April 21, 2019

Manas Gaur, Amanuel Alambo
Information Retrieval, CS 7800
Project-2 on Text Mining

This project is used to index the 20 newsgroup datasets and test classification and clustering algorithms. There is four activities conducted:

- Feature extraction
- Classification (using four algorithms and the evaluation of each classifier)
- Feature selection
- Document clustering

We walk through how each is implemented and experimental results. The codes in this project are developed and tested using python3
The computational cost of Support Vector Machine (SVM) was very high compared to other machine learning classifiers even on multithreaded execution of the experiments.

1. Feature extraction

How to Run:
python3 feature-extract.py directory_of_newsgroups_data feature_definition_file class_definition_file training_data_file
Example:
python3 feature-extract.py mini_newsgroups *feature_definition_file class_definition_file training_data_file.TFIDF*
Note: the *training_data_file* is required to have an extension of either **'.TF' or '.IDF' or '.TFIDF'**. The program will throw an error if this format is not followed.

Disclaimer: the three training data files(i,e, TF, IDF, and TF-IDF) are generated in three different executions of the above program. Thus, their entries are different. We tested our classification and clustering algorithms using the **'.TFIDF'** training dataset.

The feature extraction part of this task is implemented in code ***'feature-extract.py'***. An Index class is defined that is used to index each of the files in each of the newsgroups. We index the 'subject' and 'body of each document. Thus, we needed to parse the subject and body from the document. This parsing is done in method ***'subject_body_index'*** of code ***'feature-extract.py'***. Since the subject of the doc is identified by a **'subject'** in the doc, we used the **'subject'** entry for parsing. However, we don't have an explicit identifier for the body for each document. Yet, the ***'Lines'*** value in the doc refers to the body of the doc, which is the last ***'Lines'*** value. For this, we used the ***'deque'*** construct to grab the last ***'Lines'*** number of lines and write to a variable. Once the subject and body are extracted from the document, we performed string concatenation and assigned to a variable. Finally, we indexed the concatenated **'subject+body'** into our index structure.
We defined two methods ***'indexed_docs'*** and ***'indexed_docs_I'*** in class ***'Index'***. The ***'indexed_docs'*** method indexes the whole newsgroup datasets and stores in an inverted index (each stemmed term with stopwords removal and punctuation marks removal applied and list of the doc-ids as values). The ***'indexed_docs_I'*** method is used to return the *doc-id* to *document content* mappings. Further, we implement ***'feature_defn_gen'*** and ***'class_defn_gen'*** methods for generating **'feature definition'** and 'class definition files'. For generating the feature definition file, the method accepts the feature definition file output name while for generating the class definition file, the corresponding method accepts the output file name for

class definition file and the path to the root directory(i.e., mini_newsgroups). The term frequency, inverse document frequency, or TF-IDF scores are rounded to **'2'** decimal digits as can be seen in the screenshot below. Three helper function are defined to compute TF-IDF score of a term in a document (functions ***'filecount()', 'nDocs_in_subdir', 'idf_compute'***). Function ***'filecount()'*** is used to count the number of documents(files) in each subdirectory, while function ***'nDocs_in_subdir()'*** accepts a document-id and calls function ***'filecount()'*** to get the total number of documents in a subdirectory the corresponding document belongs to.

Sample snippet of the generated TF-IDF based training data looks like (this is in *libsvm* format):

