K-NN Classification and Condensed 1NN algorithm

Machine Learning Assignment

This algorithm falls in category of supervised machine learning algorithm. KNN is very useful in text processing and is very simple to implement. However, it is expensive in terms of time and space. The report shows result statistics for different values of k, number of training samples.

K-NN Classification and Condensed 1NN algorithm

Machine Learning Assignment

K-NN Algorithm

Here, implementation of k-NN algorithm is specific to the Letter Recognition data from UCI Machine Learning Repository. There are 20000 samples for letter recognition with each sample having a label and 16 dimension to it.

Data formatting

Given dataset of Letter Recognition has identified label as a class and remaining 16 attributes as dimensions for K-NN algorithm. Every time the experiment is carried, the data set is split into training and testing data. Testing data being 5000 throughout the experiment we change the training data from 100 to 15000. Data is divided into 4 matrices viz.

trainX - #of training samples X 16 → Containing 16 dimensions

trainY - #of training samples X 1 → Containing the labels

testX - #of testing samples X 16 → Containing 16 dimensions

result - #of testing samples X 1 → Containing labels.

Algorithm

1. Open data file.

2. Shuffle file data so that we pick random test and train samples.

3. Slice data using 'islice' in training and testing data.

4. Make 4 matrices as mentioned above.

5. Call testknn function with following parameters

Nearest Neighbor

Algorithm works on nearest neighbor to find out label for the new data. Nearest neighbor is decided using the Euclidian distance between the new point and existing training data. Label of the one with minimum distance is assigned to the new data. This idea is generalized for k-nearest neighbors.

testY = testknn(trainX, trainY, testX, k)

6. Check the accuracy using the labels returned by K-NN algorithm against result matrix.

7. Calculate the elapsed time.

We will change the value of k as 1, 3, 5, 7, 9 for each training set of 100, 1000, 5000, 10000, 15000 samples. Accuracy and elapsed time is measured for each and every case. The results are shown in Results section.

Functions

Main function in this algorithm is 'testknn'. Function performs the following:

1. Calculate Euclidian trainX and testX:

This will result in matrix with rows = #of testing samples and columns = #of training samples. For example if we have 10 training samples and 2 testing samples. Matrix will have a dimension of 2x10 where row corresponds to the number of testing example.

M[i][j] will have a distance of ith testing sample from jth training sample

2. Collect the indices of top k (k minimum elements).

3. Collect labels for these indices.

4. Output the most frequent label as a classified nearest neighbor.

Results

Below are the results for different values of K and different number of training samples:

Assumption- The host system has 8GB of RAM and Intel core i5-5200 2.20GHz processor. Other programs running on system may affect computation time of algorithm.

Time Taken						
k/#Training Samples	100	1000	2000	5000	10000	15000
1	3.13	31.5	78.43	161.78	303.3	465.22
3	3.6	32.55	77.32	164.66	306.72	452.96
5	3.06	32.89	69.12	161.08	311.34	470.12
7	3.18	31.98	81.22	154.87	312.89	450.8
9	3.22	33.06	77.87	155.02	312.37	450.23

