Project 2

Information Retrieval -Text Mining

Goal: Generate feature vectors for each document and store them in a file named feature definition file, with term frequency (TF), Inverse document frequency (IDF) and TF- IDF. With the generated feature definition file perform various text mining tasks such as feature extraction, feature selection, classification and clustering for different classifiers and different clustering evaluation metrics.

Technical details:

System requirement:

Python version: 3.7.1

Install NLTK - sudo pip install -U nltk

Install sklearn – pip install sklearn

Install matplotlib: pip install matplotlib

Note: Each training_data_file (training_data_file.TF, training_data_file.IDF, training_data_file.TFIDF) takes 30 to 40 minutes for creation.

Execution Commands:

1>Feature Extraction

training_data_file.TF:

python feature-extract.py mini_newsgroups feature_definition_file class_definition_file training_data_file 0

training_data_file.IDF:

python feature-extract.py mini_newsgroups feature_definition_file class_definition_file training_data_file 1

training_data_file. TFIDF:

>python feature-extract.py mini_newsgroups feature_definition_file class_definition_file training_data_file 2

2>Classification:

>python classification.py

3>Feature Selection:

>python feature_selection.py

4>Classification:

>python clustering.py

Implementation:

Part 1: Feature Extraction

Command: python feature-extract.py directory_of_newsgroups_data feature_definition_file class_definition_file training_data_file type_of_feature

```
C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>python feature-extract.py mini_newsgroups feature_definition
_file class_definition_file training_data_file 2
Creating training data file for terms is in progress
Feature id 1
Feature id 2
Feature id 3
Feature id 4
```

```
Feature id 32643
training_data_file.TFIDF is created

C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>
```

This is the Data preprocessing step where documents are converted to tokens along with eliminating the stop words, punctuations and performing stemming of words to avoid different lexicons of the same root word.

The command generates the training definition file for different feature types like term frequency, Inverse document frequency (IDF), TFIDF by passing parameter value 0,1,2 for each feature training data file in place of “type_of_feature”. Each document been processed at fileProcessing.py file and inverted index for features/terms been created along with feature id been assigned to the term. Which helps to create output file in libsvm format with each row corresponding to the document and each column name corresponds to feature id and values within these matrix form or libsvm file corresponds to feature vector (term frequency, IDF, TFIDF).

The above-mentioned command needs to be executed thrice with different values of type_of_feature to generate training_data_file.TF, training_data_file.IDF, training_data_file. TFIDF files.

All these files are stored in libsvm format for further processing of text mining.

< class label> <feature-id>:<feature-value> <feature-id>:<feature-value>.

Part 2: Classification:

Execution command: python classification.py

In this section, we experiment different algorithms of classification and output the F1 score, Recall and precision by macro approach where it computes performance of each class and then average it.

Experimentation is mainly done on four classifiers namely:

Multinomial Naive Bayes classifier, Bernoulli classifier, KNeighbors classifier, SVC classifiers which means Support Vector-machine Classifiers.

Once objects are created w.r.t to each classifier, we used cross-validation techniques to fit and train the data which even performs validation within given training set and output the required scores based on scoring parameter values we select.

Each classifier uses different feature values to function like multinomial used term frequency, Bernoulli with IDF, KNN classifier and SVM uses TF_IDF. Using appropriate feature vectors results in appropriate output.

Below are the outputs observed for the four-classifier related to F1 score, precision and recall.

Classifier Evaluation

```
C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>python classification.py
***********Multinomialclassifier***********
Multinomialclassifier
 F1_macro Accuracy: 0.71 (+/- 0.06)
 precision_macro Accuracy: 0.76 (+/- 0.12)
 recall_macro Accuracy: 0.71 (+/- 0.05)
***********Bernoulliclassifier***********
Bernoulliclassifier
 F1_macro Accuracy: 0.70 (+/- 0.09)
 precision_macro Accuracy: 0.71 (+/- 0.08)
 recall_macro Accuracy: 0.71 (+/- 0.10)
***********KNeighborsClassifier***********
KNeighborsClassifier
 F1_macro Accuracy: 0.21 (+/- 0.13)
 precision_macro Accuracy: 0.52 (+/- 0.21)
 recall_macro Accuracy: 0.27 (+/- 0.10)
***********SVCclassifer***********
SVCclassifer
 F1_macro Accuracy: 0.62 (+/- 0.07)
 precision_macro Accuracy: 0.67 (+/- 0.14)
 recall_macro Accuracy: 0.61 (+/- 0.08)

C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>
```

Discussion: Initially with default parameters, it might not get any reasonable measures. For example, SVM mainly deals with binary classifications default decision function will be one_over_rest (this is useful when we have only two classes to classify: it belongs to class or not). Since our data contains multi label classification we can use One_over_One (OVO) for better F1 score. This F1 score can be considered as evaluation measure based on our requirement, as it is the harmonic mean of precision and recall.

In most cases we see Precision as the king because we need algorithm which perform better accuracy. For a given Feature data Multinomial classifier has higher precision and recall. When we observe Knn classifier, it has accuracy just above 50% but of all the true classification. It predicted only 38% of existing true classification. Bernoulli and Multinomial classifier have high precision and recall which makes it show that for a given data, these algorithms are appropriate. But if we want to consider high precision and low recall then KNN can be considered. We can use any of the four if we adjusted default parameters based on our data nature.

