ASSIGNMENT-05

ANALYSIS

MI-KNN:

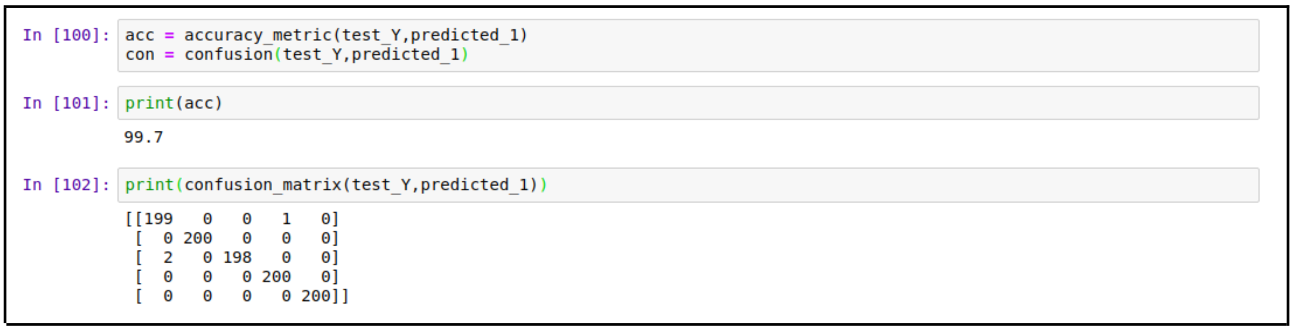
Formula used:

1. Formula used computing Mutual information:

$$I(U;C) = \frac{N_{11}}{N} log_2 \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N} log_2 \frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N} log_2 \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N} log_2 \frac{NN_{00}}{N_{0.}N_{.0}}$$

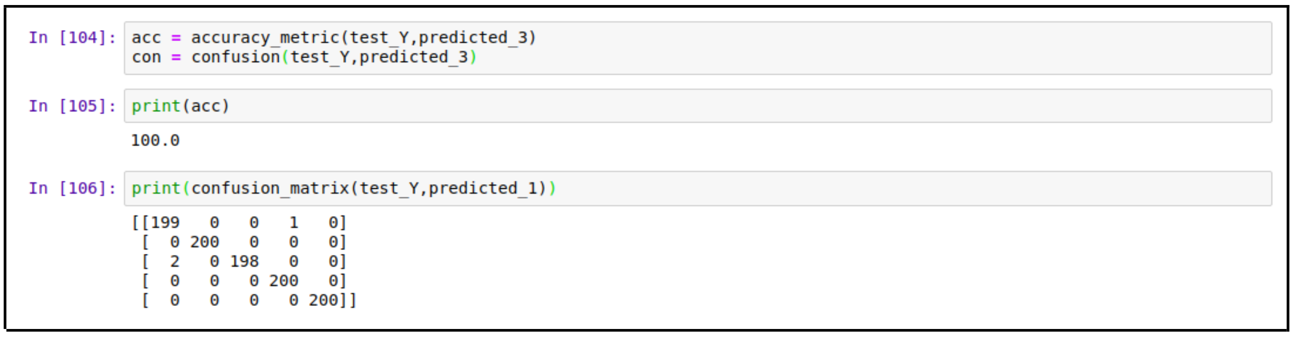
split ratio – 80:20
value of k: 0.7%

1NN



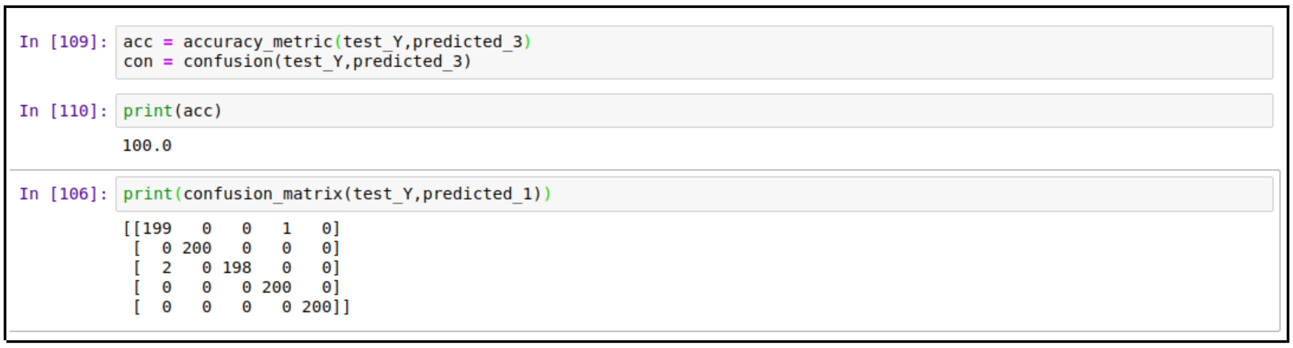
```
In [100]: acc = accuracy_metric(test_Y,predicted_1)
          con = confusion(test_Y,predicted_1)

In [101]: print(acc)

          99.7

In [102]: print(confusion_matrix(test_Y,predicted_1))

          [[199   0   0   1   0]
           [  0 200   0   0   0]
           [  2   0 198   0   0]
           [  0   0   0 200   0]
           [  0   0   0   0 200]]
```

3NN



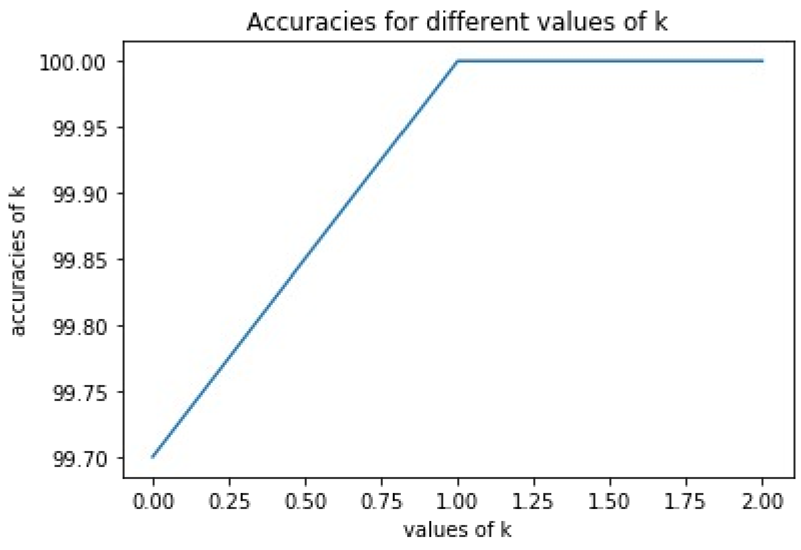
```
In [104]: acc = accuracy_metric(test_Y,predicted_3)
          con = confusion(test_Y,predicted_3)

In [105]: print(acc)

          100.0

In [106]: print(confusion_matrix(test_Y,predicted_1))

          [[199   0   0   1   0]
           [  0 200   0   0   0]
           [  2   0 198   0   0]
           [  0   0   0 200   0]
           [  0   0   0   0 200]]
```

5NN



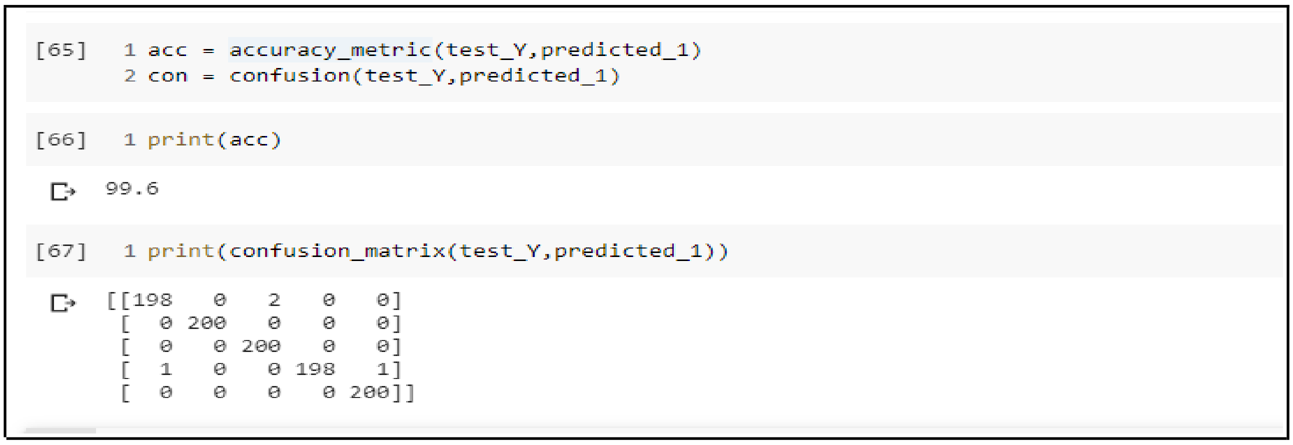
```
In [109]: acc = accuracy_metric(test_Y,predicted_3)
          con = confusion(test_Y,predicted_3)

In [110]: print(acc)

          100.0

In [106]: print(confusion_matrix(test_Y,predicted_1))

          [[199   0   0   1   0]
           [  0 200   0   0   0]
           [  2   0 198   0   0]
           [  0   0   0 200   0]
           [  0   0   0   0 200]]
```

Graph between values of k and accuracies



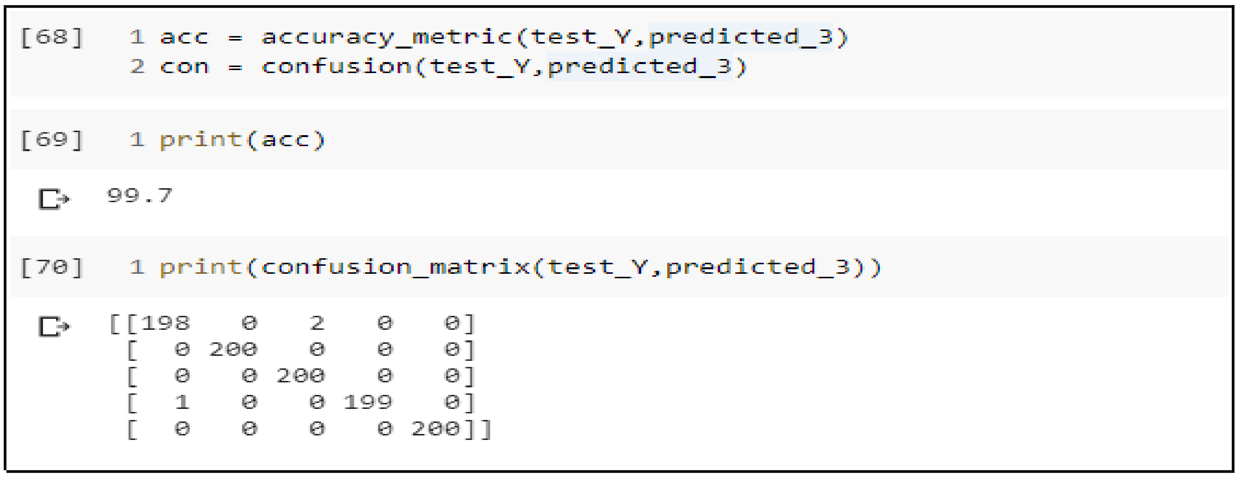
split ratio-80:20
value of k – 0.4%

1NN



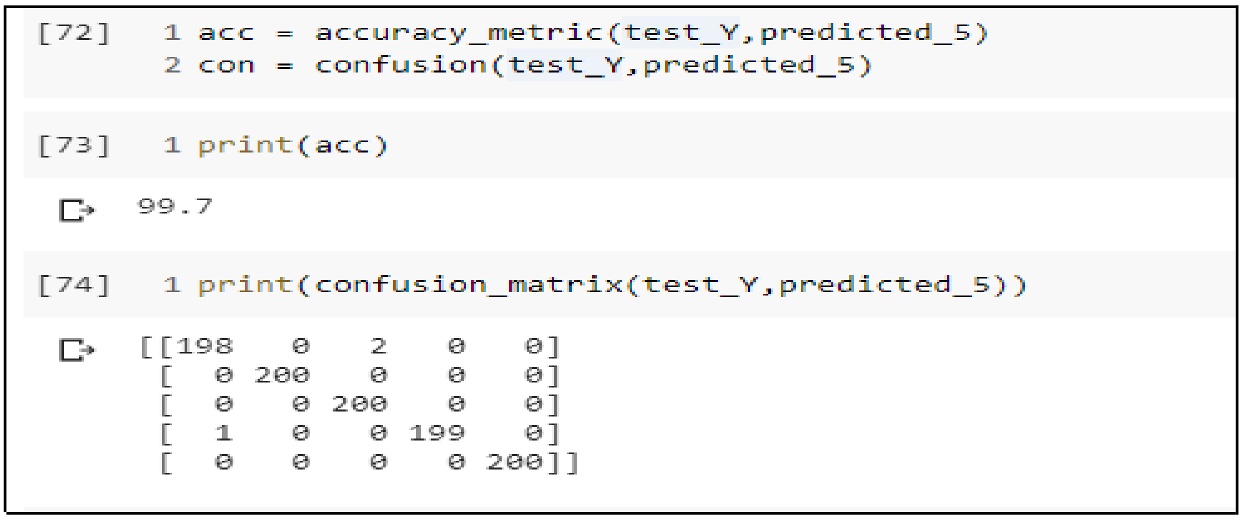
```
[65]    acc = accuracy_metric(test_Y,predicted_1)
        con = confusion(test_Y,predicted_1)

[66]    print(acc)

    99.6

[67]    print(confusion_matrix(test_Y,predicted_1))

    [[198   0   2   0   0]
     [  0 200   0   0   0]
     [  0   0 200   0   0]
     [  1   0   0 198   1]
     [  0   0   0   0 200]]
```

3NN



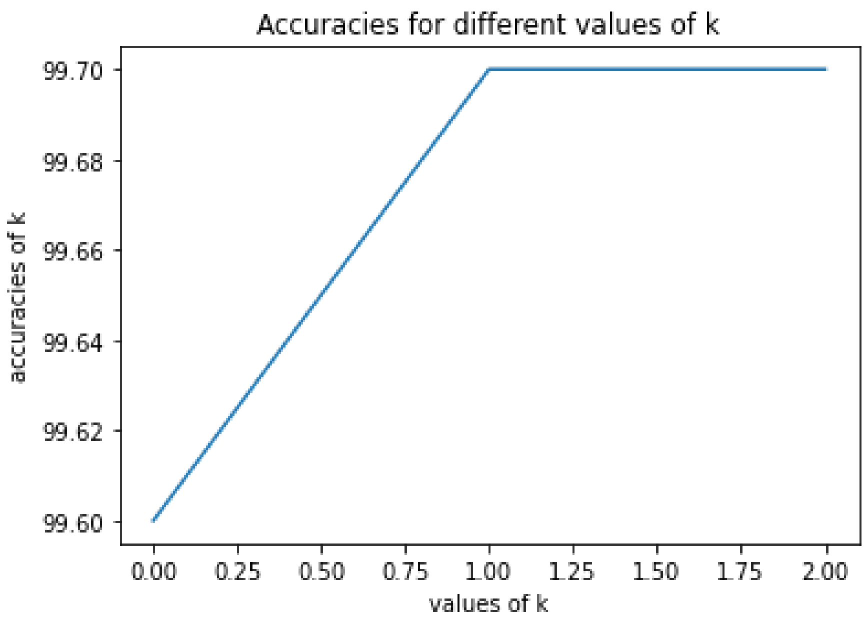
```
[68]    acc = accuracy_metric(test_Y,predicted_3)
        con = confusion(test_Y,predicted_3)

[69]    print(acc)

    99.7

[70]    print(confusion_matrix(test_Y,predicted_3))

    [[198   0   2   0   0]
     [  0 200   0   0   0]
     [  0   0 200   0   0]
     [  1   0   0 199   0]
     [  0   0   0   0 200]]
```

5NN



```
[72]    acc = accuracy_metric(test_Y,predicted_5)
        con = confusion(test_Y,predicted_5)

[73]    print(acc)

    99.7

[74]    print(confusion_matrix(test_Y,predicted_5))

    [[198   0   2   0   0]
     [  0 200   0   0   0]
     [  0   0 200   0   0]
     [  1   0   0 199   0]
     [  0   0   0   0 200]]
```

Graph between values of k and accuracies:



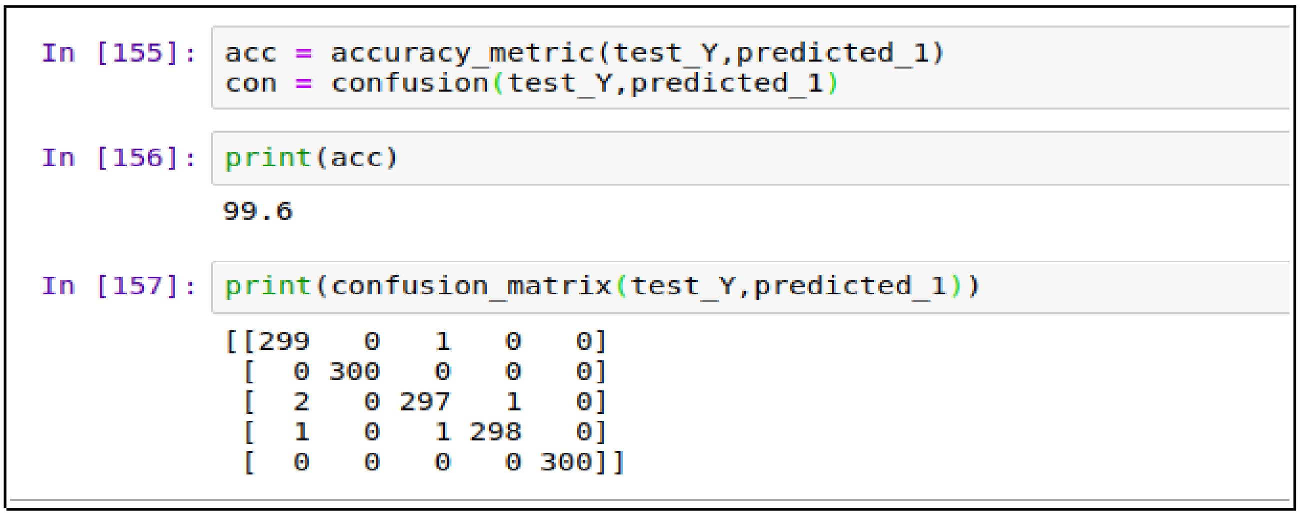
Inferences:

- There’s a slightest decrease in accuracy due to less number of feature selected. May be it’s due to the fact fe relevant features are not taken into account. Thus from this observation we can conclude that while increasing the percentage of K the unique words tends to increase. That gives rise to high accuracy.
- In this case first class is misclassified in all the casses except for 3nn and 5nn of 0.7 % top K feature selection. It may be possible that terms are not informative and as relevant as compared to others.
- Selecting weak indicators may also affect the classification accuracy of selected features set.