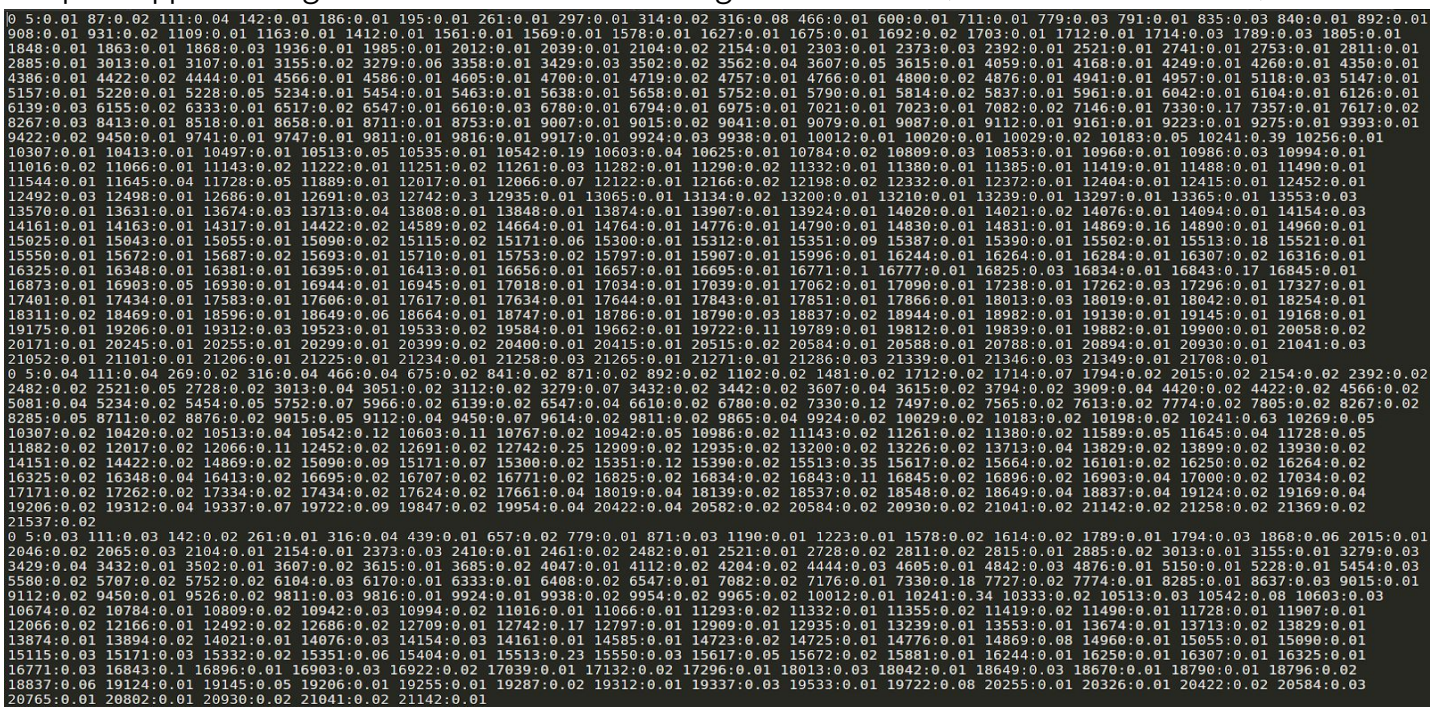

Figure 1: TF-IDF based training data with labels (target) and feature vectors in libsvm format

As shown, we have an entry of the form (<class label> <feature-id-1:feature-value-1> <feature-id-2:feature-value-2>…<feature-id-n:feature-value-n>) .

The result of executing the ***'feature-extract.py'*** program with the all the required parameters gives the following output on the screen:

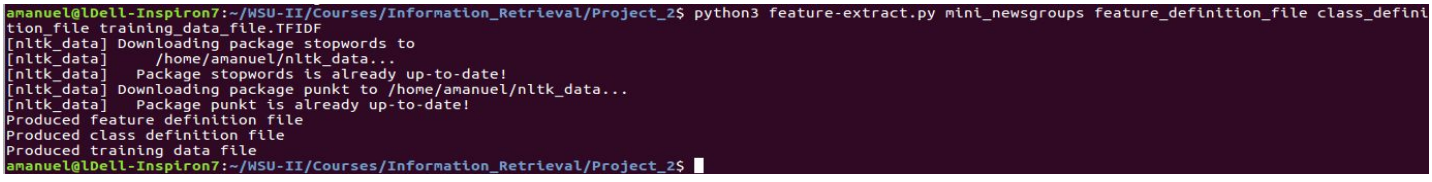

Figure 2: Snapshot showing the executing of the feature-extract.py

2. Newsgroups Classification

How to Run:
python3 classification.py

For the classification task, we utilized 4 machine learning models: Multinomial Naive Bayes, Bernoulli Naive Bayes, K-Nearest Neighbors, and Support Vector Machines. The naive bayes classification depends on basic Bayes model for probabilistic representation of bag of words for a document. Theoretically, a variant of the naive bayes classifier maximizes the conditional probability of the class label with respect to the document. It is denoted by $argmax\,(P(d \mid c).\,P(c))$ which is derived from $argmax\,P(c \mid d)$ after the application of bayes rule. This maximization procedure differentiates bernoulli naive bayes from multinomial naive bayes. In the bernoulli naive bayes, the each word feature of a document is considered to be independent and binary, whereas, in the multinomial, it integer word count for each word feature. It can be said that multinomial naive bayes is a complex model compared to bernoulli and is more vulnerable to overfitting because of requirement of large amount of data. On the other hand, the bernoulli model can easily be trained over small amount of training data.
Other pair of classifiers that can be compared are Support Vector Machines and K-Nearest Neighbors. It is because, both of the model uses distance metric (e.g. euclidean, cosine) for generating a hyperplane (a form of gaussian contour) for classifying the data points. In our study, we utilize these models without any parameter tuning to identify a reasonable classifier for classifying the documents in 20 newsgroups to 6 labels.
Our experiment used 80-20% split for training and test dataset, with a random seed for reproducibility concerns .
We defined four methods corresponding to each classifier (**MultinomialNB, BernoulliNB, KNeighbors, and svm_SVC_classifer**). Plus, we implemented method ***'evaluate_classifier()'*** to compute the performance of each classifier. Further, we implemented method ***'shuffle_train_data()'*** to shuffle our dataset when passing into our classifiers. It is needed to avoid bias or sequential pattern in the dataset and improve the predictive performance of machine learning algorithm. File ***'shuffled_train_data.txt'*** is generated as a result of shuffling method as applied on the original dataset.
Executing the ***'classification.py'*** yields the following output on the screen:

```
amanuel@lDell-Inspiron7:~/WSU-II/Courses/Information_Retrieval/Project_2$ python3 classification.py
Dimension of feature vectors: (2000, 21724)
Dimension of target vectors: (2000,)
MultinomialNB Results:
Train accuracy: 0.25125
Test accuracy: 0.245
('f1-macro : 0.07 (+/- 0.00)', 'precision-macro : 0.04 (+/- 0.00)', 'recall-macro : 0.17 (+/- 0.00)')
BernoulliNB Results:
Train accuracy: 0.729375
Test accuracy: 0.6375
('f1-macro : 0.52 (+/- 0.05)', 'precision-macro : 0.61 (+/- 0.05)', 'recall-macro : 0.52 (+/- 0.05)')
KNeighbors Results:
Train accuracy: 0.605625
Test accuracy: 0.3925
('f1-macro : 0.34 (+/- 0.03)', 'precision-macro : 0.38 (+/- 0.08)', 'recall-macro : 0.35 (+/- 0.03)')
svm_SVC Results:
Train accuracy: 0.25125
Test accuracy: 0.245
('f1-macro : 0.07 (+/- 0.00)', 'precision-macro : 0.04 (+/- 0.00)', 'recall-macro : 0.17 (+/- 0.00)')
amanuel@lDell-Inspiron7:~/WSU-II/Courses/Information_Retrieval/Project_2$
```

Figure 3: Snapshot of classification.py execution which takes in shuffled feature and target vectors and train 4 classifiers: MultinomialNB, BernoulliNB, K-Nearest Neighbors, and Support Vector Machines

3. Feature selection

The feature selection program is developed to evaluate the performance of the four classifiers defined in program **'*classification.py*'** for a selected **top k** features. We experiment with different values for k ranging from 100 to 1000. We use chi squared and mutual information for each of the k values (100-1000, 100 is selected at random. Human-in-the-loop could be beneficial for having a starting k.) and pass to our classifiers. We define method **'*plot_learning_curve()*'** that accepts the model title and a dataframe created in the main method (with data frame entries for 'k' values, Chi-square (χ^2) value, and Mutual-Information (MI) value).