Part 3: Feature Selection:

Execution command: python feature_selection.py

Here we perform or evaluate the classifiers by selecting best k feature vectors and observe the F1 scores of each classifier for two feature selection methods like “chi square” method and “Mutual information method”.

Chi-square: It determines the independence between two variables like feature vector and target label. IF the variables are dependent, we accept the hypothesis and if it is independent, we reject that hypothesis (those features as best). When we calculate chi squared value for all target labels and vectors, we can get best k features.

Mutual information method:

Mutual information method measures the mutual information between the feature and the target i.e. how much information we can get about the target when we know the feature or vice versa. This is mainly probability of one value when other is known.

Chi square gives information about dependence where mutual information means one feature gives information about another feature.

In this section we plotted graphs for chi-square and mutual information feature selection for different k values with F1 scores for four classifiers: the k values are

K values [100, 1000, 5000, 10000, 15000, 20000, 25000, 30000, 32643]

```
Markers  Properties  Servers  Snippets  Console  Progress
&lt;terminated&gt; feature_selection.py [/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7]
chi-square feature selection for different k values is in progress
K values [100, 1000, 5000, 10000, 15000, 20000, 25000, 30000, 32643]
 MultinomialNB [0.6369904475448018, 0.7675648405420609, 0.8206558543728374, 0.8090625569231775, 0.7811328959557085, 0.7538923827853609, 0.7325723576566392, 0.7151322555715597, 0.7087941317000406]
 BernoulliNB [0.7121867595941943, 0.8083722548360941, 0.8317042512590019, 0.7935623939552207, 0.7671791832719892, 0.7648105596396275, 0.7463442020273263, 0.7362862936139767, 0.6980042830429243]
 KNeighborsClassifier [0.5511261498088416, 0.4823021030395679, 0.29353317128732365, 0.2538585598641081, 0.1912010584452531 7, 0.18862596502733978, 0.19543613822370806, 0.19249153143686804, 0.191950604266542
 SVM [0.5455924884341948, 0.7114740029378698, 0.6639897629551175, 0.6357569683589192, 0.6318955436473301, 0.6251485855585359, 0.6224663366706571, 0.6228450594653243, 0.6228450594653243]
mutual Information feature selection for different k values is in progress
K values [100, 1000, 5000, 10000, 15000, 20000, 25000, 30000, 32643]
 MultinomialNB [0.6506897571802853, 0.7723832114492905, 0.7870533744764667, 0.7704521815727661, 0.765634803327082, 0.7375231665262589, 0.7269770784712924, 0.7156568041743884, 0.7087941317000406]
 BernoulliNB [0.7058319956569228, 0.7641265446140568, 0.7760935181133645, 0.777507169166245 7, 0.7433769922425071, 0.7325627198066478, 0.7199722659816286, 0.7153555373997966, 0.6980042830429243]
 KNeighborsClassifier [0.34385937328221844, 0.28377891570907643, 0.2414980978035291, 0.19318594594107527, 0.17570811695725064, 0.21502896698466795, 0.20540287496919868, 0.198754888117774, 0.191950604266542
 SVM [0.39767325064385894, 0.6772841308334628, 0.6883059054900598, 0.6706887687490377, 0.6404312744877938, 0.632556319183678, 0.6343934814187918, 0.6249729347640486, 0.6228450594653243]
```

<u>Discussion:</u>

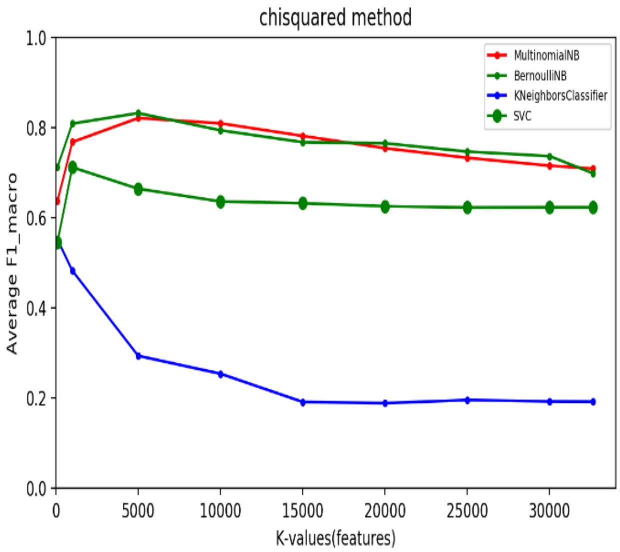
Chi square and mutual information feature selection almost behave similar at the same K value for classifiers except for KNN and SVM classifiers. Best 5000 features of chi square and MI gives a reasonable score for all the four classifiers.

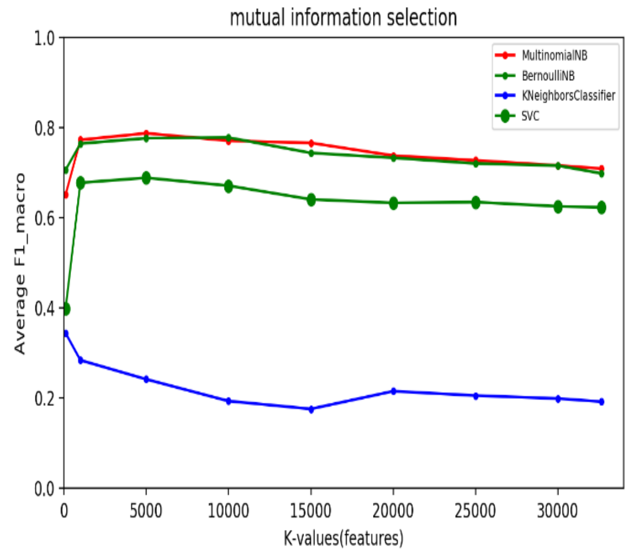
When the number of feature selection increases, the score tends to decrease or almost remain constant when all the features are considered.