- It is also observed that 1nn accuracy is less than 3nn and 5nn, it’s due to the fact 1nn is less robust. The classification of each document relies on sigle training data,which may be incorrectly labelled.

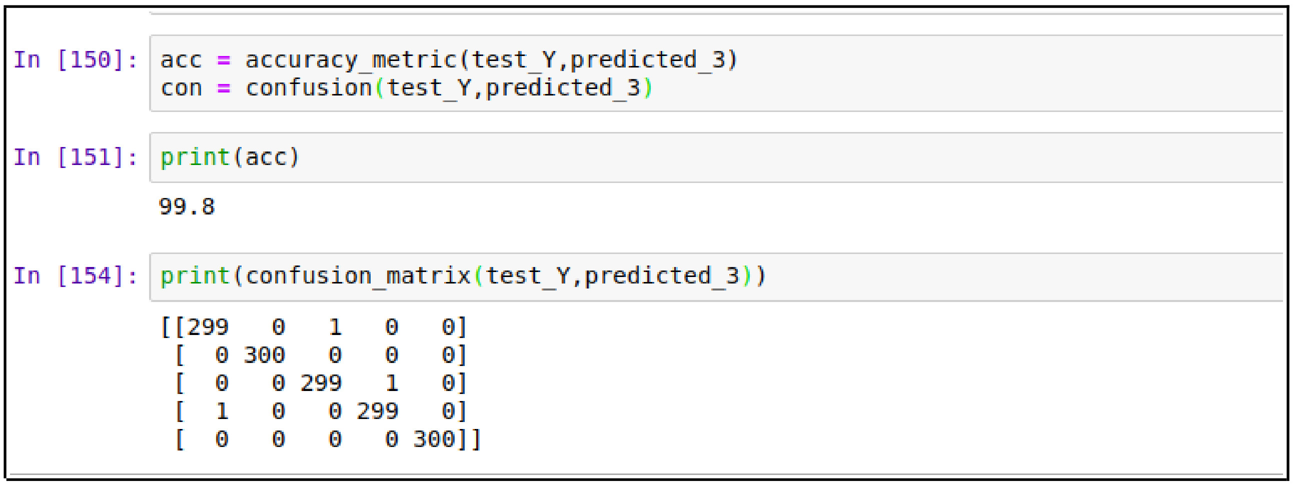
split ratio 70-30
value of k 0.7 %

1NN



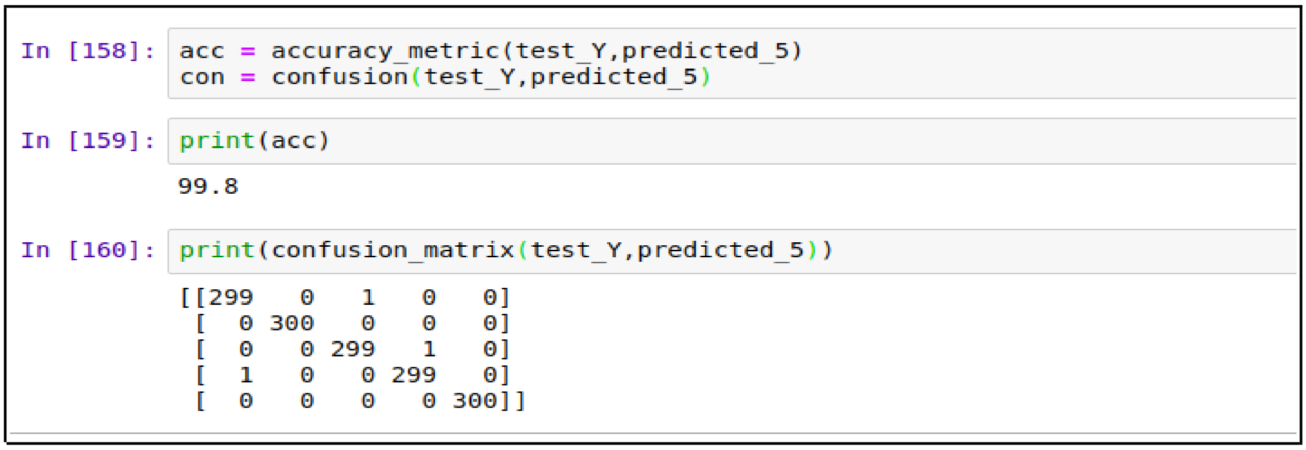
```
In [155]: acc = accuracy_metric(test_Y,predicted_1)
          con = confusion(test_Y,predicted_1)

In [156]: print(acc)

          99.6

In [157]: print(confusion_matrix(test_Y,predicted_1))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  2   0 297   1   0]
           [  1   0   1 298   0]
           [  0   0   0   0 300]]
```

3NN



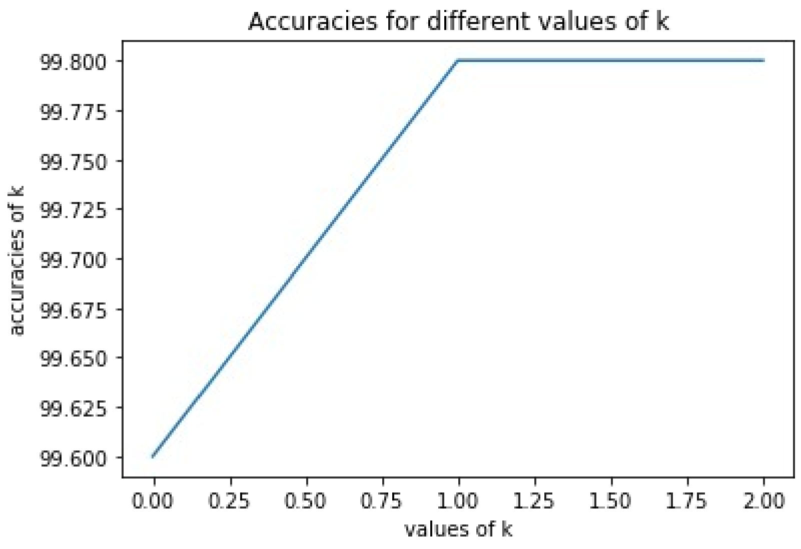
```
In [150]: acc = accuracy_metric(test_Y,predicted_3)
          con = confusion(test_Y,predicted_3)

In [151]: print(acc)

          99.8

In [154]: print(confusion_matrix(test_Y,predicted_3))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  0   0 299   1   0]
           [  1   0   0 299   0]
           [  0   0   0   0 300]]
```

5NN



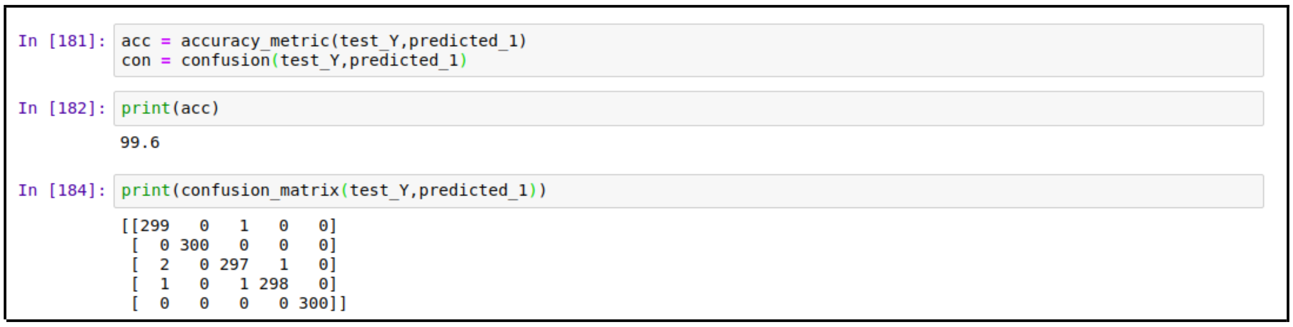
```
In [158]: acc = accuracy_metric(test_Y,predicted_5)
          con = confusion(test_Y,predicted_5)

In [159]: print(acc)

          99.8

In [160]: print(confusion_matrix(test_Y,predicted_5))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  0   0 299   1   0]
           [  1   0   0 299   0]
           [  0   0   0   0 300]]
```

Graph between values of k and accuracies:



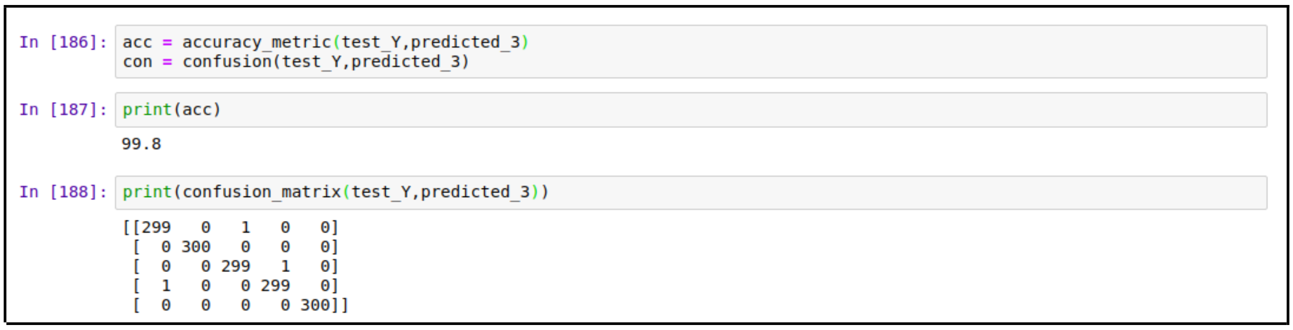
split ratio-70:30
values of k – 40%

1NN



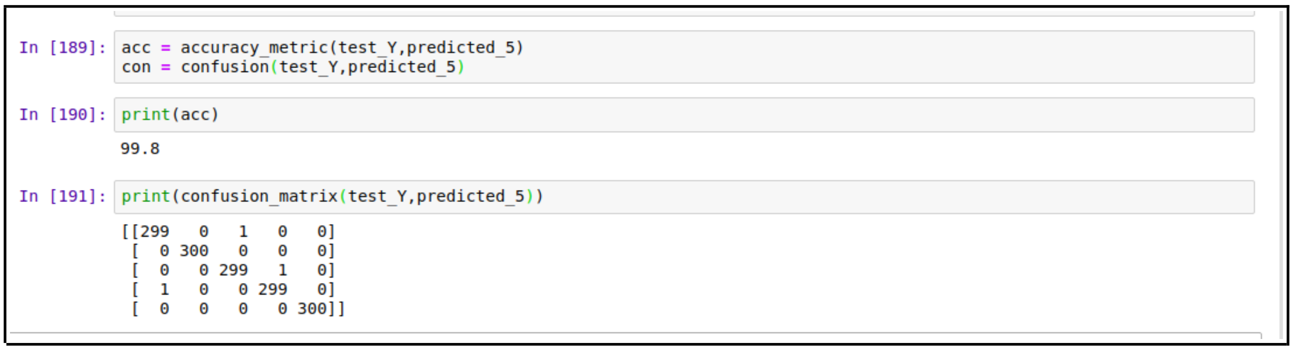
```
In [181]: acc = accuracy_metric(test_Y,predicted_1)
          con = confusion(test_Y,predicted_1)

In [182]: print(acc)

          99.6

In [184]: print(confusion_matrix(test_Y,predicted_1))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  2   0 297   1   0]
           [  1   0   1 298   0]
           [  0   0   0   0 300]]
```

3NN



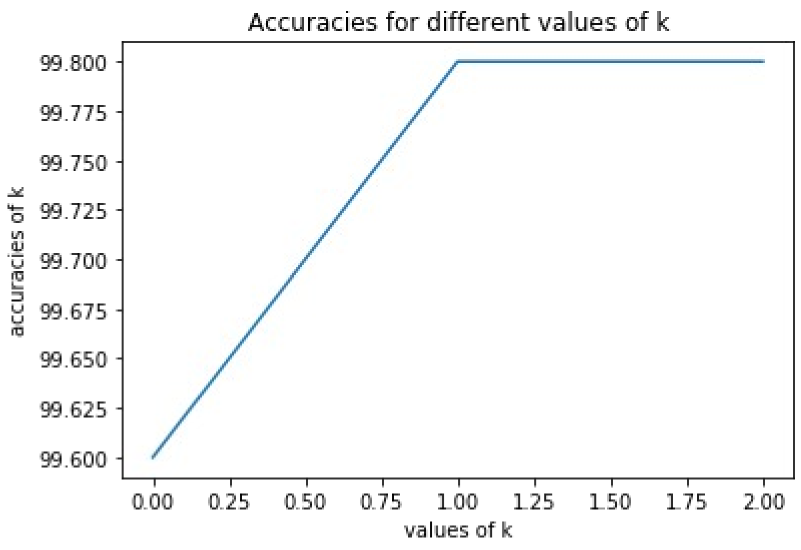
```
In [186]: acc = accuracy_metric(test_Y,predicted_3)
          con = confusion(test_Y,predicted_3)

In [187]: print(acc)

          99.8

In [188]: print(confusion_matrix(test_Y,predicted_3))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  0   0 299   1   0]
           [  1   0   0 299   0]
           [  0   0   0   0 300]]
```

5NN



```
In [189]: acc = accuracy_metric(test_Y,predicted_5)
          con = confusion(test_Y,predicted_5)

In [190]: print(acc)

          99.8

In [191]: print(confusion_matrix(test_Y,predicted_5))

          [[299   0   1   0   0]
           [  0 300   0   0   0]
           [  0   0 299   1   0]
           [  1   0   0 299   0]
           [  0   0   0   0 300]]
```

Graph between values of k and accuracies:

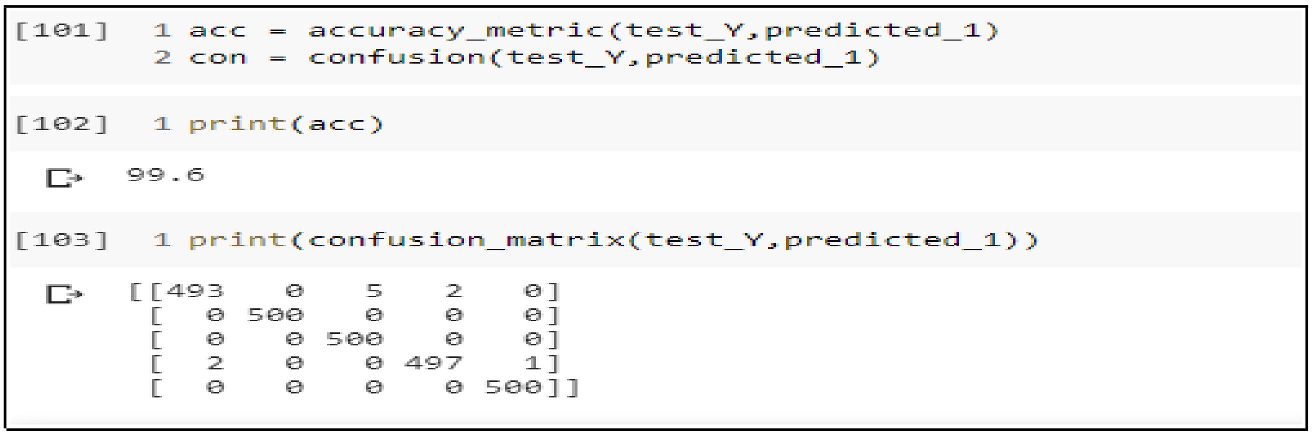


Inferences:

- It can be seen in both the top percentage K values that accuracies are same.
- There are three classes which are misclassified in both the cases which again includes comp.graphics as in 80:20 split.
- The
- Reason for misclassification may be the selection of features that contribute no incremental information over previous selected features that in case 0.7 - 0.4 %.
- Accuracy is same in case of 3nn and 5nn as boundary becomes more smoother if we increase the value of k, it’s also more than 1nn in both the cases.
- Since 1nn is not very robust and classification decision relies on single training document. Thus, it’s more likely that document can get incorrect class label.
- It is desireable for ‘k’ to be odd to make ties less likely, generally k=3 and k=5 are considered as best choices for large amount data.