How to run:

Use the following command to run the '***feature_selection.py***' program for a model:

python3 feature_selection.py model_name; where model_name is one of the models passed by the user as argument to the program. For example, to perform feature selection over our newsgroup datasets and pass to model **'*BernoulliNB*'**, we run the following command:

python3 feature_selection.py BernoulliNB

We ran the feature_selection over our dataset using χ^2 and MI and plotted the results for each classifier for different values of K (top features).

Sample output from command line is shown below:

```
amanuel@lDell-Inspiron7:~/WSU-II/Courses/Information_Retrieval/Project_2$ python3 feature_selection.py BNB
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.62875
Test accuracy: 0.575
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.49375
Test accuracy: 0.465
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.67375
Test accuracy: 0.61
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.535
Test accuracy: 0.47
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.710625
Test accuracy: 0.6625
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.576875
Test accuracy: 0.51
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.72875
Test accuracy: 0.6575
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.60125
Test accuracy: 0.535
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.73875
Test accuracy: 0.6675
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.62125
Test accuracy: 0.545
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.749375
Test accuracy: 0.685
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.651875
Test accuracy: 0.5575
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.7525
Test accuracy: 0.675
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.67875
Test accuracy: 0.585
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.763125
Test accuracy: 0.66
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.698125
Test accuracy: 0.5975
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.766875
Test accuracy: 0.6925
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.709375
Test accuracy: 0.6225
Testing classifiers(BernoulliNB_classifier) with chi2 selected features
Train accuracy: 0.773125
Test accuracy: 0.685
Testing classifiers(BernoulliNB_classifier) with mutual information selected features
Train accuracy: 0.71875
```

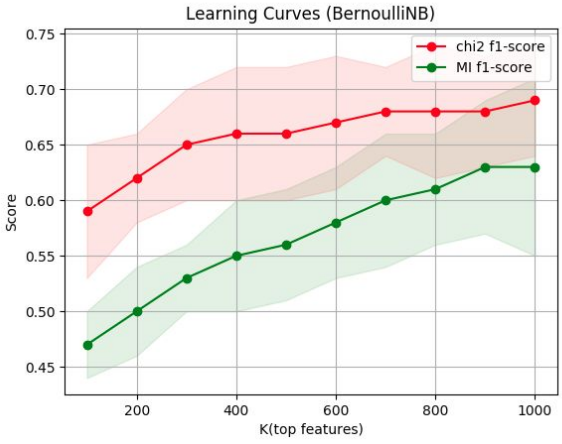

Figure 4: (Left) Execution of feature_selection.py showing the different train and test accuracy of the best performing model (BernoulliNB) for two feature selection methods: χ^2 and MI. (Right) shows the mean F1 score of BernoulliNB for χ^2 and MI. It is important to note the small variance of BernoulliNB compared to models in Figure 5 and also high f1-score.

χ^2 and MI measure is a procedure for selecting features that have relationship with the target variable. It can be seen as a measure of dimensionality reduction, with appropriate dimension is identified by means of accuracy metric. For instance, in our experiment on newsgroup, we observe that after selecting between

800 to 1000 features, which is 4-5% of the total features (~21000), the model is able to achieve a stable and high accuracy compared to process in the absence of feature selection. χ^2 is analogous to distance metric, whereas the MI is analogous to entropy measure (see Figure 4).