F1 score values w.r.t to both feature selection gives better outcomes for the average number of features (k value).

For a given sample size, data dimensionality reduction to 5000 features yields an average F1 score in the range of 0.6 to 0.7 for all the classifiers except for KNN, it has a high score which is in the range of 0.3 and 0.5 for MI and Chi-squares at low number of features i.e. at k=100

Below graph gives a visual representation of the above observation

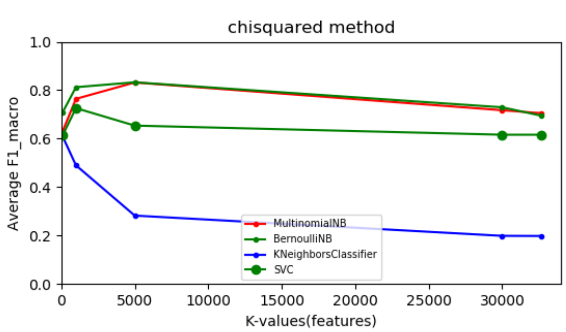


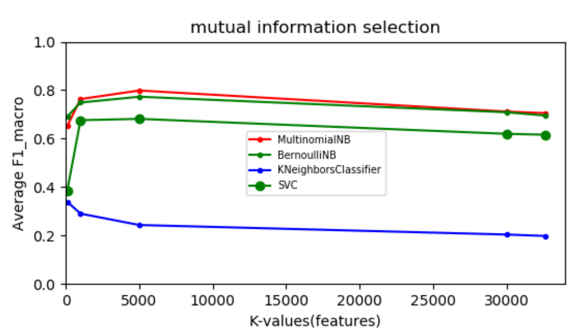


Example 2:

K values [100, 1000, 5000,30000, 32643]

```
C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>python feature_selection.py
chi-square feature selection for different k values is in progress
K values [100, 1000, 5000, 30000, 32643]
 MultinomialNB [0.6316839414483593, 0.7644151139064873, 0.8319391621144728, 0.7176138879508985, 0.7053230066835346]
 BernoulliNB [0.7096554762160058, 0.812536247550397, 0.8327668746631602, 0.7296158103679755, 0.6953278225745639]
 KNeighborsClassifier [0.6084463595717438, 0.4903547367947345, 0.28286737127219214, 0.19930066667194618, 0.1987767319126895]
 SVM [0.6145797937702374, 0.7261182362264679, 0.6540786226887152, 0.6165128039666724, 0.6165128039666724]
mutual Information feature selection for different k values is in progress
K values [100, 1000, 5000, 30000, 32643]
 MultinomialNB [0.65212643032246, 0.7635195482977025, 0.7987179388380885, 0.7120765738183732, 0.7053230066835346]
 BernoulliNB [0.6909810487618315, 0.7493187120453925, 0.773138582049474, 0.7091829424610903, 0.6953278225745639]
 KNeighborsClassifier [0.3397196566042817, 0.2908893417784367, 0.24391707763683496, 0.20472154060074366, 0.1987767319126895]
 SVM [0.386386989794549, 0.6762487492868041, 0.681839890429577, 0.6201574706827098, 0.6165128039666724]
```

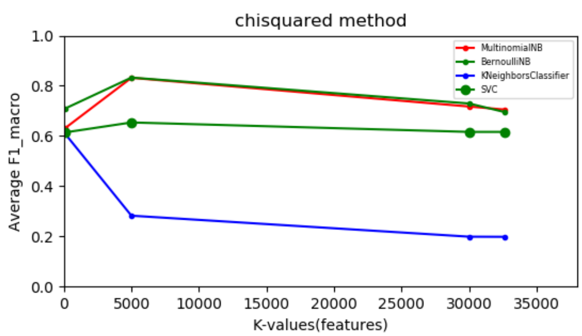


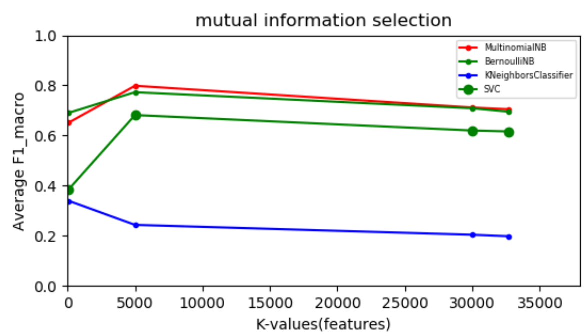


Example 3:

