Report :

The K-NN classifier has been implemented from scratch using python.

- The given matrix file "**news_data.mtx"** was read into the python program using "**mmread**" from **Scipy** library.
- The rows are read as the document number, column is the term in the document, and data is the frequency of the term in each document.
- These are read as separately and joined together as a list of lists .Each document in the matrix will be a separate list in a very big list.

  Eg: [1, 45612, 2] – 1 – document number, 45612-term, 2-frequency.

A list of such lists is created.

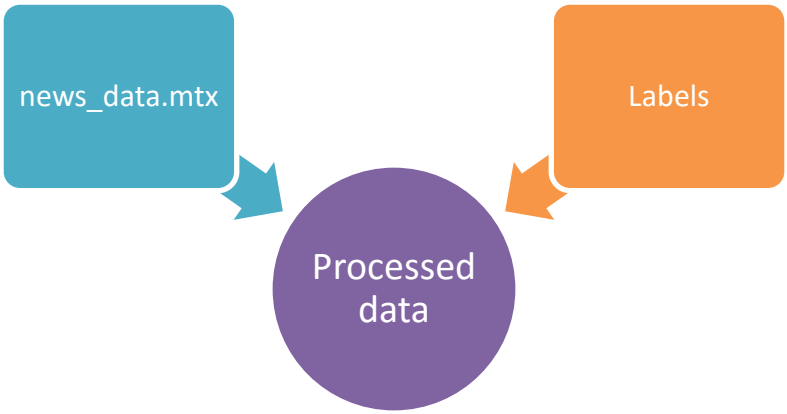
The Labels file give is changed to .txt format for easier input and then read into python. This file was then striped using regular expressions and the entries are taken as a list.

Eg: ['1','business']

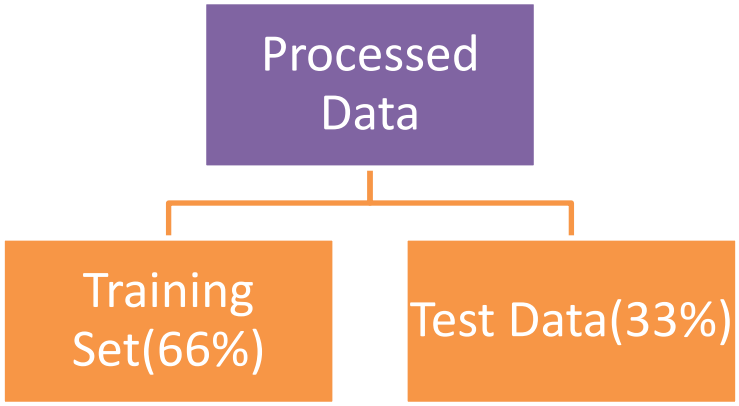
Now this list and the list formed by the matrix are joined together by comparing the document number and a separate list is formed.

Eg: ['1', '45612', '2','business']

This list is set as our processed data.



Then training set and test set are separated by 66% of the processed data. This data is taken at random each time.



Each test instance is taken in a loop and Cosine similarity is calculated between test instance document and each training instance document and this similarity is attached to a dictionary with a training instance and its corresponding similarity. They are then ordered in descending and the “k” most similar is taken.

Then K most similar are then sent to a get response method where the class is kept as the key and the number of documents in the K number of documents is kept as values. So the class with the highest value (the class variable that is repeating) is assigned for the new test instance.

Weighting scheme:

A weighting scheme is developed by taking the inverse of all the cosine similarity of all the training set documents with respect to a test instance. Then the values are added for each class variable.

For eg: if the class “business” has two entries in “k” documents then the weights of them are added, similarly the weights are added for each class. The class with the highest weight is assigned to the test instance.

This program is developed such that it can handle any number of class variables given to it.

The Accuracy is calculated for UnWeighted scheme and Weighted scheme for K values from 1-10 and are tabulated below. A sample screenshot is taken for both and kept below.

Screenshots:

```
predicted='business', actual='business'
predicted='business', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
Accuracy of UnWeighted Classifier
Accuracy: 58.44155844155844%
```

```
predicted='business', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
predicted='technology', actual='business'
Accuracy of weighted Classifier
Accuracy: 71.72859450726979%
```

K Value	Accuracy of Non Weighted Classifier	Accuracy of Weighted Classifier
1	69.7933	60.8695
2	72.7987	63.8036
3	63.7873	71.4953
4	77.9365	54.83333
5	71.5591	75.7234
6	76.1363	51.3334
7	77.0400	71.4056
8	75.8454	67.1826
9	69.9186	54.0510
10	58.6688	71.7285

Interpretation of Results:

From the results we can infer that the weighted classifier here is not accurate as that of the unweighted one. The reason is because when we inverse a cosine similarity we will get a very less value resulting in the opposite of similarity.

For example: if similarity is 0.9 then they are very similar and 0.3 means they are almost different with slight similarities.

But when we inverse it: 1/0.9 = 1.111 and 1/0.3 = 3.33.

So even if there are 2 exactly same documents their weights will add up to only 2 but weight of one single very dissimilar document will lead to a higher weight and get assigned.

Future works:

This weighting scheme can be improved by adding some other functionality like tfidf and then the weights can be calculated accordingly.