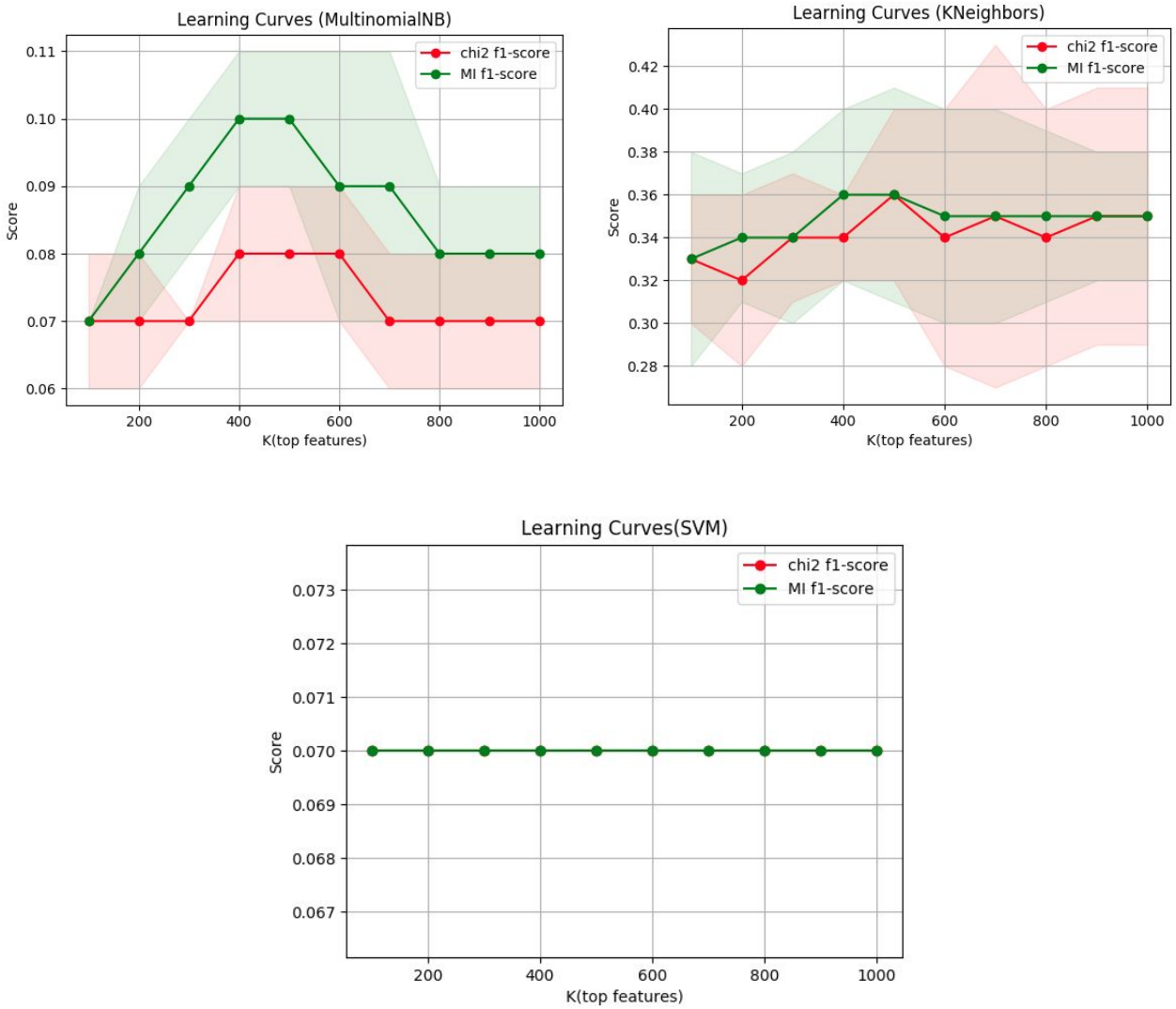

Figure 5: (Top-Left) Trend in F1-Score with changing 'k' in the feature selection procedures for the MultinomialNB and (Top-right) K-Nearest Neighbors. It is observable that both the models have shown increased variance in the results compared to BernoulliNB. As a result the models are unstable on the given amount of data. (Bottom-middle) Support Vector Machines did not show any improvement in the F1-score with changing k number of features using feature selection procedures: χ^2 and MI. Interestingly there was no variance in the performance of SVM. All the three plots are performance plots of three machine learning algorithms: MultinomialNB, K-Nearest Neighbors, and SVM.