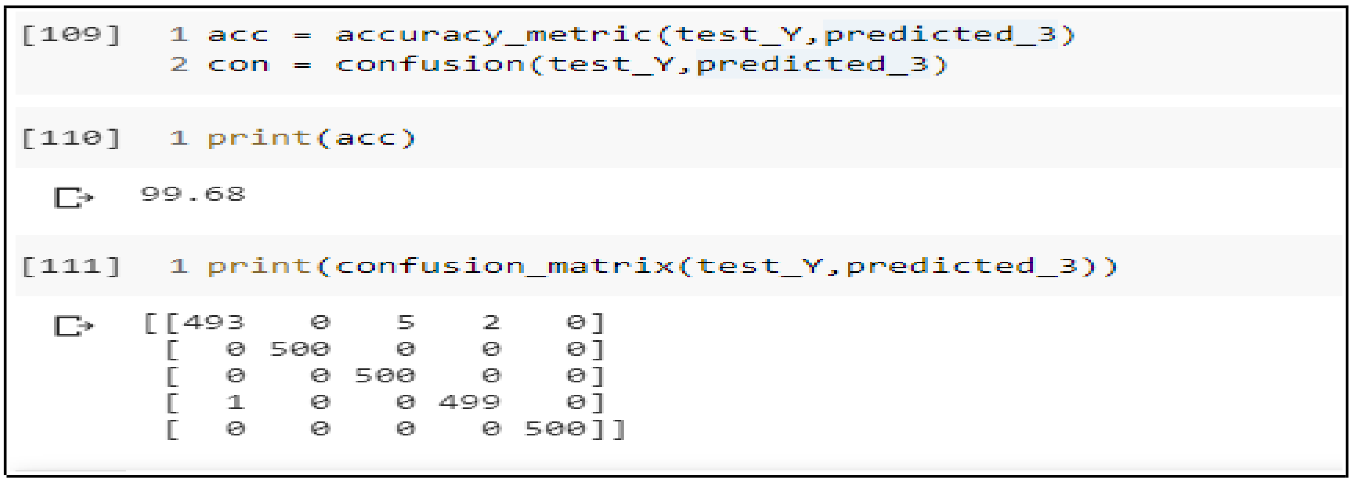
split ratio 50:50
value of k – 70

1NN



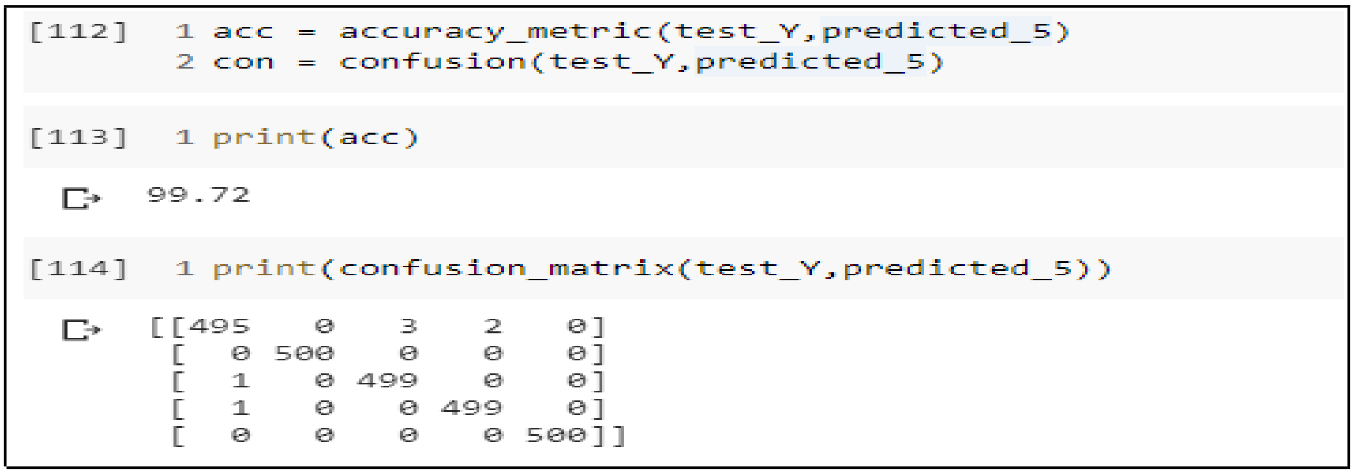
```
[101]   acc = accuracy_metric(test_Y,predicted_1)
        con = confusion(test_Y,predicted_1)

[102]   print(acc)

        99.6

[103]   print(confusion_matrix(test_Y,predicted_1))

        [[493    0    5    2    0]
         [   0 500    0    0    0]
         [   0    0 500    0    0]
         [   2    0    0 497    1]
         [   0    0    0    0 500]]
```

3NN



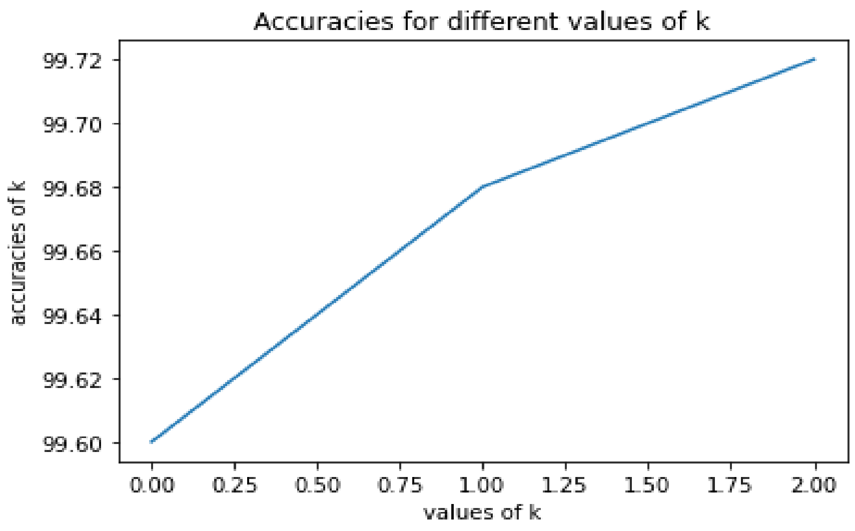
```
[109]   acc = accuracy_metric(test_Y,predicted_3)
        con = confusion(test_Y,predicted_3)

[110]   print(acc)

        99.68

[111]   print(confusion_matrix(test_Y,predicted_3))

        [[493    0    5    2    0]
         [   0 500    0    0    0]
         [   0    0 500    0    0]
         [   1    0    0 499    0]
         [   0    0    0    0 500]]
```

5NN



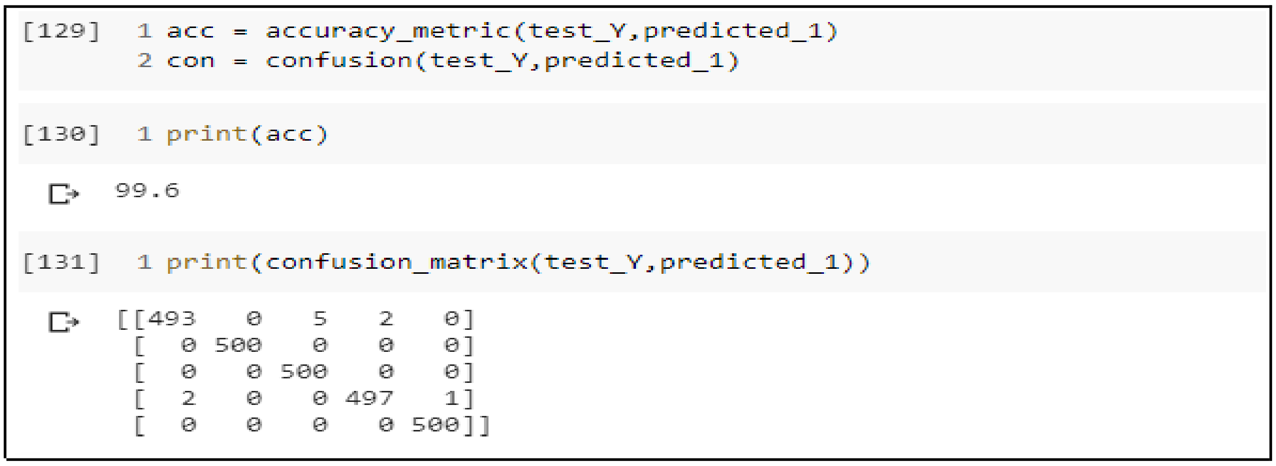
```
[112]   acc = accuracy_metric(test_Y,predicted_5)
        con = confusion(test_Y,predicted_5)

[113]   print(acc)

        99.72

[114]   print(confusion_matrix(test_Y,predicted_5))

        [[495    0    3    2    0]
         [   0 500    0    0    0]
         [   1    0 499    0    0]
         [   1    0    0 499    0]
         [   0    0    0    0 500]]
```

Graph between values of k and accuracies:



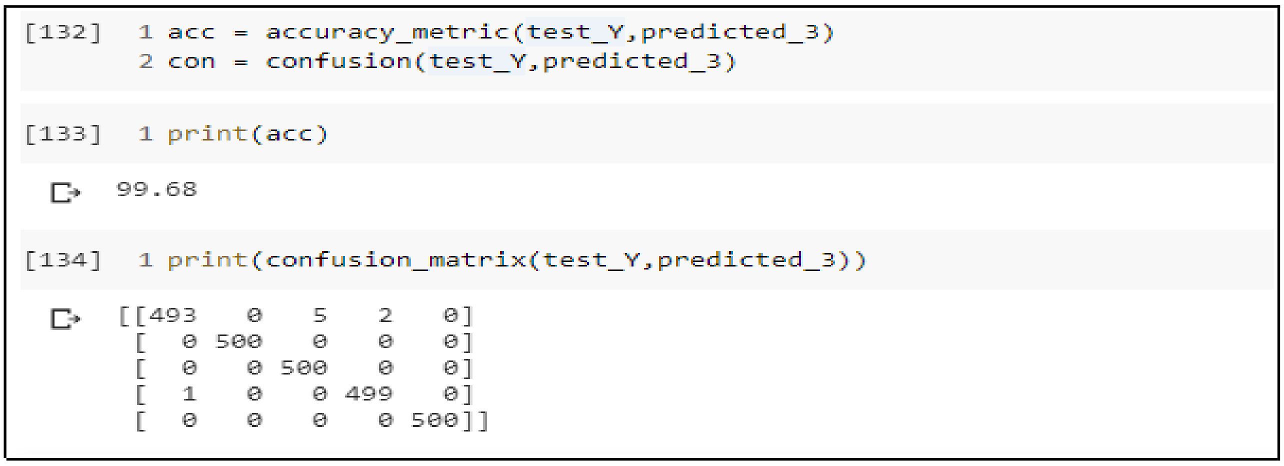
split 50-50:
value of k- 0.4 %

1NN



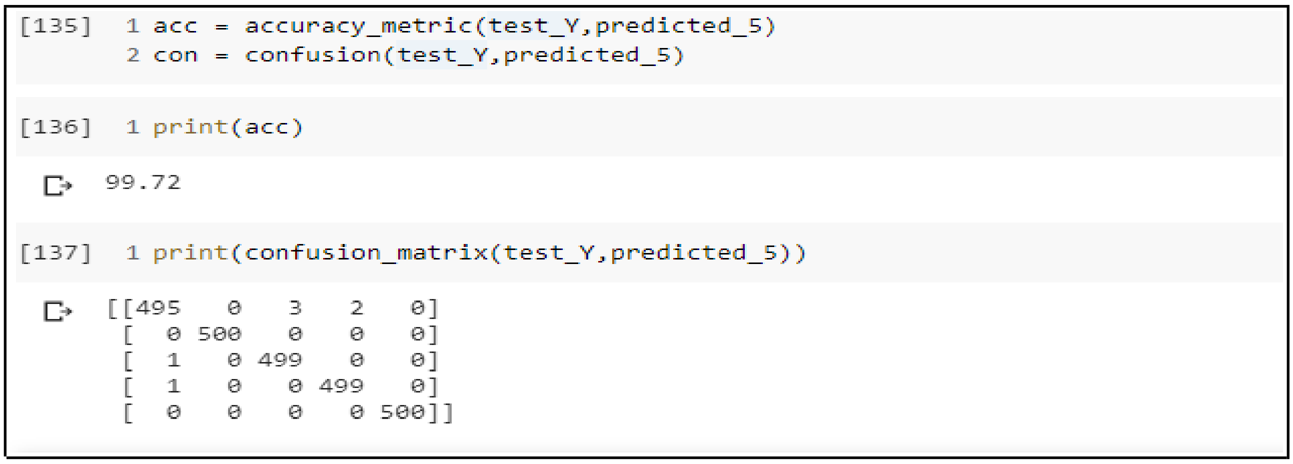
```
[129]  acc = accuracy_metric(test_Y,predicted_1)
       con = confusion(test_Y,predicted_1)

[130]  print(acc)

    99.6

[131]  print(confusion_matrix(test_Y,predicted_1))

    [[493    0    5    2    0]
     [   0  500    0    0    0]
     [   0    0  500    0    0]
     [   2    0    0  497    1]
     [   0    0    0    0  500]]
```

3NN



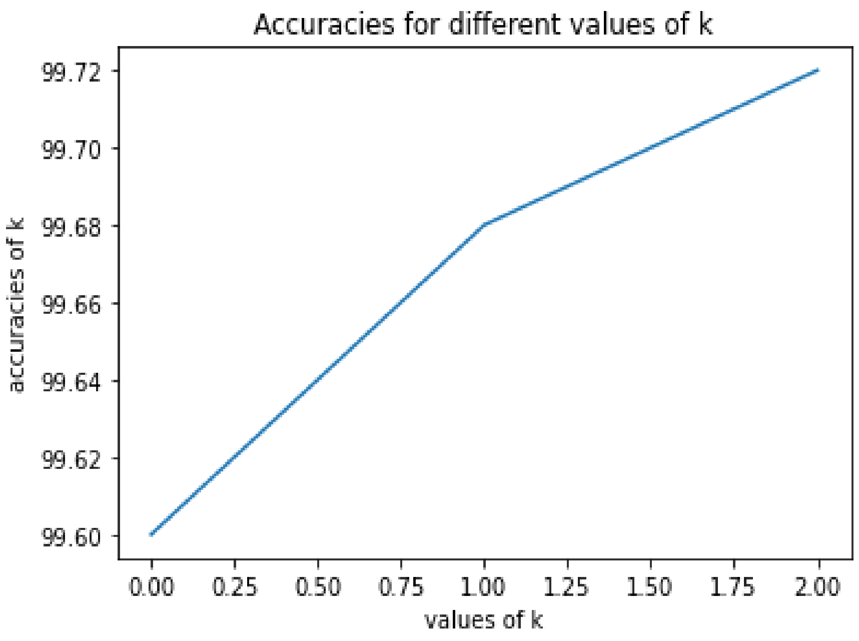
```
[132]  acc = accuracy_metric(test_Y,predicted_3)
       con = confusion(test_Y,predicted_3)

[133]  print(acc)

    99.68

[134]  print(confusion_matrix(test_Y,predicted_3))

    [[493    0    5    2    0]
     [   0  500    0    0    0]
     [   0    0  500    0    0]
     [   1    0    0  499    0]
     [   0    0    0    0  500]]
```

5NN



```
[135]  acc = accuracy_metric(test_Y,predicted_5)
       con = confusion(test_Y,predicted_5)

[136]  print(acc)

       99.72

[137]  print(confusion_matrix(test_Y,predicted_5))

       [[495   0   3   2   0]
        [  0 500   0   0   0]
        [  1   0 499   0   0]
        [  1   0   0 499   0]
        [  0   0   0   0 500]]
```

Graph between values of k and accuracies:



Inferences:

- Similar to above split 2 reasons are same.
- The misclassification of class1 is same in both 1nn and 3nn but it decreases in case of 5nn. Thus 5nn has least test error and also decreases the noise in the data.
- Boundary becomes smoother if we increase the value of k.
- It is observed that while decreasing the size the of split and giving less data for training the accuracy is decreasing and the number of misclassification increases.
- Although there's a slightest decrease in accuracy but it has been observed that more training data tends to result in better accuracy and generates more generalized model.