K values [100,5000,30000, 32643]

```
C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>python feature_selection.py
chi-square feature selection for different k values is in progress
K values [100, 5000, 30000, 32643]
 MultinomialNB [0.6316839414483593, 0.8319391621144728, 0.7176138879508985, 0.7053230066835346]
 BernoulliNB [0.7096554762160058, 0.8327668746631602, 0.7296158103679755, 0.6953278225745639]
 KNeighborsClassifier [0.6084463595717438, 0.28286737127219214, 0.19930066667194618, 0.1987767319126895]
 SVM [0.6145797937702374, 0.6540786226887152, 0.6165128039666724, 0.6165128039666724]
mutual Information feature selection for different k values is in progress
K values [100, 5000, 30000, 32643]
 MultinomialNB [0.65212643032246, 0.7987179388380885, 0.7120765738183732, 0.7053230066835346]
 BernoulliNB [0.6909810487618315, 0.773138582049474, 0.7091829424610903, 0.6953278225745639]
 KNeighborsClassifier [0.3397196566042817, 0.24391707763683496, 0.20472154060074366, 0.1987767319126895]
 SVM [0.386386989794549, 0.681839890429577, 0.6201574706827098, 0.6165128039666724]
```





Part 4: Document Clustering:

Execution command: python clustering.py

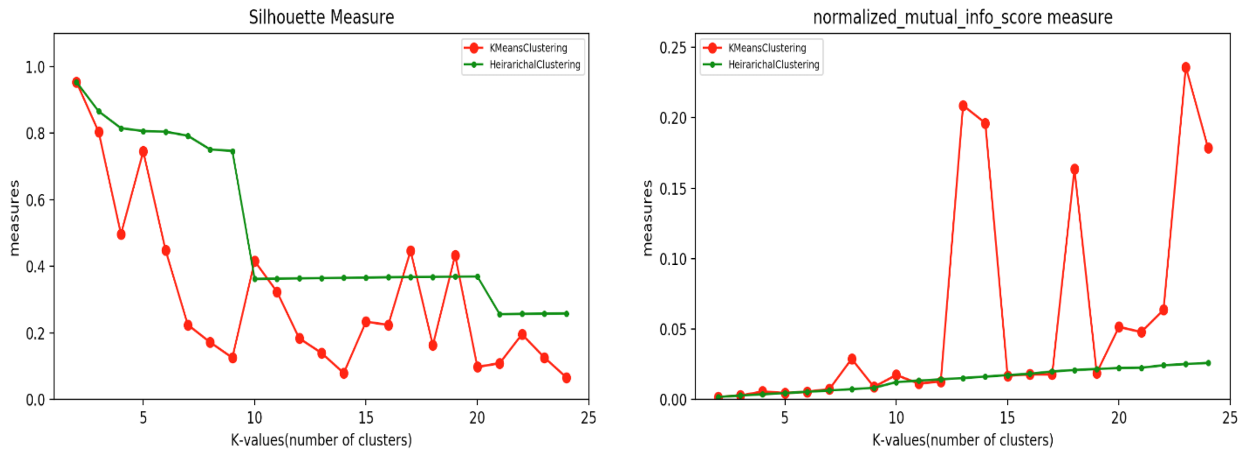
Here we cluster the documents by selecting k best features using chi-square method for faster clustering. Clustering is performed by taking TF-IDF feature values for feature ids into account.

<u>The clustering performances are evaluated with two metrics: the Silhouette Coefficient (SC) and Normalized Mutual Information (NMI).</u>

The graph has been plotted for a different number of clusters for K-*Means Clustering* and hierarchical clustering against metrics (measures) and the results are analyzed below:

<u>Discussion:</u> Taking k=5000 features using chi-square method graphs are plotted against different cluster k values ranging from [2, 25] against quality evaluation metrics scores.

```
Markers  Properties  Servers  Snippets  Console  Progress
clustering.py [/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7]
Cluster range [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Silhouette scores for K-means clustering [0.9695122528996388, 0.9122065627644556, 0.836161229310193, 0.7369480995608988, 0.584992246814821, 0.39894401156124865, 0.8194747205619326, 0.746855383432067, 0.69
Silhouette scores for AgglomerativeClustering(hierarchical clustering) [0.9695122528996388, 0.9122065627644556, 0.8794249189132644, 0.872564519405465, 0.8646385829483345, 0.8625872842458745, 0.837987146865
normalized_mutual_info_score for K-means clustering [0.00177826014994488, 0.0027234746164640344, 0.003665068309456007, 0.006126021556875656, 0.006356391375246822, 0.010018491495220752, 0.007298047379311558
normalized_mutual_info_score for AgglomerativeClustering(hierarchical clustering) [0.00177826014994488, 0.0027234746164640344, 0.003665068309456007, 0.004603072901156853, 0.005557382105735637, 0.0063520167
```



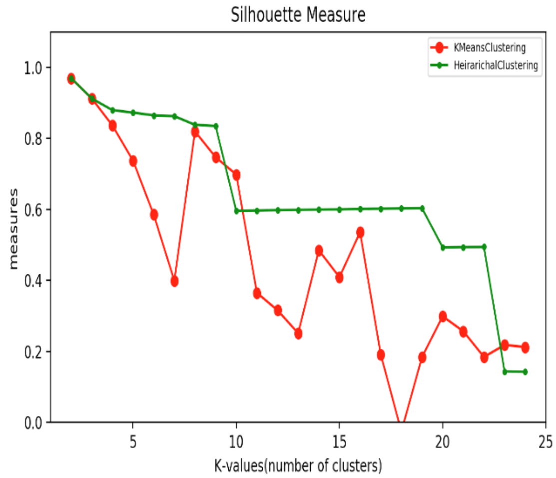
For SC quality measure, the score tends to "decrease as the number of clusters increases", for both clustering methods. However, there is much irregularity with K means clustering whereas hierarchical clustering reaming constant for few numbers of value of clusters.