4. Document Clustering

For the clustering the documents, we utilized **K-Means and Agglomerative** clustering algorithms and evaluated the performance using silhouette metric and normalized mutual information. The two clustering algorithms differs in their approach to cluster the feature vectors, wherein, K-Means identifies k number of centroids representing clusters and allocate a data point to a cluster by minimizing the inter-cluster

distance. Incase of agglomerative clustering, there is a formation of cluster tree where each node represent a cluster of similar documents. The minimization function is similar to K-Means, however, the k parameter is used to identify the threshold for generating **k-clusters**. In layman terms, a flat line is drawn over the cluster tree, decomposing the tree into k-clusters. Since, both the algorithms are parametric on k, we used silhouette and normalized mutual information (NMI) metric to evaluate the cluster stability. The silhouette metric evaluates the cluster stability without taking into account the target labels, making it an unsupervised metric. On the other hand, the NMI takes into account the target variable, and its interpretation is information-content centric, analogous to entropy. A high score on silhouette metric is desired as it explains the high stability in cluster. From the document clustering plot of K-Means (see Figure 7), we observe high cluster stability when number of clusters is less than 5. Whereas the interpretation of NMI is as follows: when the number of clusters (k) is set to 2 and the number of class labels are 6, what is the NMI or quality of the clustering. Similar to silhouette, higher NMI is desired, and from the plot of K-Means (Figure 7), it is achieved when the number of clusters are 18.

How to run the program:

python3 clustering.py {kmeans, agglomerative}
We tested K-means and agglomerative clustering separately and passed each as argument in the call to the '*clustering.py*' program.
Sample command for running the K-Means clustering:
python3 clustering.py agglomerative

Sample output for agglomerative yields the following:

```
amanuel@lDell-Inspiron7:~/WSU-II/Courses/Information_Retrieval/Project_2$ python3 clustering.py agglomerative
number of clusters:  2
silhoutte_score:  0.196828705058575
nmi_score:  0.014887914581989892
number of clusters:  3
silhoutte_score:  0.08015531054872187
nmi_score:  0.01538897340465141
number of clusters:  4
silhoutte_score:  0.0745871114719176
nmi_score:  0.015964755696721226
number of clusters:  5
silhoutte_score:  0.027145012493355984
nmi_score:  0.019500407635282883
number of clusters:  6
silhoutte_score:  0.028085231144517792
nmi_score:  0.019822058564659364
number of clusters:  7
silhoutte_score:  0.015059324412288773
nmi_score:  0.021639629353578157
number of clusters:  8
silhoutte_score:  0.015840555877799558
nmi_score:  0.022094678784226555
number of clusters:  9
silhoutte_score:  0.015471644375032751
nmi_score:  0.026543486214773416
number of clusters:  10
silhoutte_score:  0.016033356599060456
nmi_score:  0.027132757205434524
number of clusters:  11
silhoutte_score:  -0.008388495071893183
nmi_score:  0.036599465520834665
number of clusters:  12
silhoutte_score:  -0.007102995506751115
nmi_score:  0.038244938987908889
number of clusters:  13
silhoutte_score:  -0.0031980989325568645
nmi_score:  0.040741395390103
number of clusters:  14
silhoutte_score:  -0.0016123600146670542
nmi_score:  0.04121002366916924
number of clusters:  15
silhoutte_score:  0.0017481077550081112
nmi_score:  0.04439054529119546
number of clusters:  16
silhoutte_score:  0.0023506255881149654
nmi_score:  0.0447203843682901
number of clusters:  17
silhoutte_score:  0.0021438921269804 99
nmi_score:  0.045731423973004476
number of clusters:  18
silhoutte_score:  0.002316509695389734
nmi_score:  0.046054580524107575
number of clusters:  19
silhoutte_score:  0.0028565333871119957
nmi_score:  0.04660922466361704
number of clusters:  20
silhoutte_score:  0.0027518321357684174
nmi_score:  0.04844843826209807
number of clusters:  21
silhoutte_score:  0.002393485452323779
```

