Assignment-3

Question 1

In Question 1, dataset used is 20_newsgroups in which we worked on 5 labels- `'comp.graphics', 'rec.sport.hockey', 'sci.med', 'sci.space', 'talk.politics.misc'`

Aim

- Train and test Naive Bayes on different ratios
- Calculated and use TF-ICF feature selection for Naive Bayes
- Compare both the models on different ratios

Steps performed in preprocessing

- Removing stop words like is, are
- Converting numbers to words
- Removing special symbols
- Removing extra spaces
- Performing lemmatization and stemming
- Converting whole text to lower case
- Performing word tokenization
- Removing word whose length is less than two

Now, storing the words after preprocessing with their labels in the DataFrame. Saving the DataFrame using pickle. After this, we have built two models from scratch, one is the normal Naive Bayes model and another is Naive Bayes with TF-ICF.

Models

1. Naive Bayes includes,
    - In the training phase,

      a). created a dictionary to store the words per class

      b). created bag of words containing the total words in the train set

      c). calculating the frequency of each word in the specific class and also calculating the number of words in each class

      d). calculating the frequency of the occurrences of each class in y_train

- In the testing phase,

  a). calculating the class-wise probability for each word set in X_test

  b). there will be nested loops, the first loop will be of X_test in which one by one-word set will be iterated, the For inner loop will be of classes and the innermost loop will be of words in word set of that X_test[i]. Now, class-wise the probability of each word in the word set is calculated according to the word frequency in that class and the total number of words in that specific class.

  c). smooth the probability value by adding 1 in the numerator and adding the length of a bag of words in the denominator.

  d). used np.log() and addition instead of multiplication to prevent the probability from getting zero.

  e). after calculating the probability of the word set, consider the class to which that word set belongs whose probability is maximum.

2. Naive Bayes using TF-ICF includes,
   - In the training phase,

     a). same steps used in the training phase of the normal Naive Bayes model

     b). calculating TF ICF values, where TF is the term frequency the number of times that word is occurring in the bag of words and ICF is the log of total classes divided by the number of classes in which that word occurs + 1 for normalization in numerator and denominator.

     c). After calculating TF ICF values for each word, now sort the dictionary in reverse order according to the TF ICF values and select the top k features from the dictionary. In our implementation, we have selected the top 10% of features from the dictionary.

   - In the testing phase,

     a). same steps used in the testing phase of the normal Naive Bayes model.

     b). only the modification is in the probability part in which we add the length of the bag of words in normal Naive Bayes but here we will add the length of the dictionary having TF ICF values.

     Significance-

Performing feature selection on TF ICF values is a good technique as it only includes important features rather than a bag of words.

*Both the models use a confusion matrix and accuracy function for the output.

Comparison between both the models

- In the second model i.e. Naive Bayes using TF ICF has less noise due to feature selection which only selects the important words whose TF ICF value is more than other words.
- Naive Bayes doesn't need a large dataset; it only needs correct sets of words for a specific class so that it can predict more accurately.
- Naive Bayes with TF ICF performs better than the normal Naive Bayes because of top k feature selection.

Question 2

For Question 2, we have selected a dataset named **Wiki-Vote**.

About Dataset :

Wikipedia is a free encyclopedia written collaboratively by volunteers around the world. A small part of Wikipedia contributors are administrators, who are users with access to additional technical features that aid in maintenance. For a user to become an administrator, a Request for adminship (RfA) is issued and the Wikipedia community via a public discussion or a vote decides who to promote to adminship. Using the latest complete dump of Wikipedia page edit history (from January 3, 2008) we extracted all administrator elections and voting history data. This gave us 2,794 elections with 103,663 total votes and 7,066 users participating in the elections (either casting a vote or being voted on). Out of these 1,235 elections resulted in a successful promotion, while 1,559 elections did not result in the promotion. About half of the votes in the dataset are by existing admins, while the other half comes from ordinary Wikipedia users.

Dataset Statistics :

Nodes	7115
Edges	103689
Nodes in largest WCC	7066 (0.993)
Edges in largest WCC	103663 (1.000)
Nodes in largest SCC	1300 (0.183)
Edges in largest SCC	39456 (0.381)
Average clustering coefficient	0.1409
Number of triangles	608389
Fraction of closed triangles	0.04564
Diameter (longest shortest path)	7
90-percentile effective diameter	3.8

Data Analysis and Exploration :

Dataset is available in text format. The first four lines contain basic text about the data. The fourth line contains the heading “FromNodeId” and “ToNodeId”. From the fifth line main Information starts that helped in forming an adjacency list for graph representation.

