtype='<U11')
```

In [0]:
```
compute_similarity("price")
```

Out[0]:
```
array(['minut', 'water', 'fat', 'juic', 'chicken', 'skim', 'virgin',
       'fructos', 'satur', 'tran'], dtype='<U11')
```

Conclusion

No Of Importent Words Considered are 2000 for calculating co-occurence Matrix

Vectorizing Each word Using TruncatedSVD

250 Dimensions are considered for each word by plotting Explained Varience Ratio

No of Clusters Obtained by Using K-Means are 4

buy, best, new, food, order, realli, time are some importent words in cluster 0

flavor, good, product, tea, make, love, are some importent words in cluster 1

tast importent word in cluster 2