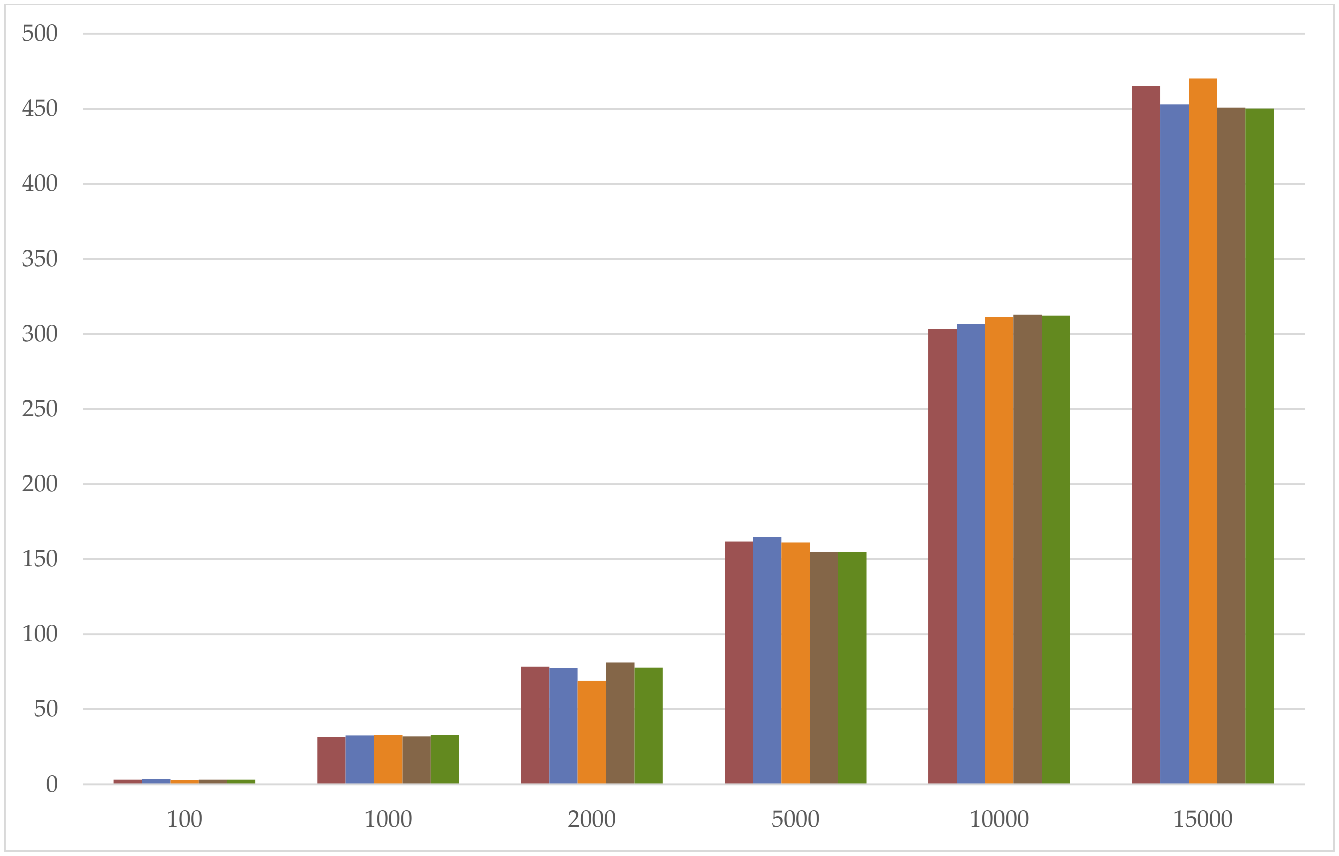

Fig 1: Graphical representation of Time taken by the K-NN algorithm for different training samples (X-Axis) and time taken in seconds (Y- Axis). Each color column represents the value for k.

Accuracy of K-NN algorithm:

Accuracy						
k/#Training Samples	100	1000	2000	5000	10000	15000
1	41.68	76.44	86.34	91.24	94.24	95.28
3	28.38	69.22	84.54	90.02	93.46	95.24
5	26.74	66.98	81.87	89.66	93.58	95.02
7	23.68	64.8	80.33	88.76	93.2	94.24
9	21.97	63.02	78.51	87.96	93.12	94.04