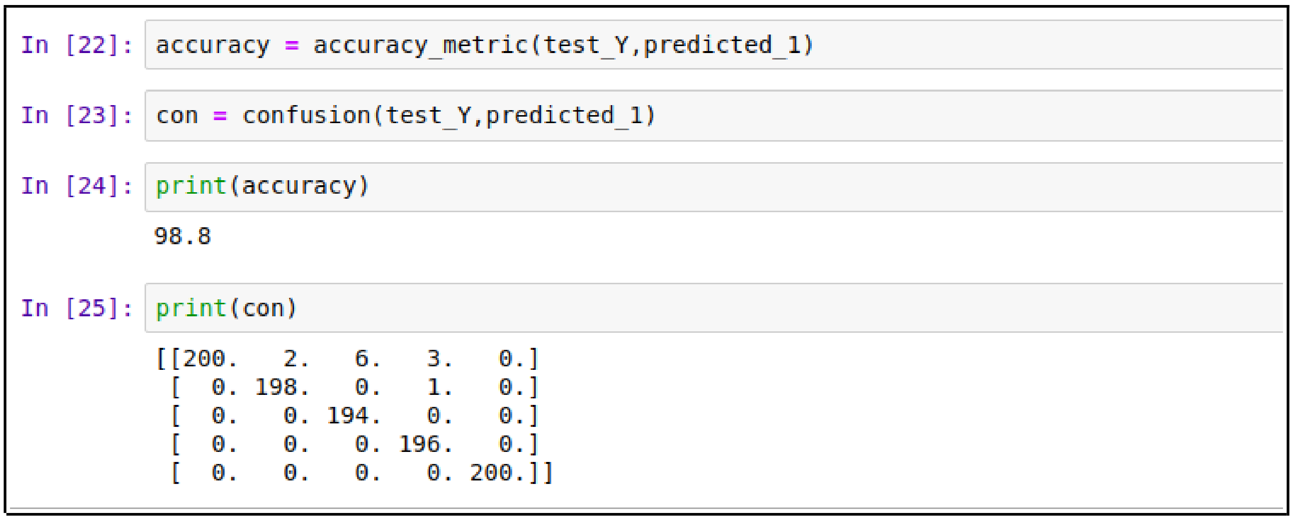
TF-IDF KNN

1. Formula used for computing TF-IDF:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

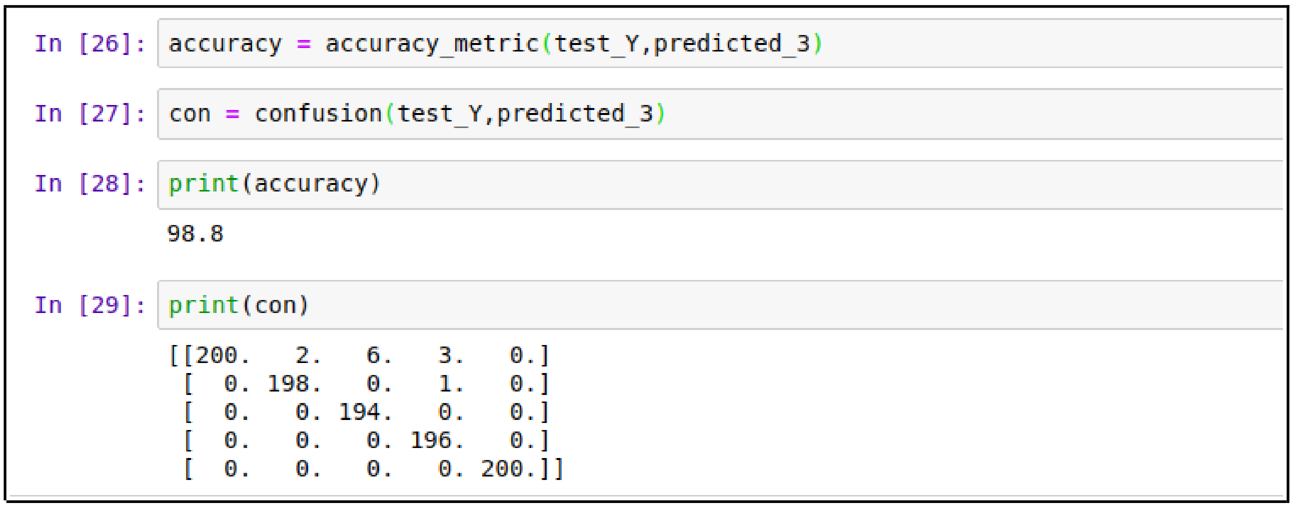
split ratio – 80:20
value of k 0.7 %

1NN



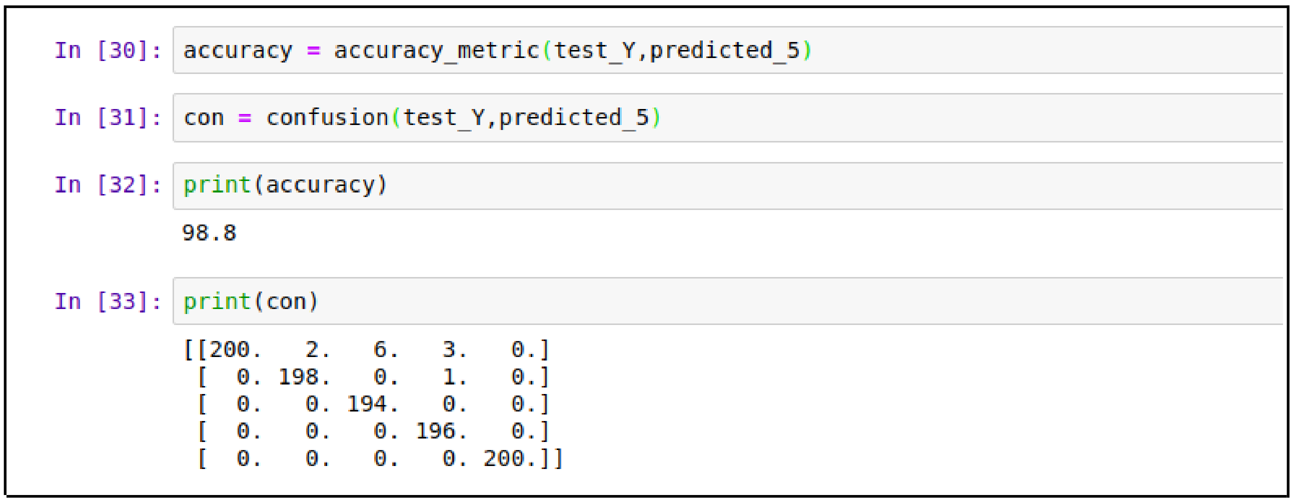
```
In [22]: accuracy = accuracy_metric(test_Y,predicted_1)

In [23]: con = confusion(test_Y,predicted_1)

In [24]: print(accuracy)

         98.8

In [25]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

3NN



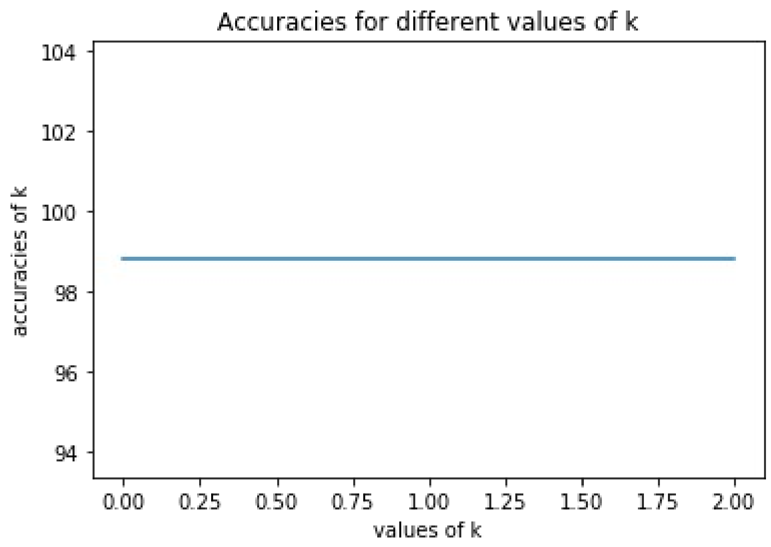
```
In [26]: accuracy = accuracy_metric(test_Y,predicted_3)

In [27]: con = confusion(test_Y,predicted_3)

In [28]: print(accuracy)

         98.8

In [29]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

5NN



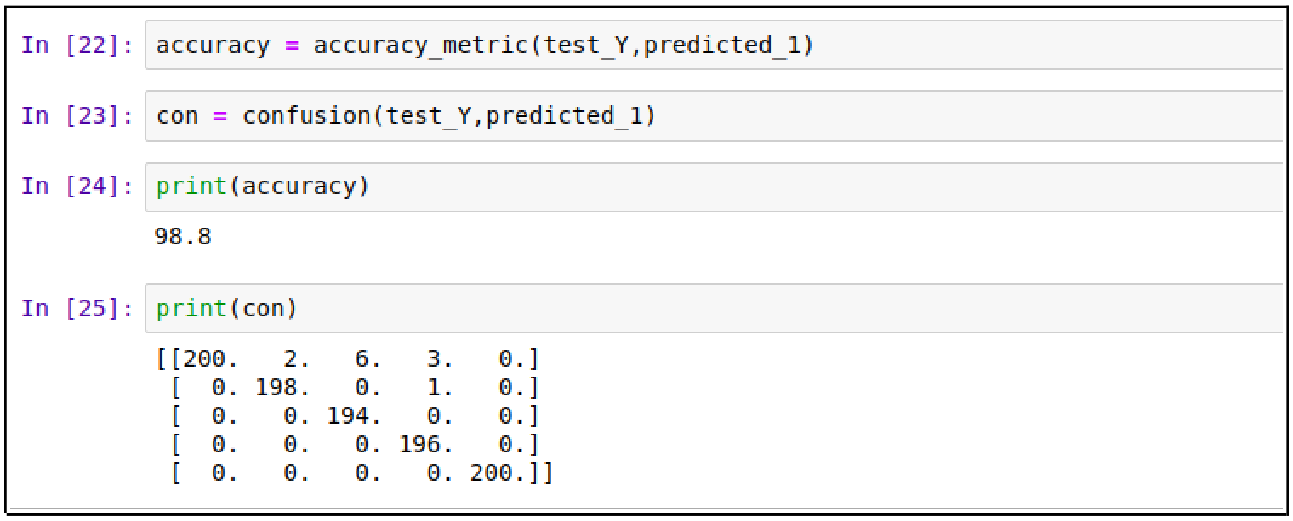
```
In [30]: accuracy = accuracy_metric(test_Y,predicted_5)

In [31]: con = confusion(test_Y,predicted_5)

In [32]: print(accuracy)

         98.8

In [33]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

Graph between values of k and accuracies:



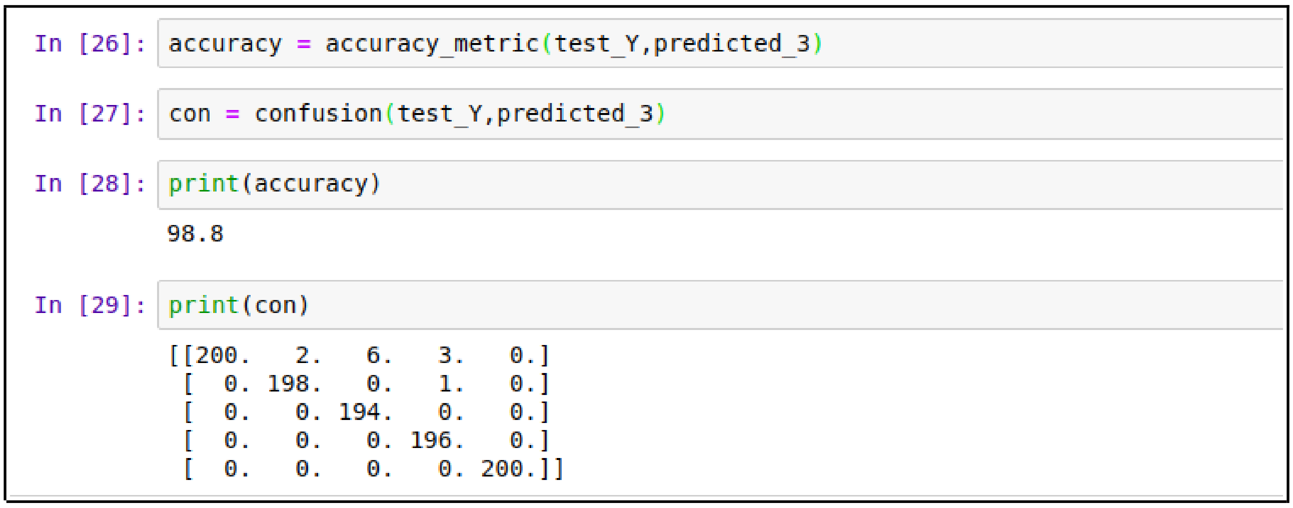
split ratio 80-20
value of k 0.4 %

1NN



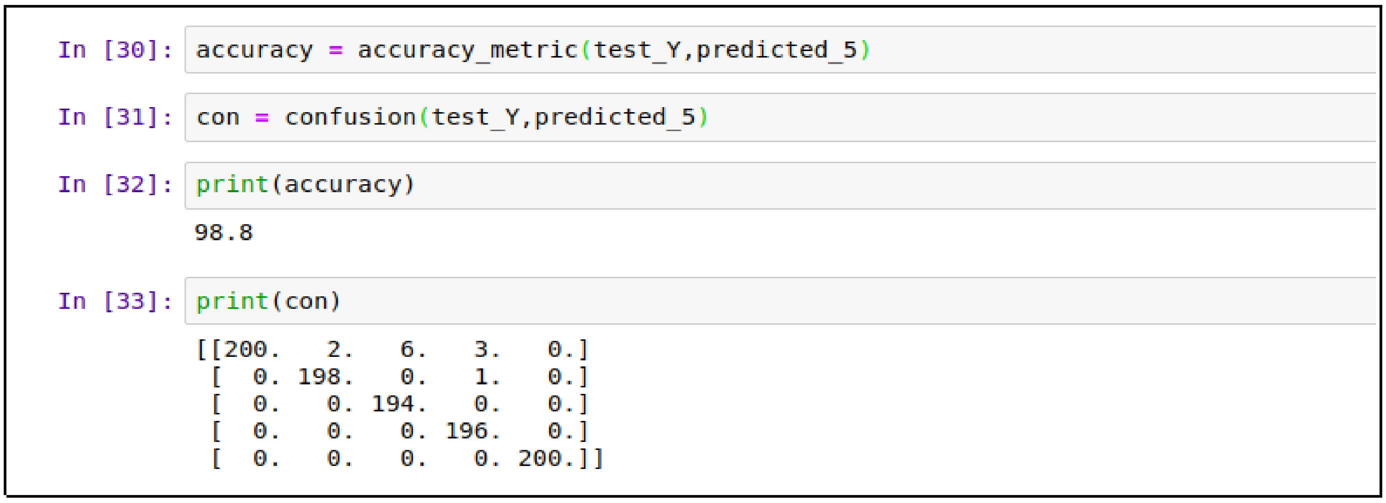
```
In [22]: accuracy = accuracy_metric(test_Y,predicted_1)

In [23]: con = confusion(test_Y,predicted_1)

In [24]: print(accuracy)