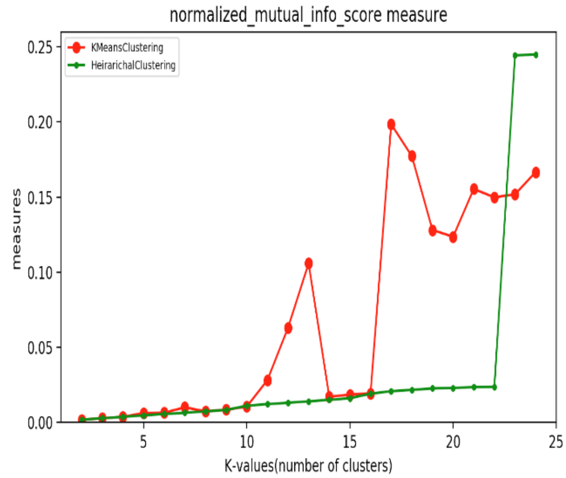
But with NMI measure, the trend seems quite the opposite. The scores tend to "increase for both the clustering methods". As mentioned earlier the irregularity exist with k mean clustering and bottom-up clustering score tends to increase with number of clusters.

Both clusters behave similarly w.r.t to both the metrics because in clustering, we use bottom-up approach parameter linkage = 'ward' hence it behaves similar to k means clustering.

Increasing or decreasing pattern w.r.t to two metrics is due to the parameter of class labels provided in each of the metrics method. In SC, no external class labels are provided, hence the Silhouette Coefficient is generally higher for convex clusters such as density. When number of clusters is less, the cluster tends to have high density. So, the score is nearly 1 for both the clustering methods. Increase in the trend of NMI is due to the number of clusters. When the number of clusters is less, the mutual dependence is less because the given data sample has 5 classes, so the NMI score i.e. mutual dependence increases with increase in number of clusters.

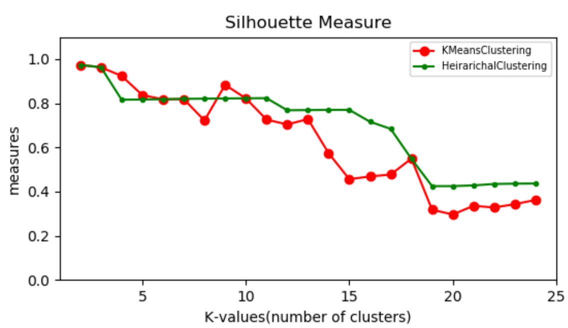
For K=1000,

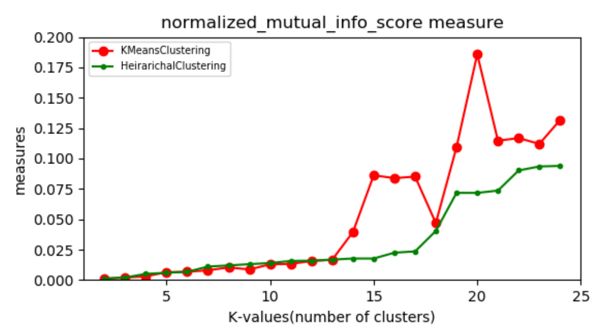




K=100

```
C:\Users\vvais\OneDrive\Desktop\IR_TEXTMINING\IR_TextMining>python clustering.py
Clustering and Quality evaluation is in Progress for feature count k=100 for clusters count in range[2,25]
Cluster range [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Silhouette scores for K-means clustering [0.9733794615518677, 0.9626561697366032, 0.8158880464077215, 0.8956475129971877, 0.8624439421897974, 0.88212556753791, 0.882990
6847054466, 0.883418618679369, 0.4659584878519319, 0.638078517955743, 0.8368483024527054, 0.729401991547623, 0.7070799259387899, 0.45269964094803095, 0.5079863183823428
, 0.5260165557548875, 0.4328874725692552, 0.3201530747949614, 0.45890685999116354, 0.46818331043332906, 0.3311964266766884, 0.3348202828000002, 0.36392003170565373]
Silhouette scores for AgglomerativeClustering(hierarchical clustering) [0.9733794615518677, 0.9626561697366032, 0.8158880464077215, 0.8170473386576013, 0.81812517483843
72, 0.819841799403431, 0.8207628889429887, 0.8216037473850886, 0.8223214936789993, 0.8230550160345806, 0.7682816234893762, 0.769148394917881, 0.7699473233305913, 0.7700
090298824241, 0.7163419668389261, 0.6839307714723927, 0.5487039653580114, 0.4247897723363779, 0.4250465606618293, 0.4282551809948454, 0.4347043885902225, 0.436143249985
45373, 0.4367875275944386]
normalized_mutual_info_score for K-means clustering [0.0011238034461919836, 0.0020723303363166146, 0.005045966385670541, 0.00482437964252138, 0.007703593976847201, 0.00
7963789627666529, 0.007931631862706996, 0.009561888194284293, 0.07058505468175609, 0.0429061642935239, 0.013717893874391453, 0.018003150546969098, 0.018174874497010374,
 0.07800706365186706, 0.061432668031586764, 0.041790453494995786, 0.10895856630117744, 0.10357523258468396, 0.10752669284903674, 0.10821576720376193, 0.1143260900899585
7, 0.10524996194102176, 0.13291228107807496]
normalized_mutual_info_score for AgglomerativeClustering(hierarchical clustering) [0.0011238034461919836, 0.0020723303363166146, 0.005045966385670541, 0.005968341945428
891, 0.006763376276932367, 0.011082618062801858, 0.01199981875478137, 0.01307038047922065, 0.01397905294296894, 0.01571498299469413, 0.015914239801000186, 0.01681162600
0169895, 0.017705830847926634, 0.017694434755142555, 0.022480425898187033, 0.02356123251053695, 0.04039436184801191, 0.07171864553177261, 0.0716841237017485, 0.07360512
510172817, 0.09017401329548577, 0.09343656194796257, 0.09389611230529939]
```





Test Cases:

1.Feature-Selection verification test case:

Feature selection is also known as attribute selection is a process of *extracting the most relevant features* from the dataset and then applying machine learning algorithms for the better performance of the model.