From the adjacency list, we get the following results -

```
Number of nodes :  7115
Number of edges :  103689
```

```
Average In-Degree :  14.573295853829936
Average Out-Degree :  14.573295853829936
```

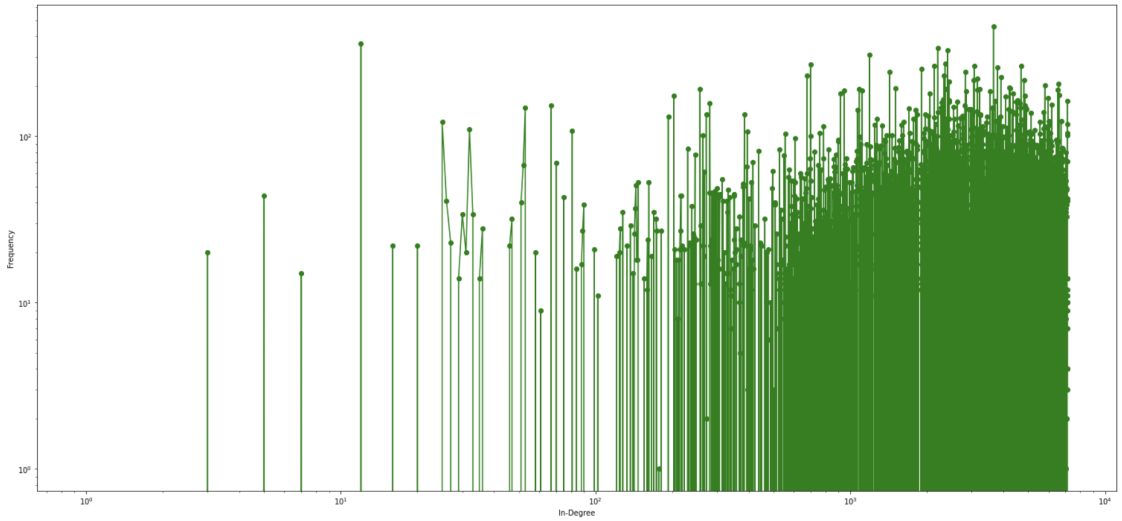
```
Node with maximum  457  In-Degree :  Node  4037
Node with maximum  893  Out-Degree :  Node  2565
```

```
Density of the network is :  0.0020485375110809584
```

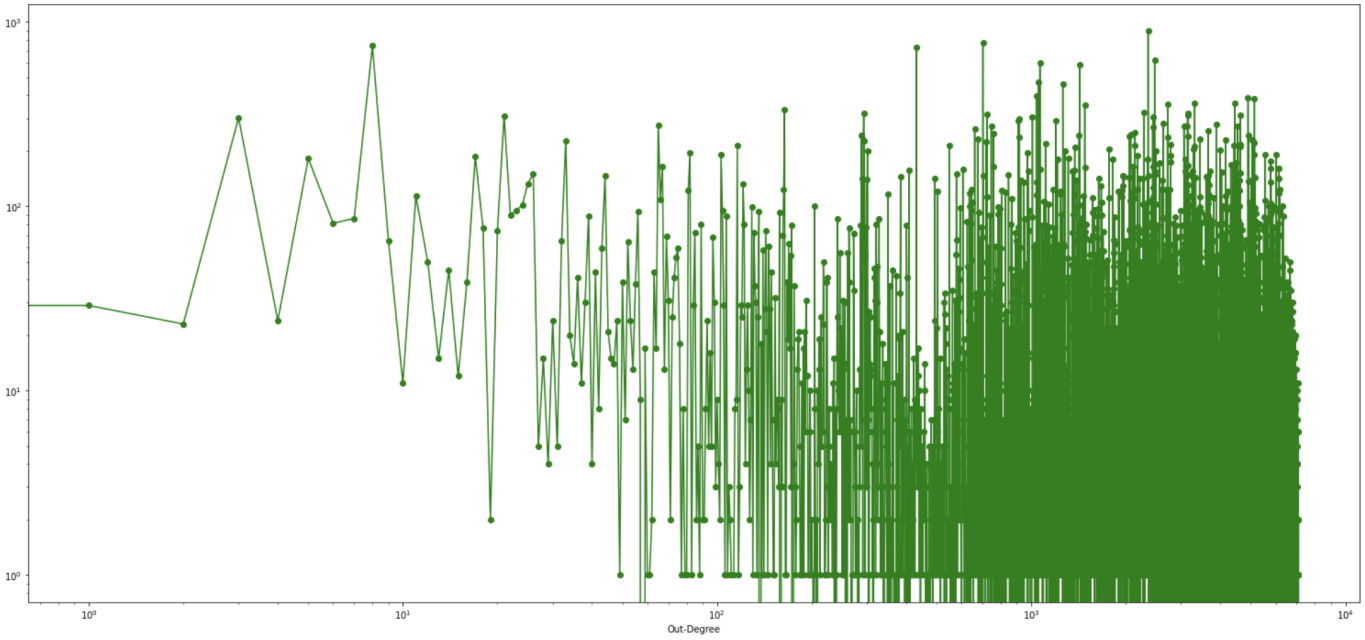
Results :

1. Plot degree distribution of the network (in case of a directed graph, plot in-degree and out-degree separately).

In-Degree of node x1 refers to the number of edges incident to x1.