         98.8

In [25]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

3NN



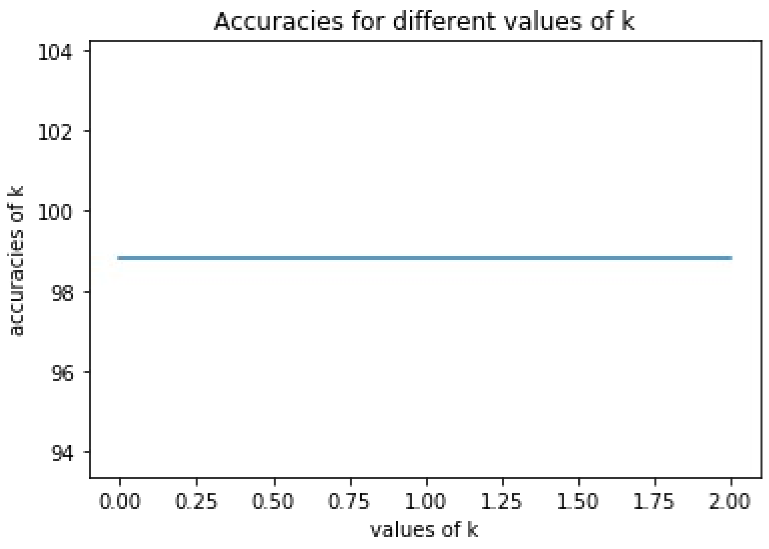
```
In [26]: accuracy = accuracy_metric(test_Y,predicted_3)

In [27]: con = confusion(test_Y,predicted_3)

In [28]: print(accuracy)

         98.8

In [29]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

5NN



```
In [30]: accuracy = accuracy_metric(test_Y,predicted_5)

In [31]: con = confusion(test_Y,predicted_5)

In [32]: print(accuracy)

         98.8

In [33]: print(con)

         [[200.   2.   6.   3.   0.]
          [  0. 198.   0.   1.   0.]
          [  0.   0. 194.   0.   0.]
          [  0.   0.   0. 196.   0.]
          [  0.   0.   0.   0. 200.]]
```

Graph between values of k and accuracies:

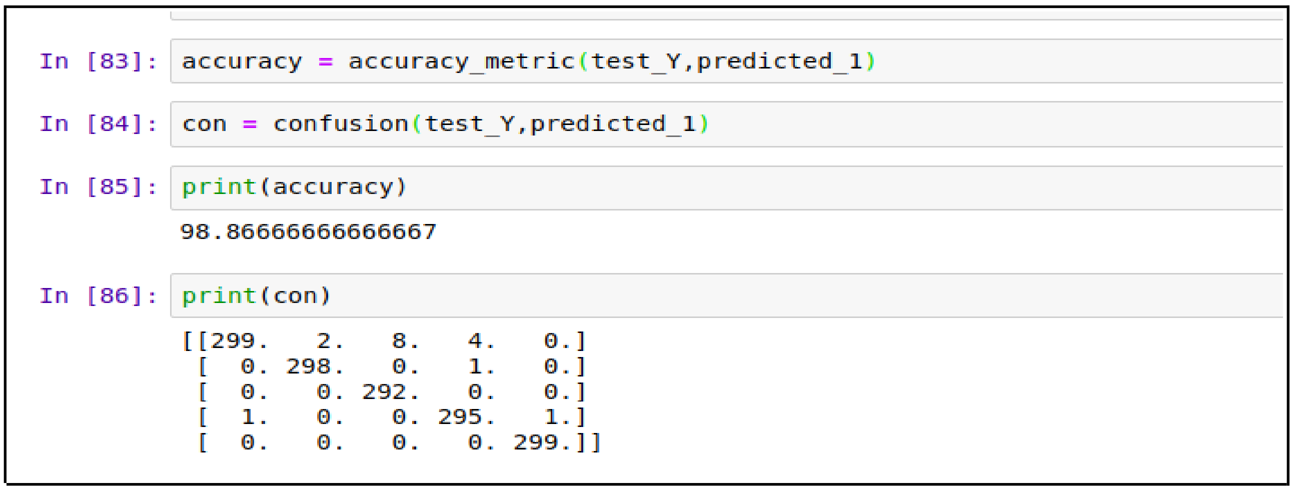


Inferences:

- In both the cases for all values of k accuracies are same. Thus model is well trained in both cases as train data is large. Thus training model is well generalised.
- Moreover, in both the cases among all three misclassified classes, rec.sport.hockey is being misclassified most of the times.
- It may be possible that terms are not unique enough which results low tf-idf values.
- Another case may be due to the randomness in train-test split ratio train data for this class which consists of non generalised documents.

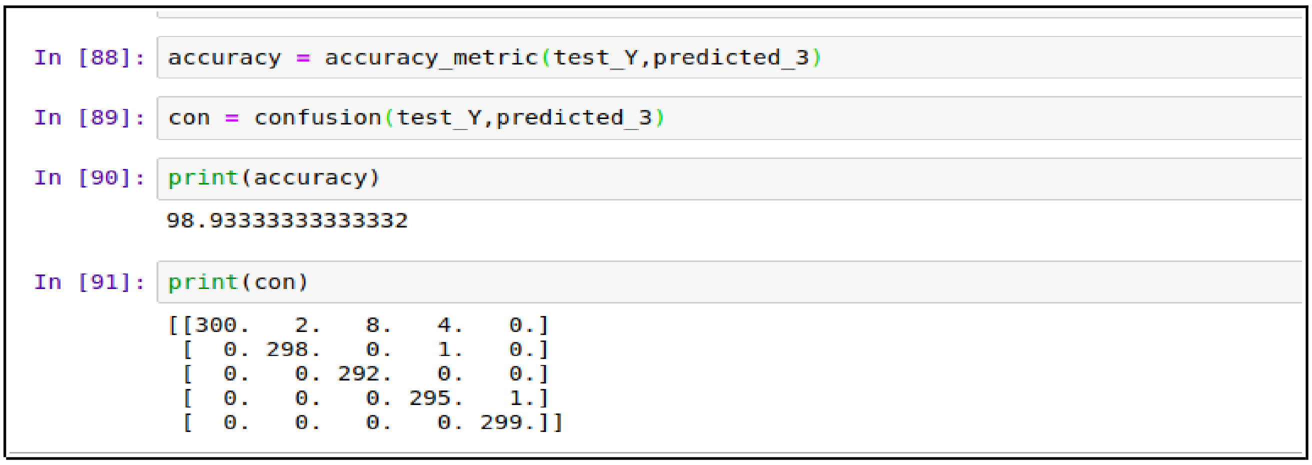
split ratio: 70-30
percentange of K – 0.7 %

1NN



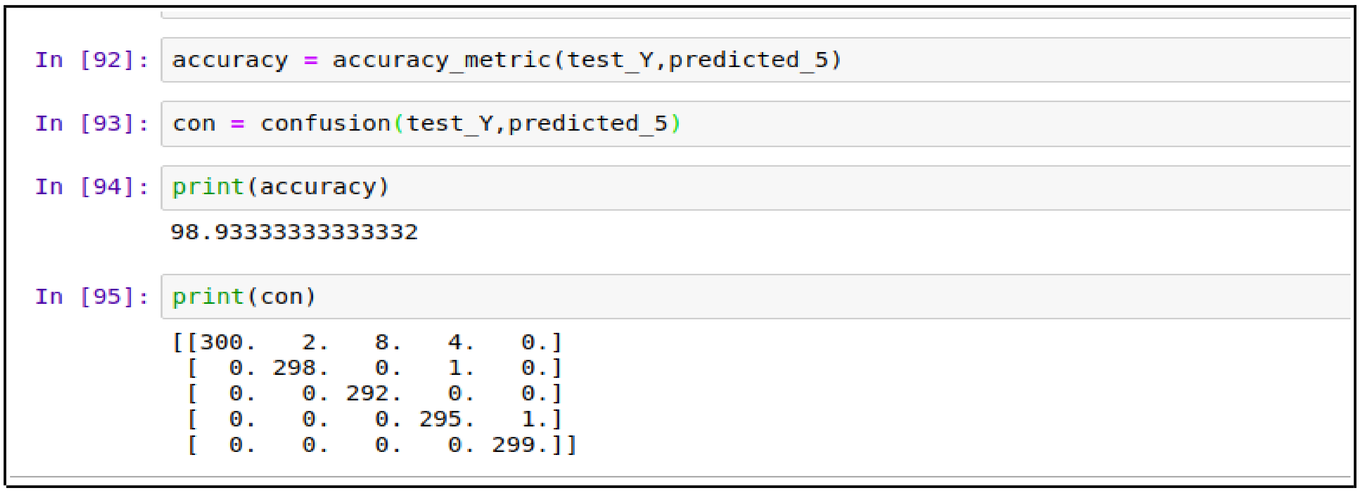
```
In [83]: accuracy = accuracy_metric(test_Y,predicted_1)

In [84]: con = confusion(test_Y,predicted_1)

In [85]: print(accuracy)

         98.86666666666667

In [86]: print(con)

         [[299.   2.   8.   4.   0.]
          [  0. 298.   0.   1.   0.]
          [  0.   0. 292.   0.   0.]
          [  1.   0.   0. 295.   1.]
          [  0.   0.   0.   0. 299.]]
```

3NN



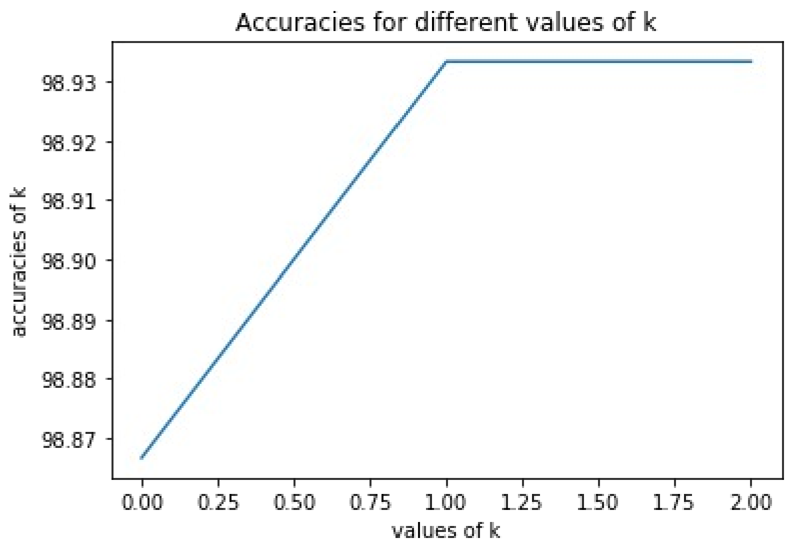
```
In [88]: accuracy = accuracy_metric(test_Y,predicted_3)

In [89]: con = confusion(test_Y,predicted_3)

In [90]: print(accuracy)

         98.93333333333332

In [91]: print(con)

         [[300.   2.   8.   4.   0.]
          [  0. 298.   0.   1.   0.]
          [  0.   0. 292.   0.   0.]
          [  0.   0.   0. 295.   1.]
          [  0.   0.   0.   0. 299.]]
```

5NN



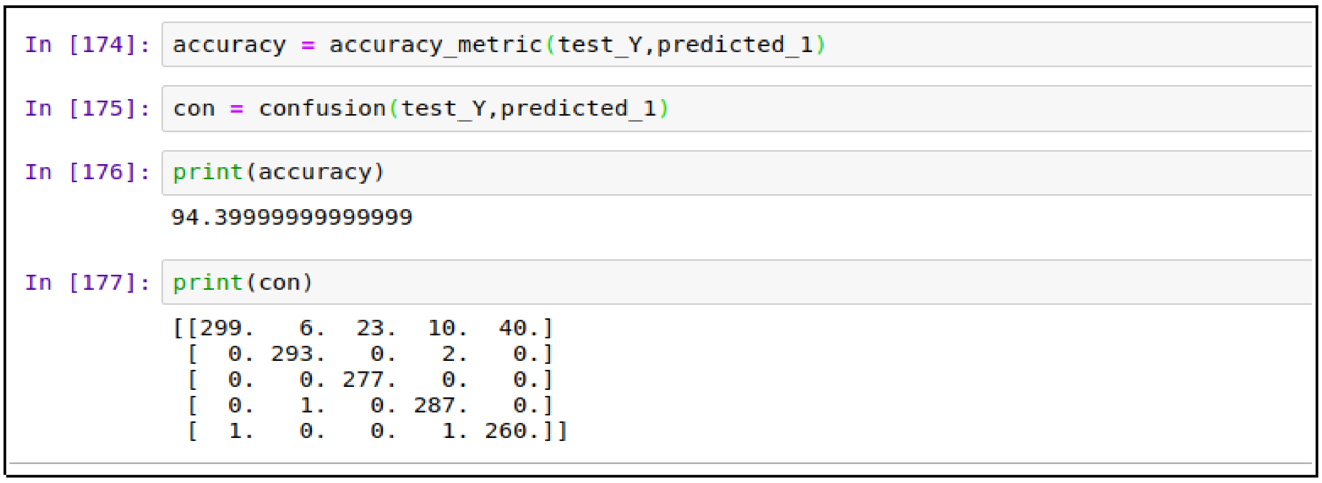
```
In [92]: accuracy = accuracy_metric(test_Y,predicted_5)

In [93]: con = confusion(test_Y,predicted_5)

In [94]: print(accuracy)

         98.93333333333332

In [95]: print(con)

         [[300.   2.   8.   4.   0.]
          [  0. 298.   0.   1.   0.]
          [  0.   0. 292.   0.   0.]
          [  0.   0.   0. 295.   1.]
          [  0.   0.   0.   0. 299.]]
```

Graph between values of k and accuracies:



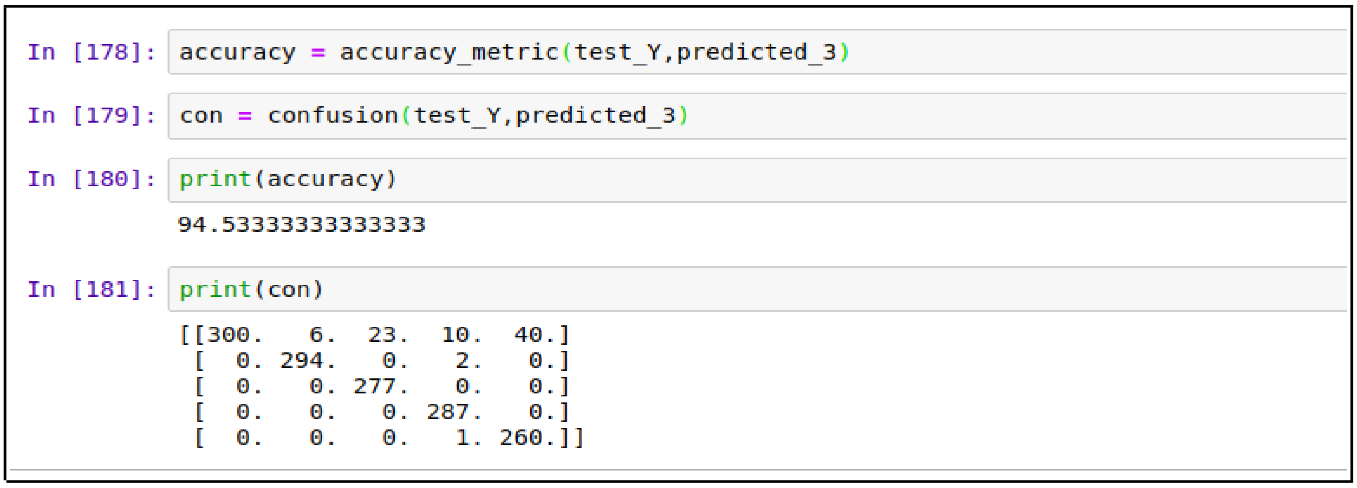
split ratio 70:30
valuek -40%

1NN



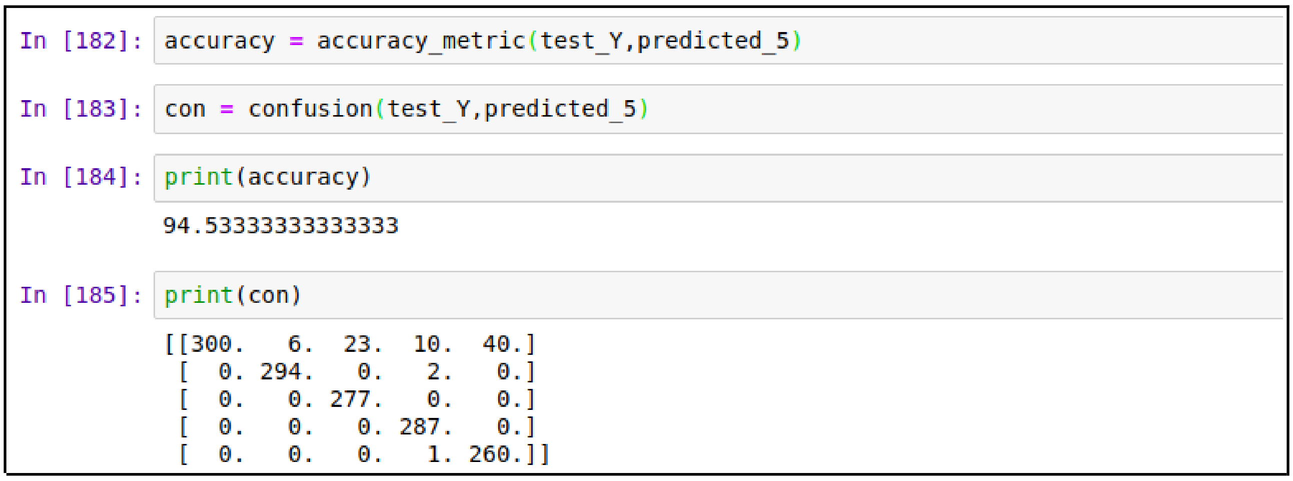
```
In [174]: accuracy = accuracy_metric(test_Y,predicted_1)