Manual verification of the feature selection using chi-square and mutual informal method are appropriate.

Chi square test is used to test the independence of two events, which for us are: occurrence of a term and occurrence of a class.

o = Observed Frequency

e = Expected frequency

$$\chi^2 = \sum \frac{(o-e)^2}{e}$$

Calculating Chi-square for sample libsvm term frequency feature.

Feature vector data of training_data_file.TF data for two classes:

0 684:2 897:1 1306:1 1621:2 2223:1

1 171:1 412:3 684:2 897:1 1306:1 1621:2 2223:1

training_data_file.txt — Edited

```
0 684:2 897:1 1306:1 1621:2 2223:1
1 171:1 412:3 684:2 897:1 1306:1 1621:2 2223:1
```

Using chi-square 4 best features extracted from training_data_file.TF are:

Test code:

```
def Ftest():
    X_Chi = SelectKBest(chi2, k=4).fit_transform(train_feature_tf, train_target_tf)
    X_Mut = SelectKBest(mutual_info_classif, k=10).fit_transform(train_feature_tf, train_target_tf)
    print(X_Chi)
    print("*******")
    print(X_Mut)
```

Chi-square statistical calculation:

Sample data of Term frequency feature:

Libsvm file - labels → 0   684:2   897:1   1306:1   1621:2   2223:1

1   171:1   412:3   684:2   897:1   1306:1   1621:2   2223:1

feature id   feature values:

feature id Position	feat Val/id	Labels 0	1	Total
0	171	0	1	1 — M
1	412	0	3	3
2	684	2	2	4
3	897	1	1	2
4	1306	1	1	2
5	1621	2	2	4
6	2223	1	1	2
Total →		7 P	11 Q	18 (N)

$$\chi^2 = \sum_{i=1}^{n} \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}}$$

No. of classes

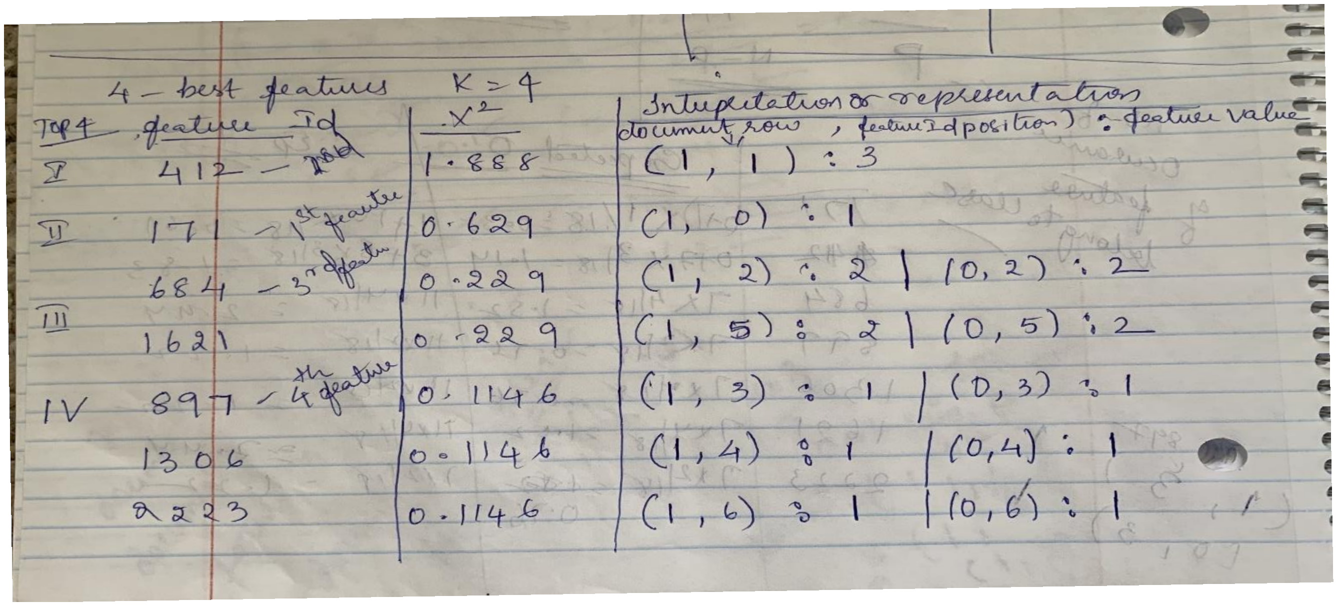
Expected value $= \left(\frac{P/Q \times M}{N}\right)$.