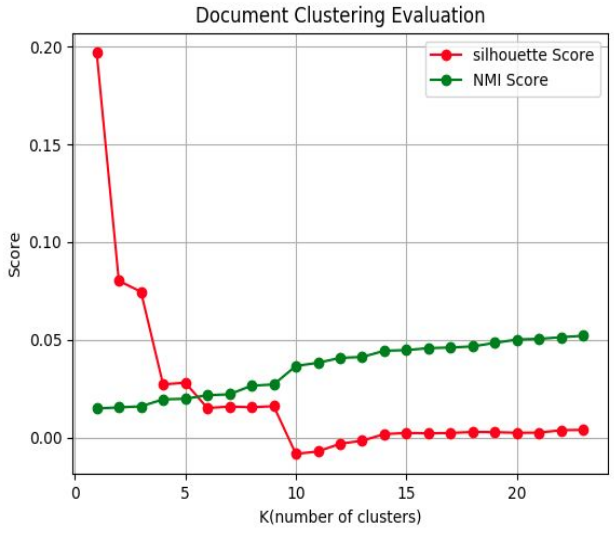

Figure 6: (Left) Execution of Document Clustering python program : clustering.py for Agglomerative Clustering. (right) The plot of document clustering evaluation using NMI and Silhouette metric. The are

outliers being present in the dataset as seen from the negative Silhouette score and also can be the reason of disagreement between NMI and silhouette.

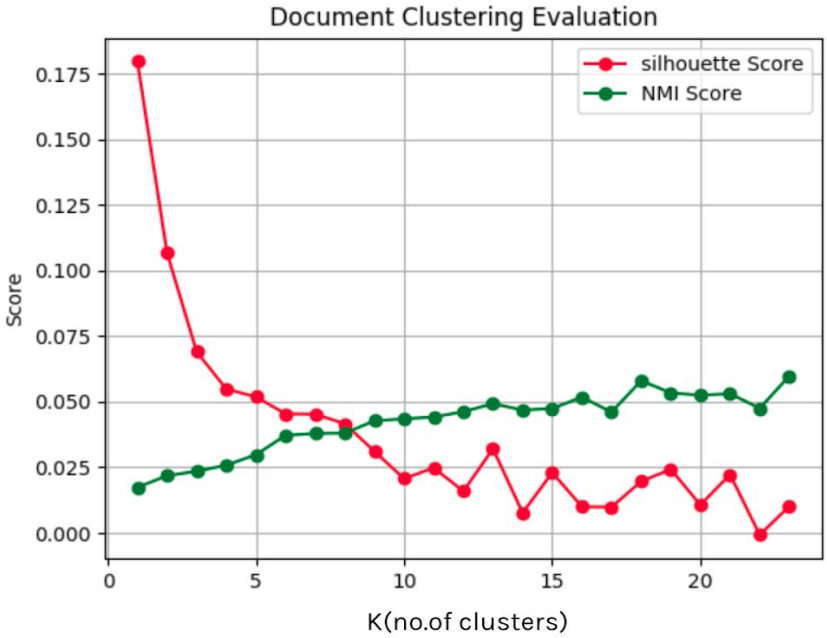

Figure 7: The plot of document clustering evaluation using NMI and Silhouette metric for K-Means Clustering. As seen from the curve, there are no identified outliers in the dataset as seen from the non-negative Silhouette score. Further, we observe high silhouette score for number of clusters = 2, which furnishes the high f1-score for BernoulliNB which considers binary features.

From figures 7 and 8, we observe that both K-Means and Agglomerative clustering have high silhouette score for k=2 number of clusters. However, for the NMI, K-Means has the highest NMI for k=18, whereas for agglomerative clustering it is k=25. Further, we notices near-monotonic increase in NMI with the number of clusters which support independence assumptions over features for the Naive Bayes models.

References:

1. https://nlpforhackers.io/building-a-simple-inverted-index-using-nltk/
2. https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
3. https://scikit-learn.org/stable/modules/cross_validation.html