In [175]: con = confusion(test_Y,predicted_1)

In [176]: print(accuracy)

          94.39999999999999

In [177]: print(con)

          [[299.   6.  23.  10.  40.]
           [  0. 293.   0.   2.   0.]
           [  0.   0. 277.   0.   0.]
           [  0.   1.   0. 287.   0.]
           [  1.   0.   0.   1. 260.]]
```

3NN



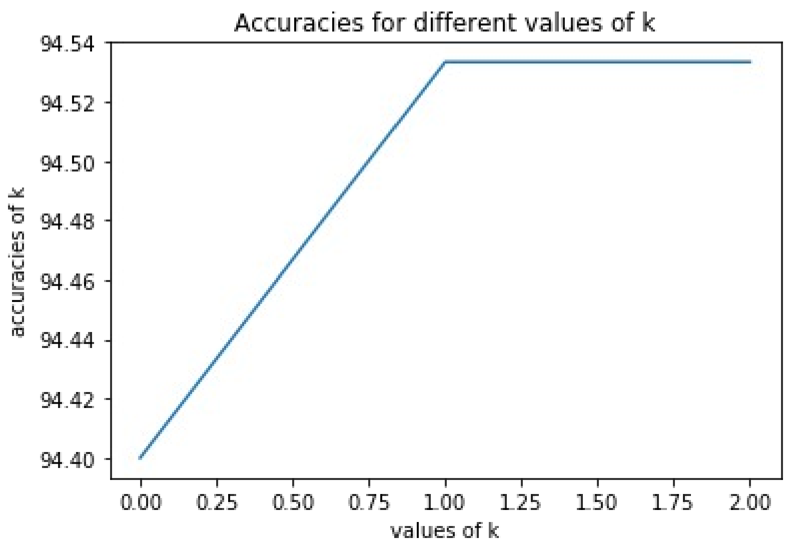
```
In [178]: accuracy = accuracy_metric(test_Y,predicted_3)

In [179]: con = confusion(test_Y,predicted_3)

In [180]: print(accuracy)

          94.53333333333333

In [181]: print(con)

          [[300.   6.  23.  10.  40.]
           [  0. 294.   0.   2.   0.]
           [  0.   0. 277.   0.   0.]
           [  0.   0.   0. 287.   0.]
           [  0.   0.   0.   1. 260.]]
```

5NN



```
In [182]: accuracy = accuracy_metric(test_Y,predicted_5)

In [183]: con = confusion(test_Y,predicted_5)

In [184]: print(accuracy)

          94.53333333333333

In [185]: print(con)

          [[300.   6.  23.  10.  40.]
           [  0. 294.   0.   2.   0.]
           [  0.   0. 277.   0.   0.]
           [  0.   0.   0. 287.   0.]
           [  0.   0.   0.   1. 260.]]
```

Graph between values of k and accuracies:

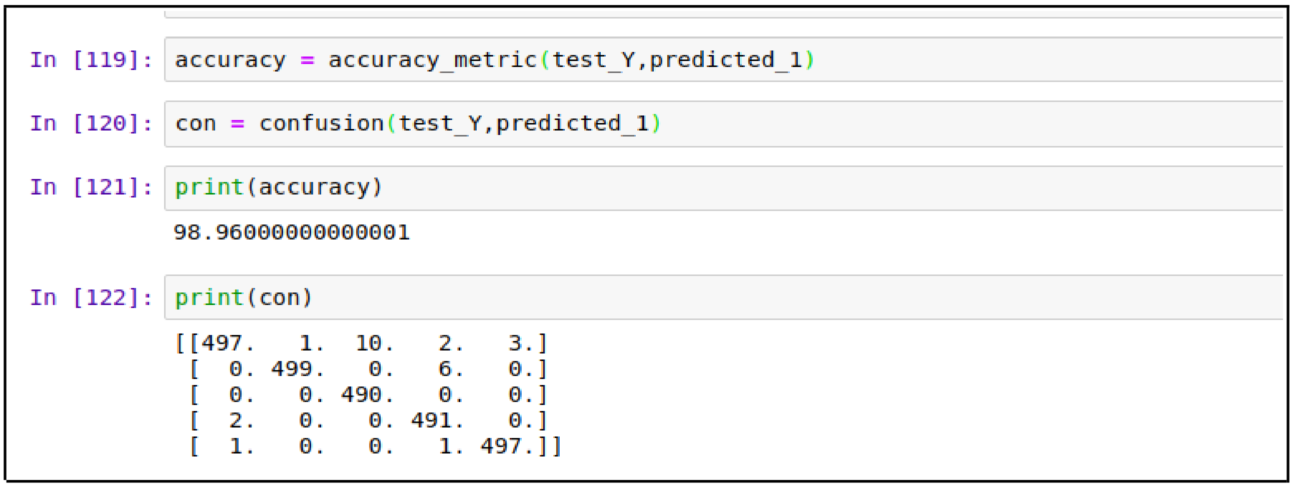


Inferences:

- It can be seen that in both the cases 3nn and 5nn are values are same and more than 1nn. As the reason stated above it’s robustness of large value of k.
- It is also observed that in case when 0.4% of data is selected the accuracy decreases.
- It may be possible that some of the unique words which are present in the test document are absent in train data after feature selection has been done.
- And misclassification rate also increase in 0.4% which can be observed using confusion matrix.

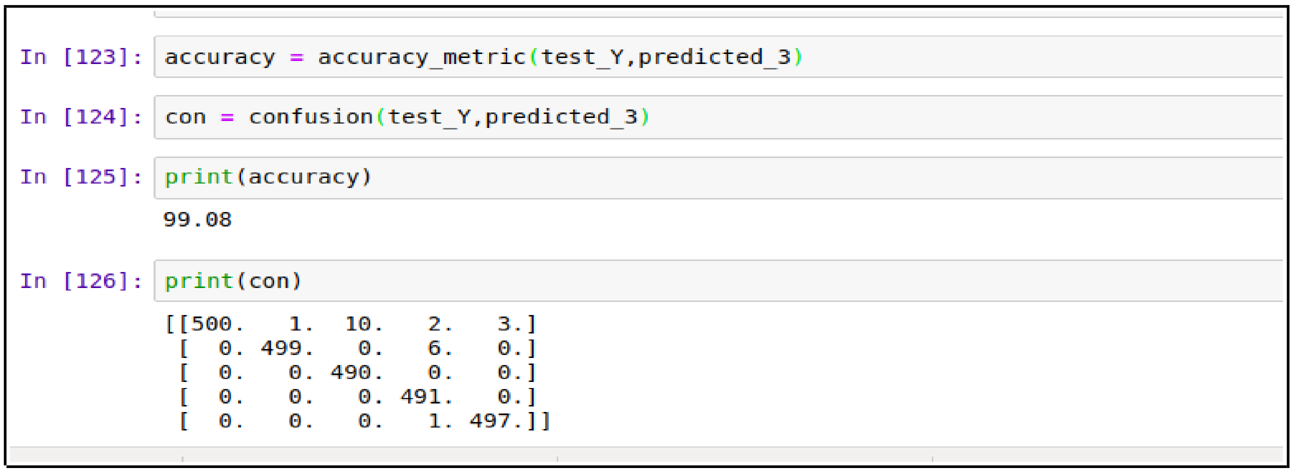
split ratio – 50:50
values of k – 0.7 %

1NN



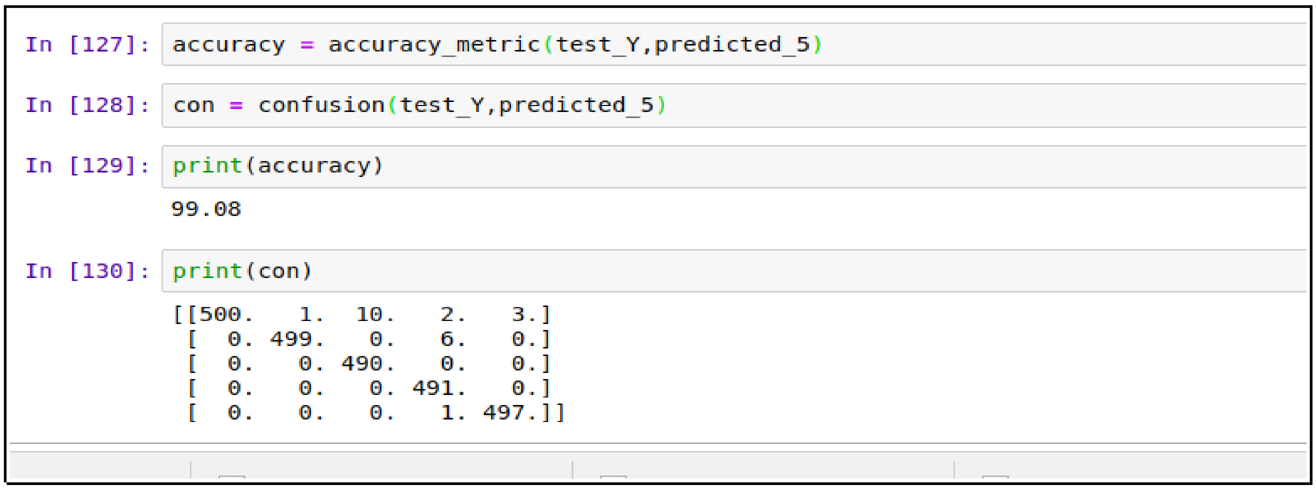
```
In [119]: accuracy = accuracy_metric(test_Y,predicted_1)

In [120]: con = confusion(test_Y,predicted_1)

In [121]: print(accuracy)

          98.96000000000001

In [122]: print(con)

          [[497.   1.  10.   2.   3.]
           [  0. 499.   0.   6.   0.]
           [  0.   0. 490.   0.   0.]
           [  2.   0.   0. 491.   0.]
           [  1.   0.   0.   1. 497.]]
```

3NN



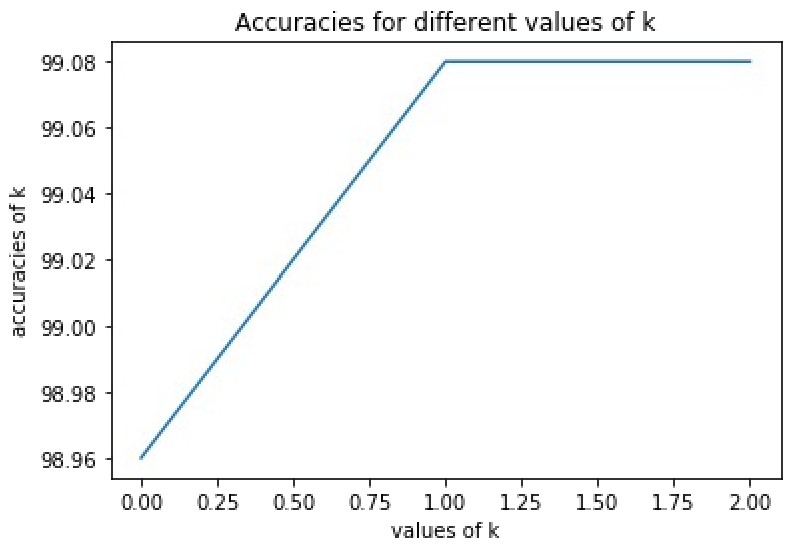
```
In [123]: accuracy = accuracy_metric(test_Y,predicted_3)

In [124]: con = confusion(test_Y,predicted_3)

In [125]: print(accuracy)

          99.08

In [126]: print(con)

          [[500.   1.  10.   2.   3.]
           [  0. 499.   0.   6.   0.]
           [  0.   0. 490.   0.   0.]
           [  0.   0.   0. 491.   0.]
           [  0.   0.   0.   1. 497.]]
```

5NN



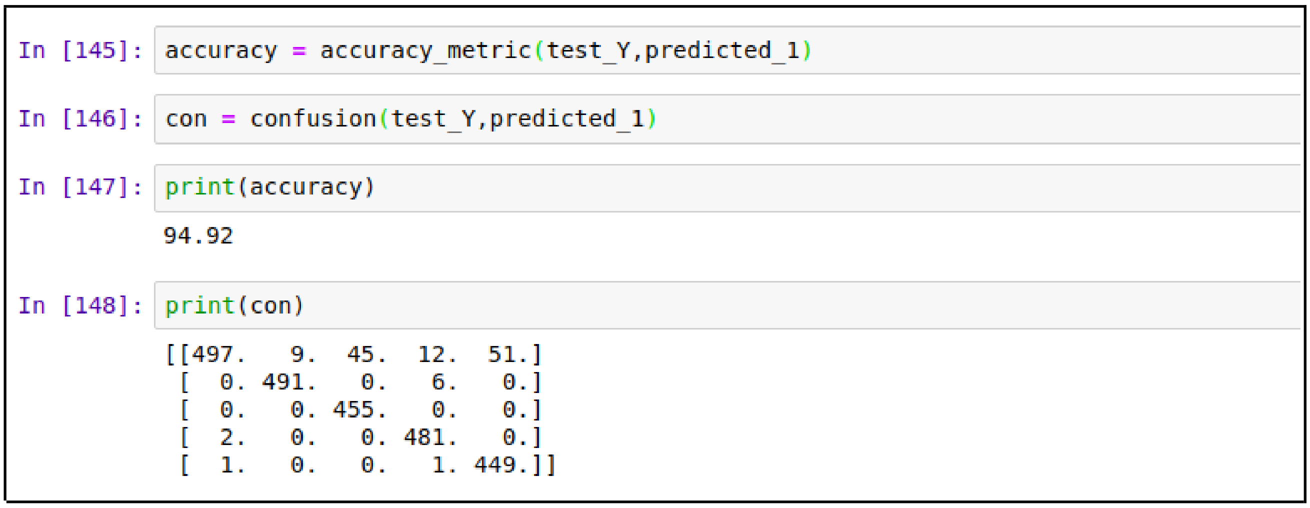
```
In [127]: accuracy = accuracy_metric(test_Y,predicted_5)

In [128]: con = confusion(test_Y,predicted_5)

In [129]: print(accuracy)

          99.08

In [130]: print(con)