Calculating Expected Value for each feature

	Class : 0	Class : 1
171	$7 \times 1/18 = 0.38$	$11 \times 1/18 = 0.61$
412	$7 \times 3/18 = 1.14$	$11 \times 3/18 = 1.83$
684	$7 \times 4/18 = 1.52$	$11 \times 4/18 = 2.44$
897	$7 \times 2/18 = 0.76$	$11 \times 2/18 = 1.22$
1306	$7 \times 2/18 = 0.76$	$11 \times 2/18 = 1.22$
1621	$7 \times 4/18 = 1.52$	$11 \times 4/18 = 2.44$
2223	$7 \times 2/18 = 0.76$	$11 \times 2/18 = 1.22$
	$\simeq 7$	$\simeq 11$

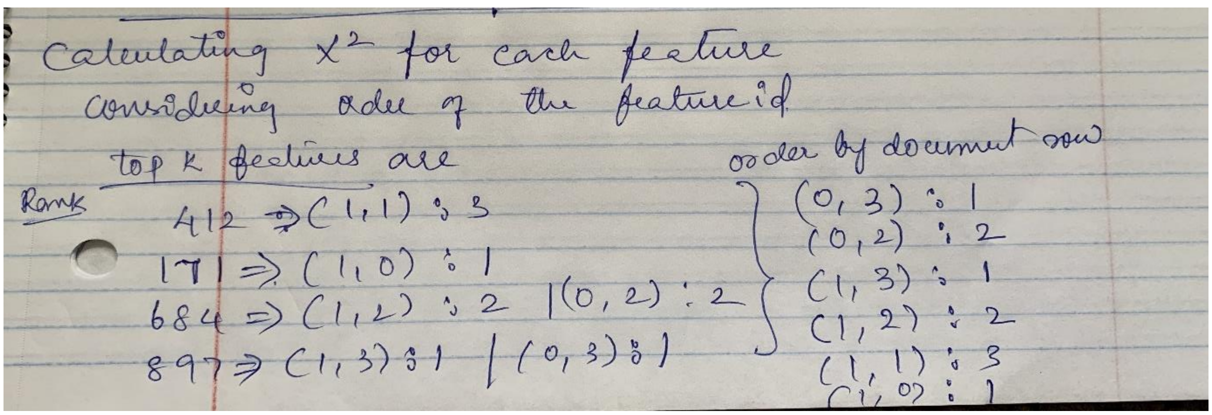
Calculating χ^2 for each feature

	classlabel : 0	classlabel : 1	χ^2 Sum	Rank
171	$(0-0.38)^2/0.38 = 0.38$	$(1-0.61)^2/0.61 = 0.249$	$0.38 + 0.249 = 0.629$	II
412	$(0-1.14)^2/1.14 = 1.14$	$(3-1.83)^2/1.83 = 0.748$	1.888	I
684	$(2-1.52)^2/1.52 = 0.15$	$(2-2.44)^2/2.44 = 0.079$	0.229	III
897	$(1-0.76)^2/0.76 = 0.075$	$(1-1.22)^2/1.22 = 0.0396$	0.1146	IV
1306	$(1-0.76)^2/0.76 = 0.075$	$(1-1.22)^2/1.22 = 0.0396$	0.1146	IV
1621	$(2-1.52)^2/1.52 = 0.15$	$(2-2.44)^2/2.44 = 0.079$	0.229	III
2223	$(1-0.76)^2/0.76 = 0.075$	$(1-1.22)^2/1.22 = 0.0396$	0.1146	IV


4 – best features K = 4

TOP 4	feature Id	X^2	Interpretation or representation (document row, feature Id position) : feature value
I	412 – 2nd	1.888	(1, 1) : 3
II	171 – 1st feature	0.629	(1, 0) : 1
	684 – 3rd feature	0.229	(1, 2) : 2 | (0, 2) : 2
III	1621	0.229	(1, 5) : 2 | (0, 5) : 2
IV	897 – 4th feature	0.1146	(1, 3) : 1 | (0, 3) : 1
	1306	0.1146	(1, 4) : 1 | (0, 4) : 1
	2223	0.1146	(1, 6) : 1 | (0, 6) : 1

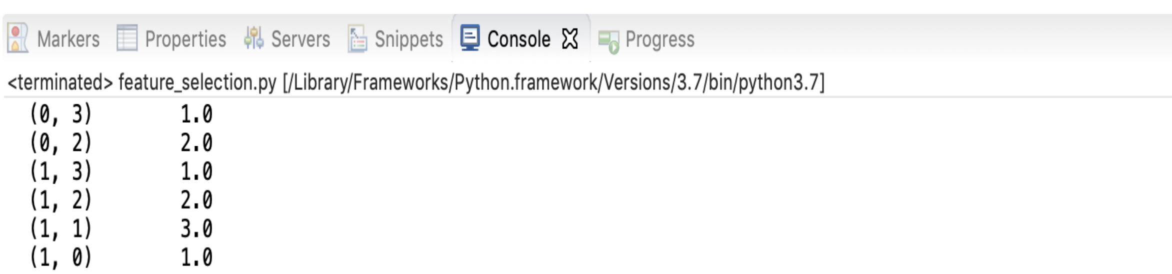
Expected:


Calculating X^2 for each feature
considering order of the feature id
top k features are

Rank
412 ⇒ (1, 1) : 3
171 ⇒ (1, 0) : 1
684 ⇒ (1, 2) : 2 | (0, 2) : 2
897 ⇒ (1, 3) : 1 | (0, 3) : 1

order by document row
(0, 3) : 1
(0, 2) : 2
(1, 3) : 1
(1, 2) : 2
(1, 1) : 3
(1, 0) : 1

Actual:


Markers Properties Servers Snippets Console Progress

```
<terminated> feature_selection.py [/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7]
  (0, 3)        1.0
  (0, 2)        2.0
  (1, 3)        1.0
  (1, 2)        2.0
  (1, 1)        3.0
  (1, 0)        1.0
```

2.Classification verification test case:

This test case validates the predicted and actual value by dividing the data into training set and test. Using fit method to train with training set and once the model is trained, prediction is made on test data set and compared with test data target set. Furthermore, mean error also been calculated. This validation covers this testing experiment on four classifiers named multinomial, Bernoulli, KNN and SVM.