Out-Degree of node x1 refers to the number of edges that emerged from x1 to other nodes.



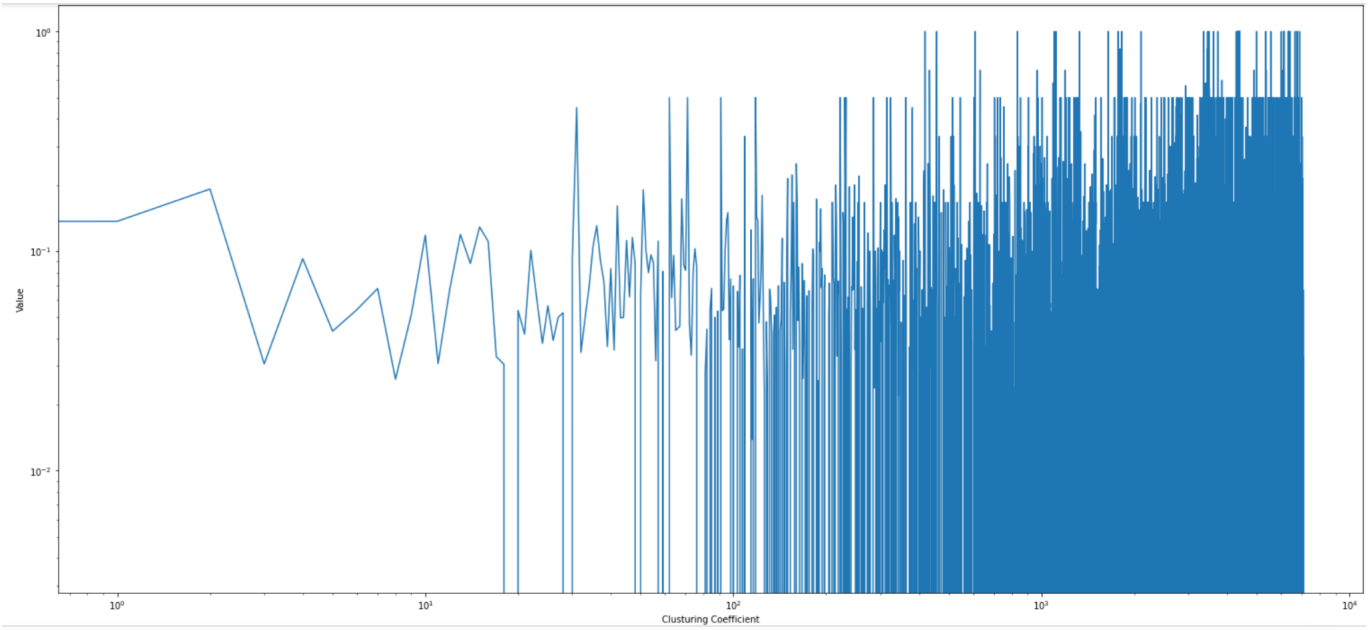
2. Calculate the local clustering coefficient of each node and plot the clustering coefficient distribution of the network.

The local clustering coefficient for a vertex is given by the proportion of links between the vertices within its neighborhood divided by the number of links that could exist between them.

Local Clustering Coef. = n/m*(m-1)

n: No. of links between the vertices within its neighborhood

m: No. of possible links



3. Find any 1 centrality measure for each node. Store the values in a separate text file.

Degree centrality of a node refers to the number of edges attached to the node. To know the standardized score, you need to divide each score by n-1

Degree Centrality = n/(m-1)

n: No. of links between the vertices within its neighborhood

m: No. of nodes

After calculation Degree Centrality we stored the data in the text file named dCentrality.txt which contains degree centrality with the corresponding node as (Node: Degree-Centrality)

dCenterality.txt

```
1  3 : 0.005342
2  4 : 0.015603
3  5 : 0.013635
4  6 : 0.391763
5  7 : 0.007169
6  8 : 0.200028
7  9 : 0.049339
8  10 : 0.069441
9  11 : 2.024318
10 12 : 0.029941
11 13 : 0.001827
12 14 : 0.055665
13 15 : 0.023053
14 16 : 0.003514
15 17 : 0.024459
16 18 : 0.002390
17 19 : 0.023053
18 20 : 0.161653
```