          [[500.   1.  10.   2.   3.]
           [  0. 499.   0.   6.   0.]
           [  0.   0. 490.   0.   0.]
           [  0.   0.   0. 491.   0.]
           [  0.   0.   0.   1. 497.]]
```

Graph between values of k and accuracies:



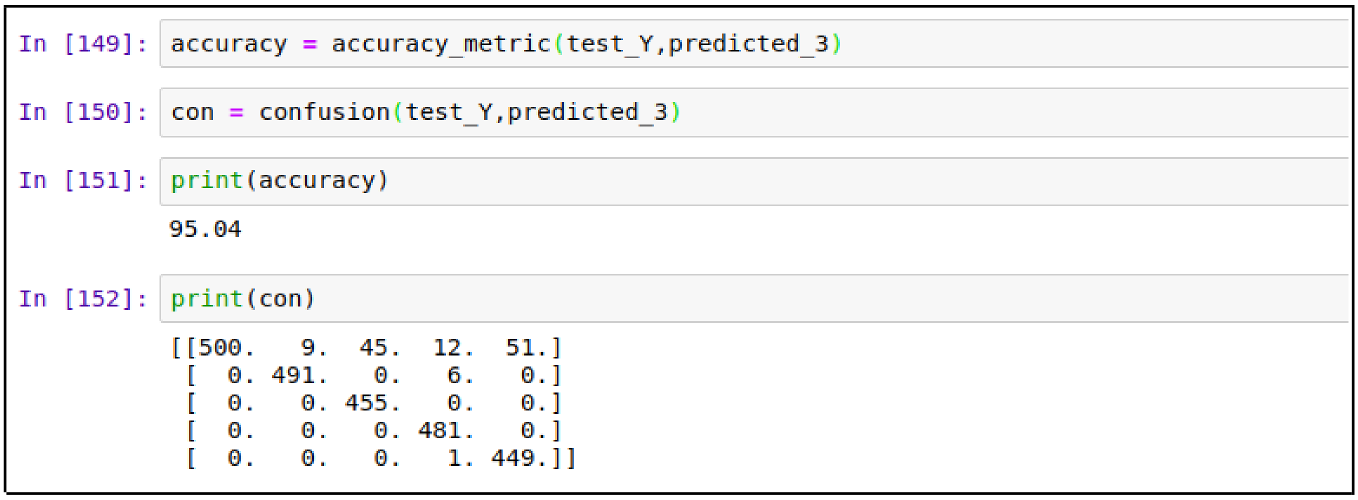
split ratio-50:50
value of k – 40%

1NN



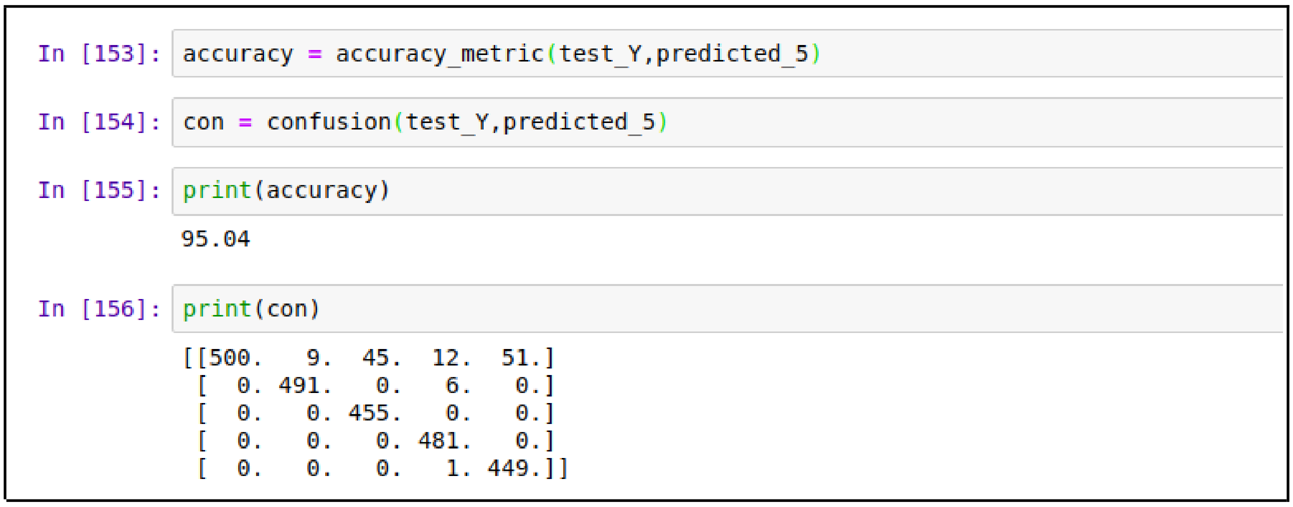
```
In [145]: accuracy = accuracy_metric(test_Y,predicted_1)

In [146]: con = confusion(test_Y,predicted_1)

In [147]: print(accuracy)

          94.92

In [148]: print(con)

          [[497.   9.  45.  12.  51.]
           [  0. 491.   0.   6.   0.]
           [  0.   0. 455.   0.   0.]
           [  2.   0.   0. 481.   0.]
           [  1.   0.   0.   1. 449.]]
```

3NN



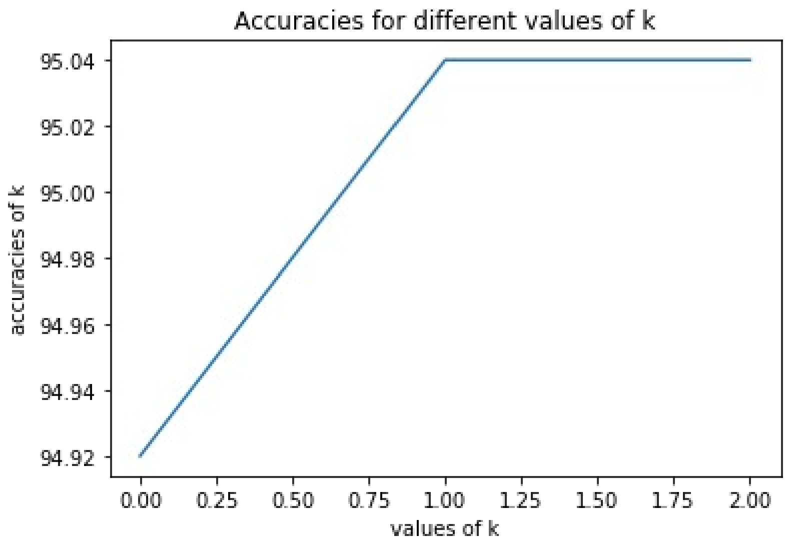
```
In [149]: accuracy = accuracy_metric(test_Y,predicted_3)

In [150]: con = confusion(test_Y,predicted_3)

In [151]: print(accuracy)

          95.04

In [152]: print(con)

          [[500.   9.  45.  12.  51.]
           [  0. 491.   0.   6.   0.]
           [  0.   0. 455.   0.   0.]
           [  0.   0.   0. 481.   0.]
           [  0.   0.   0.   1. 449.]]
```

5NN



```
In [153]: accuracy = accuracy_metric(test_Y,predicted_5)

In [154]: con = confusion(test_Y,predicted_5)

In [155]: print(accuracy)

          95.04

In [156]: print(con)

          [[500.   9.  45.  12.  51.]
           [  0. 491.   0.   6.   0.]
           [  0.   0. 455.   0.   0.]
           [  0.   0.   0. 481.   0.]
           [  0.   0.   0.   1. 449.]]
```

Graph between values of k and accuracies:



Inferences:

- Shows the same trend as in 70:30 split. In this cse 0.4% has less accuracy compared to 0.7%.
- In this case strangely, 1nn works best in first case. It is may be due to low bias and in close proximity to train data. Training error is zero in this case.
- Another reason may be due random shuffling of data.
- But in case top 0.4% values 3nn and 5nn works best and results in low variance.

Conclusion made on both the techniques:

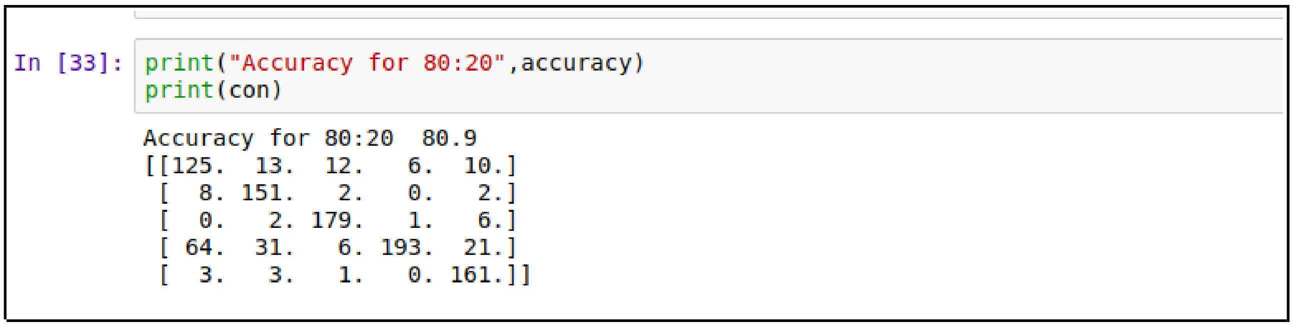
- Increase in train test split, increases the processing time in both the cases.
- Generally, with increase in percentage of top K values, accuracy increases.
- KNN performs more computational time on test data in comparison with train data. The training phase only consists of storing the feature vector and class label.
- The training time for any value of k is same in same split.
- Performing feature selection is important in case of kNN because kNN likely to overfit model.
- The classification accuracy tends to increase with large value of k in kNN as the boundary becomes more smoother.
- Although by increase k in kNN accuracy should decrease. Since smaller subset is checked here, the accuracy is not decreasing if we choose value of k in between 5-200. But after the value of k is more than 200 the accuracy tends to decreases.
- Both techniques MI and TF-IDF gives almost same performance. Even though TFIDF is text task-free i.e it does use the class category information present in training set but MI is task-dependent.
- TFIDF is basically a adhoc approach of selecting features from the corpus. Even though it’s found to be reliable.
- kNN uses distance metricss, which becomes meaningless, when the dimension of the data increases significantly. And sometimes incapable to handle more number of features due to confusion between different classes.
- Hence in both the above technique kNN classifier works almost same. Thus it is suggested that kNN works better with less number of features.

MI Naive Bayes

1. Probability formula used:

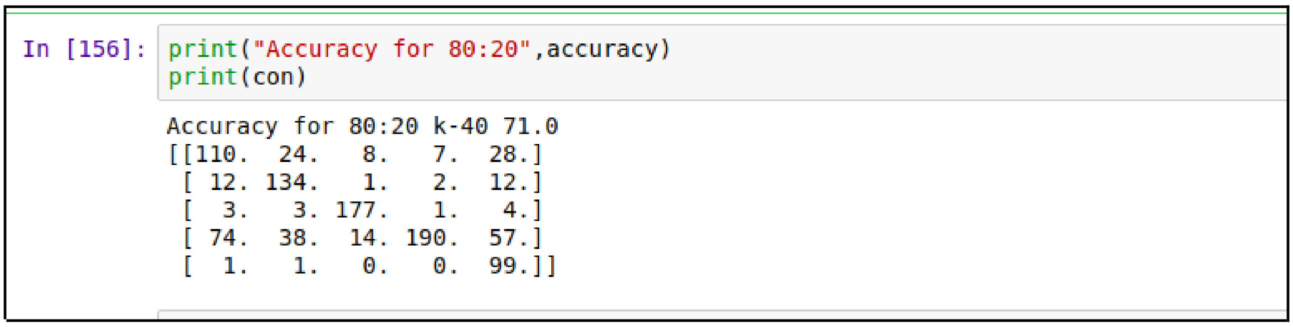
$$\boxed{P(c|d) \propto P(c)\prod_{k=1}^{n_d} P(t_k|c)}$$

split ratio-80:20
value of k -0.7%



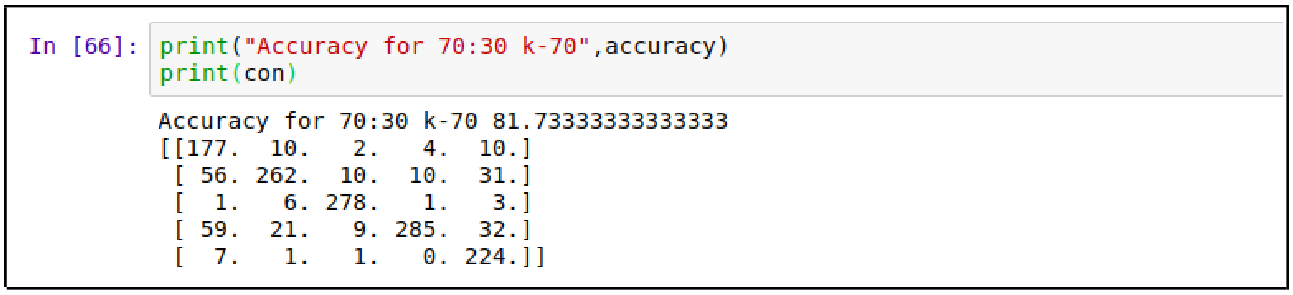
```
In [33]: print("Accuracy for 80:20",accuracy)
         print(con)

         Accuracy for 80:20  80.9
         [[125.  13.  12.   6.  10.]
          [  8. 151.   2.   0.   2.]
          [  0.   2. 179.   1.   6.]
          [ 64.  31.   6. 193.  21.]
          [  3.   3.   1.   0. 161.]]
```

split ratio 80:20
value of k- 0.4%



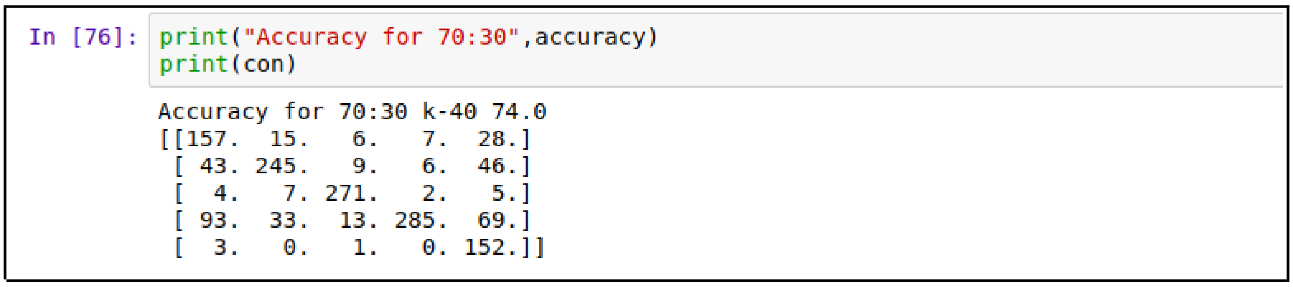
```
In [156]: print("Accuracy for 80:20",accuracy)
          print(con)

          Accuracy for 80:20 k-40 71.0
          [[110.  24.   8.   7.  28.]
           [ 12. 134.   1.   2.  12.]
           [  3.   3. 177.   1.   4.]
           [ 74.  38.  14. 190.  57.]
           [  1.   1.   0.   0.  99.]]
```

split ratio 70:30
value of k-0.7%



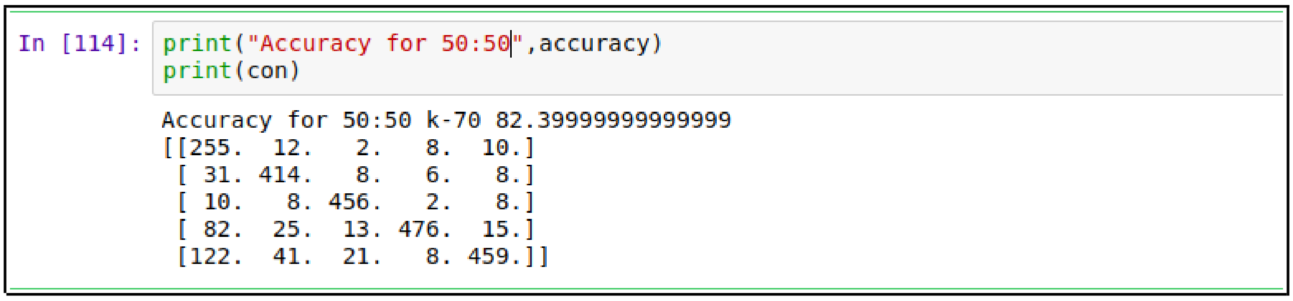
```
In [66]: print("Accuracy for 70:30 k-70",accuracy)
         print(con)

         Accuracy for 70:30 k-70 81.73333333333333
         [[177.  10.   2.   4.  10.]
          [ 56. 262.  10.  10.  31.]
          [  1.   6. 278.   1.   3.]
          [ 59.  21.   9. 285.  32.]
          [  7.   1.   1.   0. 224.]]
```

split ratio 70:30
value of k – 0.4%



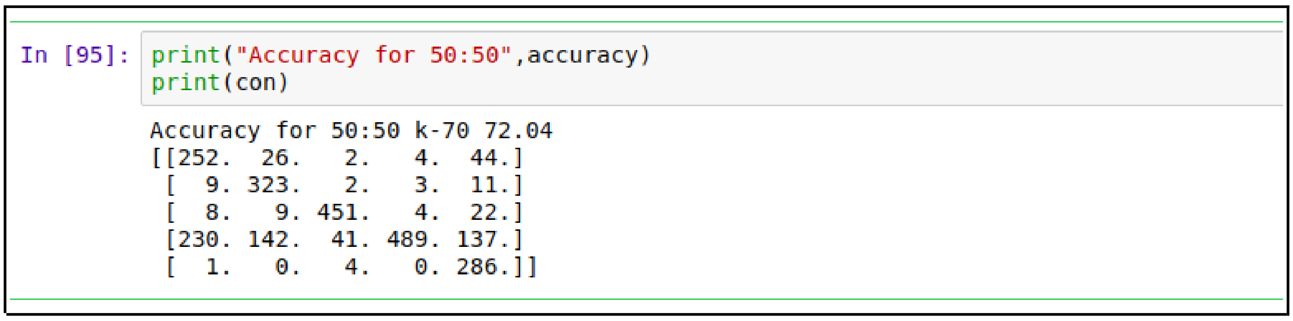
```
In [76]: print("Accuracy for 70:30",accuracy)
         print(con)