Test Code:

```
def test():
    '''Evaluating the system on test set and see mean square error cost function'''
    '''splitting the training data and test data 80 :20'''
    trainsize_tf=int(len(targets_tf)*0.8)
    train_feature_tf=feature_vectors_tf[:trainsize_tf]
    train_target_tf=targets_tf[:trainsize_tf]
    test_feature_tf=feature_vectors_tf[trainsize_tf:]
    test_target_tf=targets_tf[trainsize_tf:]
    'IDF training data file'
    training_data_file_IDF = "training_data_file.IDF"
    feature_vectors_IDF, targets_IDF = load_svmlight_file(training_data_file_IDF)
    trainsize_IDF=int(len(targets_IDF)*0.8)
    train_feature_IDF=feature_vectors_IDF[:trainsize_IDF]
    train_target_IDF=targets_IDF[:trainsize_IDF]
    test_feature_IDF=feature_vectors_IDF[trainsize_IDF:]
    test_target_IDF=targets_IDF[trainsize_IDF:]

    'TF-IDF training data file'
    training_data_file_TFIDF = "training_data_file.TFIDF"
    feature_vectors_TFIDF, targets_TFIDF = load_svmlight_file(training_data_file_TFIDF)
    trainsize_TFIDF=int(len(targets_TFIDF)*0.8)
    train_feature_TFIDF=feature_vectors_TFIDF[:trainsize_TFIDF]
    train_target_TFIDF=targets_TFIDF[:trainsize_TFIDF]
    test_feature_TFIDF=feature_vectors_TFIDF[trainsize_TFIDF:]
    test_target_TFIDF=targets_TFIDF[trainsize_TFIDF:]
    #print(test_feature_TFIDF.shape,test_target_TFIDF.shape)
    for name,classifer in classifers.items():
        clf=classifer
        warnings.filterwarnings("ignore")
        if name=='Multinomialclassifier':
            print(name+"Prediction with TF features ")
            train_feature=train_feature_tf
            train_target=train_target_tf
            test_feature=test_feature_tf
            test_target=test_target_tf
        if name=='Bernoulliclassifier':
            print(name+"Prediction with IDF features ")
            train_feature=train_feature_IDF
            train_target=train_target_IDF
            test_feature=test_feature_IDF
            test_target=test_target_IDF
        if (name=='KNeighborsClassifier'):
            print(name+"Prediction with TFIDF features ")
            train_feature=train_feature_TFIDF
            train_target=train_target_TFIDF
            test_feature=test_feature_TFIDF
            test_target=test_target_TFIDF
            #print(test_feature.shape,test_target.shape)
        if name=='SVCclassifer':
            print(name+"Prediction with TFIDF features ")
            train_feature=train_feature_TFIDF
            train_target=train_target_TFIDF
            test_feature=test_feature_TFIDF
            test_target=test_target_TFIDF

        clf.fit(train_feature,train_target)
        Y_predit=clf.predict(test_feature[315])
        test_final_Prediction=clf.predict(test_feature)
        'calculating square root of mean squared error training on train data test on test set'
        test_final_rmse=np.sqrt(mean_squared_error(test_target, test_final_Prediction))
        print("Test data Predicted value",Y_predit,"Expected value",test_target[315])
        Y_predit=clf.predict(train_feature[0])
        train_final_Prediction=clf.predict(train_feature)
        'calculating square root of mean squared error training and evalutation on training data'
        train_final_rmse=np.sqrt(mean_squared_error(train_target, train_final_Prediction))
        print("Training Data Predicted value",Y_predit,"Expected value",train_target[0])
        print("RMSE on training and test data",train_final_rmse,test_final_rmse)
```

Result:

```
<terminated> classification.py [/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7]
MultinomialclassifierPrediction with TF features
        Test data Predicted value [1.]  Expected value 1.0
        Training Data Predicted value [0.]      Expected value 0.0
        RMSE on training and test data 0.4819187497721559 1.8283082337074499
BernoulliclassifierPrediction with IDF features
        Test data Predicted value [2.]  Expected value 1.0
        Training Data Predicted value [0.]      Expected value 0.0
        RMSE on training and test data 0.3138924511505739 1.4187275015961192
KNeighborsClassifierPrediction with TFIDF features
        Test data Predicted value [5.]  Expected value 1.0
        Training Data Predicted value [0.]      Expected value 0.0
        RMSE on training and test data 0.025294174537134735 3.4262414443209646
SVCclassiferPrediction with TFIDF features
        Test data Predicted value [2.]  Expected value 1.0
        Training Data Predicted value [0.]      Expected value 0.0
        RMSE on training and test data 0.3189474050950887 1.3206988772224633
```

Observation:

For few cases expected did not match with actual on the test set. but it worked fine on training set. It shows that overfitting of data been occurred. Tuning the parameter values can result in better predictions and less error on test data.