Part B.
On the same wiki-Vote dataset, we calculate different scores for each node namely:

1. PageRank score for each node
2. Hub and Authority score for each node

Library used:

'NetworkX' is a python library package that is used for the creation, manipulation, and study of different graphical structures and networks.
In this part of the assignment, we use two algorithm libraries:

1. PageRank algorithm library (to calculate the PageRank score of each node)
2. HITS algorithm library (to calculate both Hub and Authority scores of each node)

For PageRank score:

PageRank is an algorithm used by google search to rank websites in their search engine results i.e, measuring the importance of website pages.
This algorithm works by counting the number and quality of links to a page to roughly estimate and determine the importance of the website as more important websites (nodes) are likely to receive more links (incoming edges/links) from other website pages.

Now, we define the damping factor in the PageRank algorithm with the notion that any imaginary surfer randomly clicking links will eventually stop clicking. Therefore, the probability that at any step a person continues surfing the links is known as the 'damping factor. Upon testing different damping factor values, the general assumption regarding the value is thought to be around 0.85.

By taking the default value of damping factor (i.e., 0.85) and a maximum number of iterations as 100, we calculate the PageRank scores of each node as follows:
The node number along with their PageRank scores (top to lowest) is as shown in the snippet below:

```
{'4037': 0.0046127158911675485,
 '15': 0.0036812207295292792,
 '6634': 0.003524813657640259,
 '2625': 0.0032863743692309023,
 '2398': 0.0026053331717250192,
 '2470': 0.0025301053283849546,
 '2237': 0.002504703800483994,
 '4191': 0.0022662633042363454,
 '7553': 0.002170185049195958,
 '5254': 0.0021500675059293235,
 '1186': 0.0020438936876029153,
 '2328': 0.0020416288860889186,
 '1297': 0.0019518608216122285,
 '4335': 0.0019353014475784877,...
```

For Hub and Authority scores:

Given a query to a search engine, the set of highly relevant web pages are called roots. They are potential authorities and the pages that are not quite relevant but point/ lead to pages in the root or authority are referred to as Hubs.
Thus, the authority score of each node is based on the incoming nodes from Hubs since it is a page that many Hubs link to whereas, Hub score of each node is based on the outgoing links that links them to the authority nodes.

Hyperlink Induced Topic Search (HITS) algorithm is a link-analysis algorithm that rates webpages by calculating their Hub and Authority scores.

Authority Scores for each node:

```
#Top to least Authorities score for each node
dict(sorted(authorities.items(), key=lambda item: item[1], reverse = True))
```

```
{'2398': 0.002580147178008918,
 '4037': 0.002573241124142803,
 '3352': 0.002328415091537902,
 '1549': 0.0023037314804751075,
 '762': 0.00225587485637424,
 '3089': 0.0022534066884266454,
 '1297': 0.00225014463679536,
 '2565': 0.002223564103945871,
 '15': 0.00220154349254381,
 '2625': 0.0021978968035237852,
 '2328': 0.0021723715454129585,
 '2066': 0.0021070409397065445,
 '4191': 0.0020811941305208825,...
```

Hub Scores for each node:

```
[ ]  #Top to least Hub Score for each node
     dict(sorted(hubs.items(), key=lambda item: item[1], reverse = True))
```

```
{'2565': 0.007940492708074 03,
 '766': 0.007574335297444512,
 '2688': 0.006440248991012525,
 '457': 0.00641687049019565,
 '1166': 0.006010567902433343,
 '1549': 0.0057207540583986485,
 '11': 0.004921182064008282,
 '1151': 0.004572040701802756,
 '1374': 0.004467888792672376,
 '1133': 0.003918881732047633,...
```

Comparison:

Both authority score and pagerank score are used to understand the importance and relevance of a web page based on their incoming links. But upon tallying the results and analysing the outputs, we see that different nodes are marked as highly important/relevant for both the score metrics.
This is due to the fact that PageRank score takes the node that has the maximum in-degree that is node 4037 (as seen in the output of both Part A in-degree value and Part B PageRank score).
On the other hand, Authority scores are high only for those nodes who have incoming links from the maximum number of Hub nodes, which may not include all the in-degree linked nodes from the graphical network.
Regarding Hub Score, we refer to the node with reference to their outgoing nodes thus, the node with maximum number of outgoing links (out-degrees) is node 2565 as seen in both Part A output and in the top Hub Score node value.