         Accuracy for 70:30 k-40 74.0
         [[157.  15.   6.   7.  28.]
          [ 43. 245.   9.   6.  46.]
          [  4.   7. 271.   2.   5.]
          [ 93.  33.  13. 285.  69.]
          [  3.   0.   1.   0. 152.]]
```

split ratio 50:50
value of k-0.7%



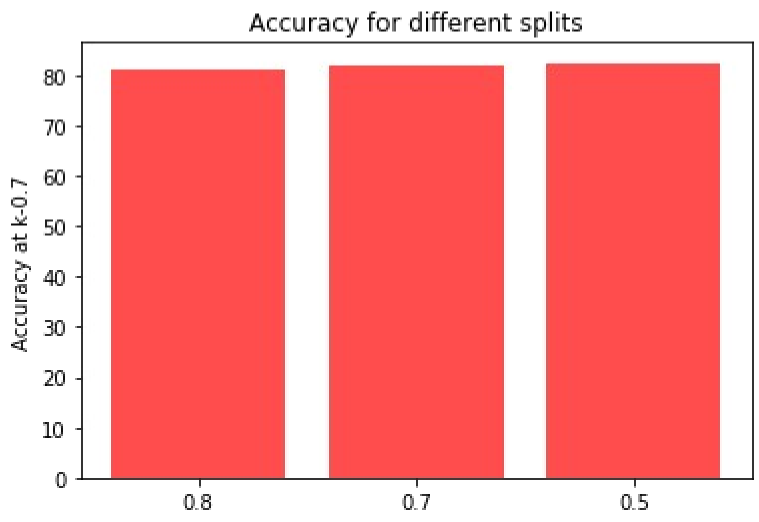
```
In [114]: print("Accuracy for 50:50",accuracy)
          print(con)

          Accuracy for 50:50 k-70 82.39999999999999
          [[255.  12.   2.   8.  10.]
           [ 31. 414.   8.   6.   8.]
           [ 10.   8. 456.   2.   8.]
           [ 82.  25.  13. 476.  15.]
           [122.  41.  21.   8. 459.]]
```

split ratio 50:50
value of k-0.4%



```
In [95]: print("Accuracy for 50:50",accuracy)
         print(con)

         Accuracy for 50:50 k-70 72.04
         [[252.  26.   2.   4.  44.]
          [  9. 323.   2.   3.  11.]
          [  8.   9. 451.   4.  22.]
          [230. 142.  41. 489. 137.]
          [  1.   0.   4.   0. 286.]]
```

Graph of accuracy vs split ratio.



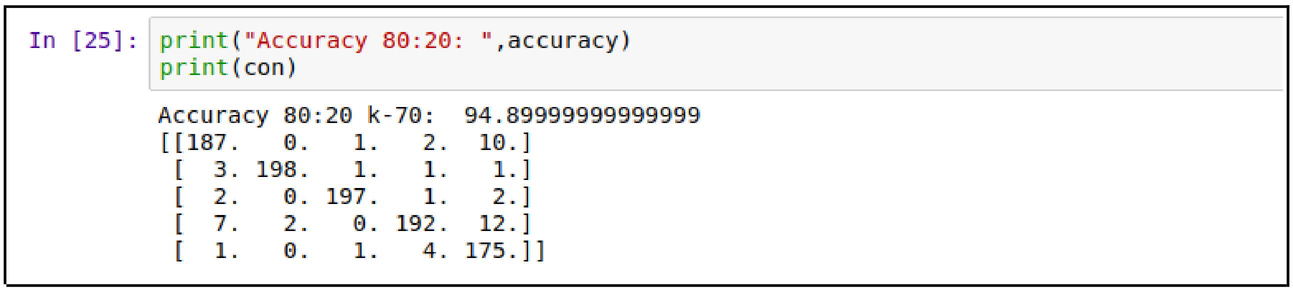
Inferences:

- Accuracies are almost same in all 3 splits.

- The low accuracy in 0.4% may be due to presence of weak indicator (terms are less informative for a particular class).
- Also, It may be possible that terms are not informative and as relevant as compared to others.
- Accuracy in both 80:20 and 70:30 is less due to presence of noisy words or may be due randomness in the data.
- 50:50 split was getting better accuracy may be because of the data shuffling before splitting the data.
- As we increase the train data there's also a chance of getting noise into the dataset.
- When we consider top 0.4% of values in all splits, last class is being misclassified most of the times. It may be due to the presence of less informative terms in that class.

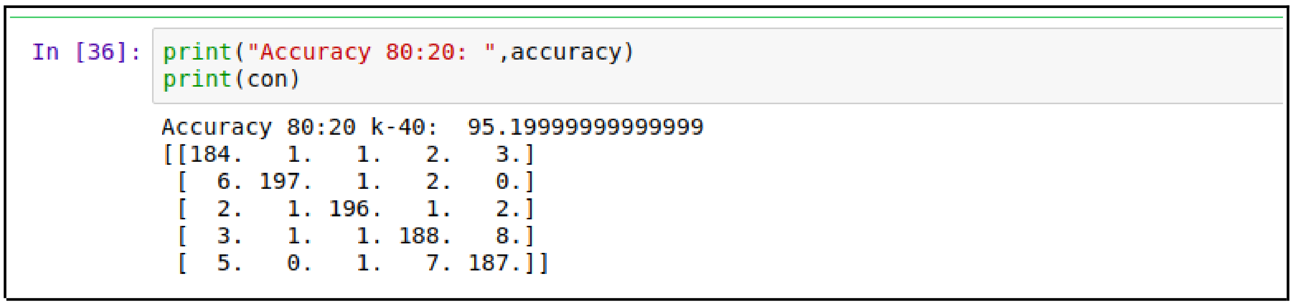
TF-IDF Naive Bayes

split ratio 80:20
value of k-0.7%



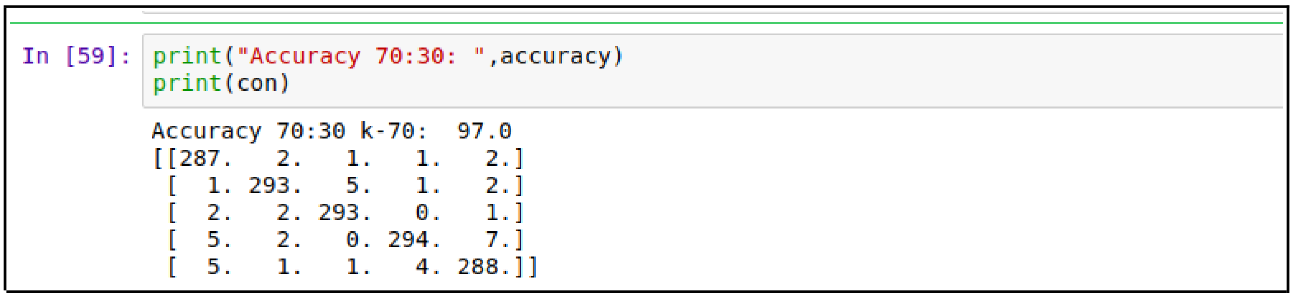
```
In [25]: print("Accuracy 80:20: ",accuracy)
         print(con)

         Accuracy 80:20 k-70:  94.89999999999999
         [[187.   0.   1.   2.  10.]
          [  3. 198.   1.   1.   1.]
          [  2.   0. 197.   1.   2.]
          [  7.   2.   0. 192.  12.]
          [  1.   0.   1.   4. 175.]]
```

split ratio 80:20
value of k-0.4%



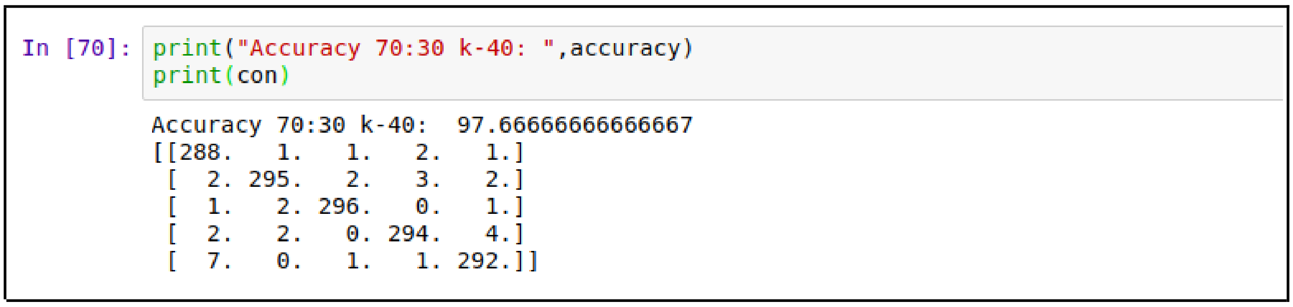
```
In [36]: print("Accuracy 80:20: ",accuracy)
         print(con)

         Accuracy 80:20 k-40:  95.19999999999999
         [[184.   1.   1.   2.   3.]
          [  6. 197.   1.   2.   0.]
          [  2.   1. 196.   1.   2.]
          [  3.   1.   1. 188.   8.]
          [  5.   0.   1.   7. 187.]]
```

split ratio 70:30
value of k-0.7%



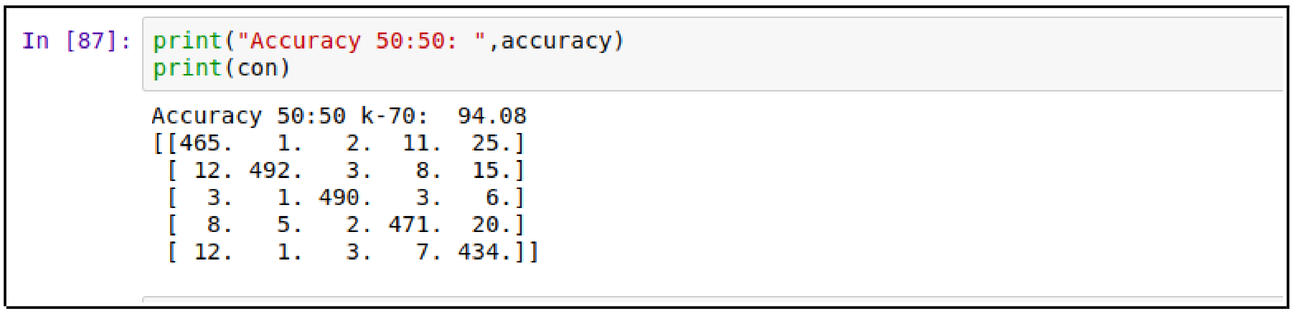
```
In [59]: print("Accuracy 70:30: ",accuracy)
         print(con)

         Accuracy 70:30 k-70:  97.0
         [[287.   2.   1.   1.   2.]
          [  1. 293.   5.   1.   2.]
          [  2.   2. 293.   0.   1.]
          [  5.   2.   0. 294.   7.]
          [  5.   1.   1.   4. 288.]]
```

split ratio 70:30
value of k-0.4%



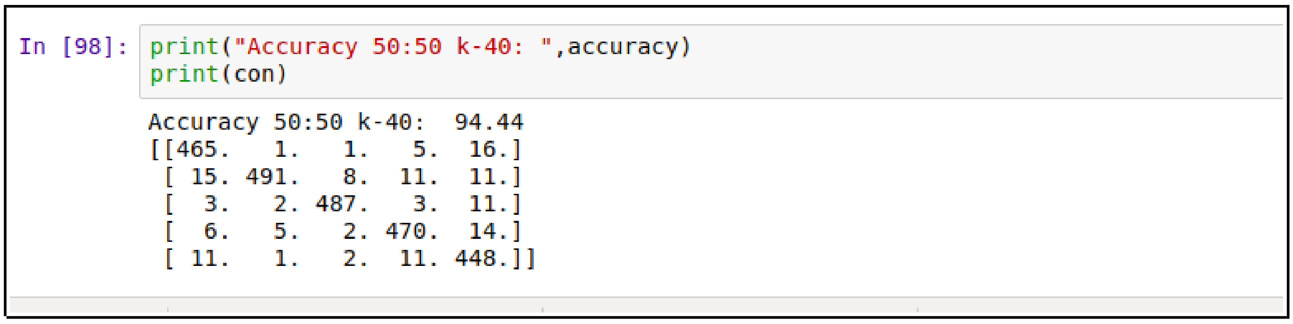
```
In [70]: print("Accuracy 70:30 k-40: ",accuracy)
         print(con)

         Accuracy 70:30 k-40:  97.66666666666667
         [[288.   1.   1.   2.   1.]
          [  2. 295.   2.   3.   2.]
          [  1.   2. 296.   0.   1.]
          [  2.   2.   0. 294.   4.]
          [  7.   0.   1.   1. 292.]]
```

split ratio 50:50
value of k-0.7%



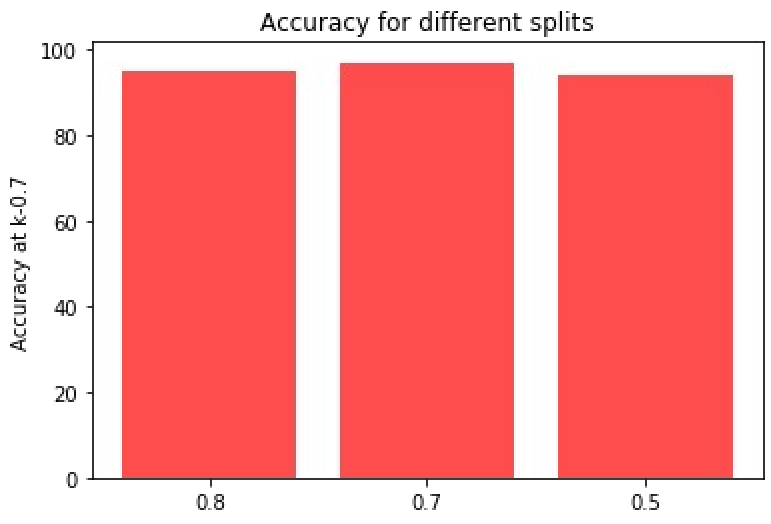
```
In [87]: print("Accuracy 50:50: ",accuracy)
         print(con)

         Accuracy 50:50 k-70:  94.08
         [[465.   1.   2.  11.  25.]
          [ 12. 492.   3.   8.  15.]
          [  3.   1. 490.   3.   6.]
          [  8.   5.   2. 471.  20.]
          [ 12.   1.   3.   7. 434.]]
```

split ratio 50:50
value of k-0.4%



```
In [98]: print("Accuracy 50:50 k-40: ",accuracy)
         print(con)

         Accuracy 50:50 k-40:  94.44
         [[465.   1.   1.   5.  16.]
          [ 15. 491.   8.  11.  11.]
          [  3.   2. 487.   3.  11.]
          [  6.   5.   2. 470.  14.]
          [ 11.   1.   2.  11. 448.]]
```

Graph of accuracies vs splits



Inferences:

- Accuracy is nearly same in all 3 splits.
- 70:30 is considered as best split ratio in machine learning, thus it gives high accuracy. And due to low generalisation and less amount of data in 50-50 split the accuracy is slightly low.
- Since accuracy increases with increase in features in case of TF-IDF.
- In all the splits the last class is misclassfied most of the times. This is due to the absence of unqiue terms which makes IDF value as zero or low.
- And it may be possible due to independence assumption of Naive Bayes.

Conclusions made on both the techniques:

- In my experiment, TF-IDF performace is better than MI.
- In TF-IDF accuracy increase with increase in data and also depend upon the occurences of the term in the entire corpus but in case of MI the accuracy reaches maximum when a particular term is a perfect indicator for class membership.
- As we increase the split ratio the processing time also increases.
- Accuracy will also tend to increase with increase in split ratio but increase in accuracy is not too certain.
- However, in both the techniques we can see one similarity, class 4 is being misclassified the most in all the results achieved so far.
- It may be possible due to the absence of non unique terms in this class.
- The performance of Naive Bayes may degrade if the data contains highly correlated features.
- The misclassfication reason in Naive Bayes is due to it’s strong assuption of independence.
- In Naive Bayes TF-IDF outperforms MI mentioned in Discrimination-Based Feature Selection for Multinomial Naive Bayes Text Classification research paper.
- Feature selection with naive bayes results in better accuracy since it gives only important feature values.
- Naive bayes doesn’t require a huge amount of data for classification, it just requires the correct set of words from the feature selection.

Note: Accuracies may vary according to shuffling and randomness of data.