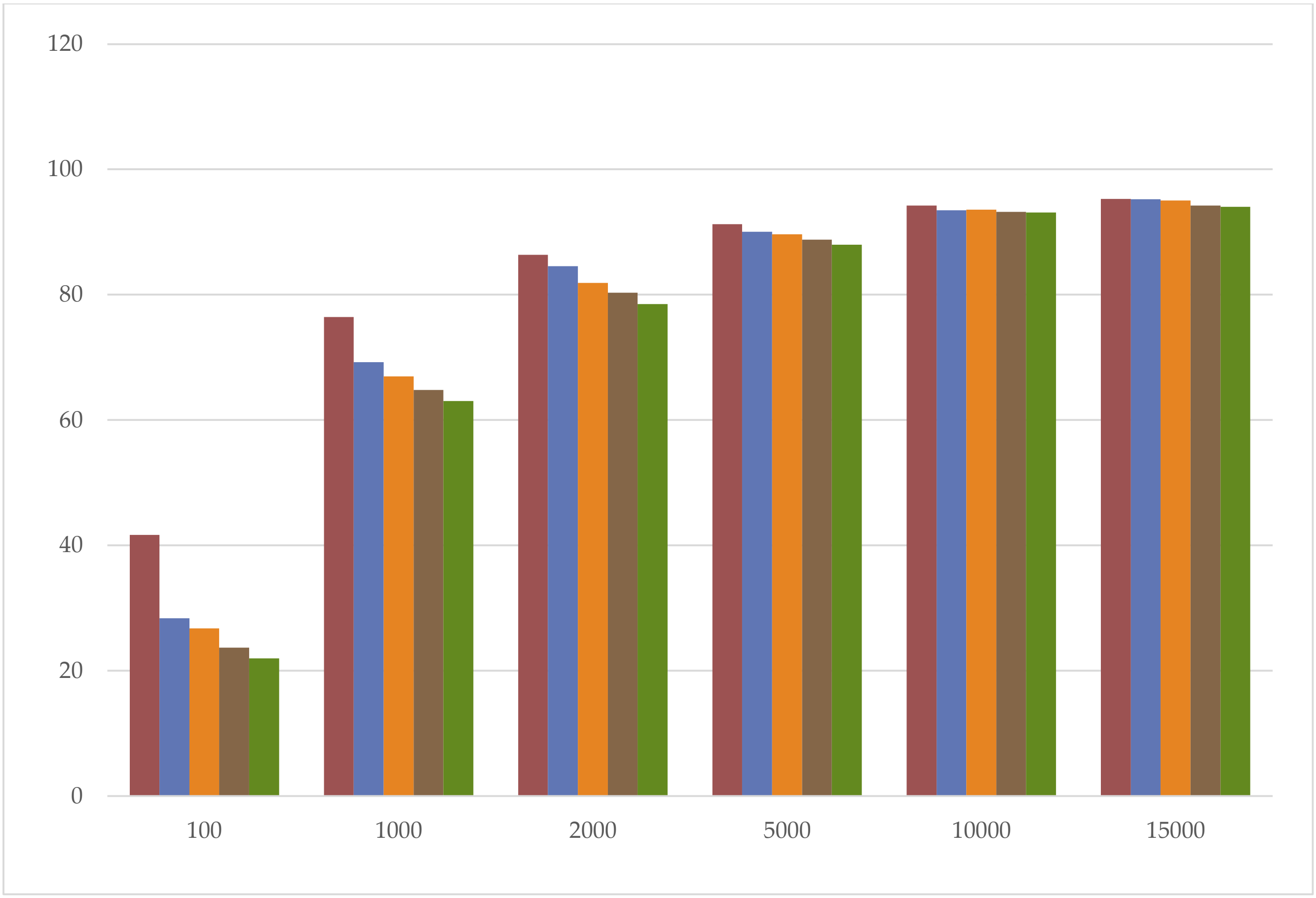

Fig 2: Graphical representation of accuracy by the K-NN algorithm for different training samples (X-Axis) and accuracy in % (Y- Axis). Each color column represents the value for k.

Conclusion

Summing up the results shown in the above section, we can say that algorithm efficiency increases with increase in training examples. Time taken by the algorithm depicts that how expensive it is in terms of computations.

Condensed K-NN algorithm

Condensing is a way to make K-NN faster. This technique removes data which is not needed and eventually saves the computation time. This experiment uses condensed set of samples and run K-NN algorithm with various parameters like k= 1, 3, 5, 7, 9 and different number of training samples.

Algorithm

Below are the steps in algorithm-

1. Split the dataset as mentioned in K-NN algorithm.

2. Pass the training data X and training data Y to the condense function.

3. Indices of the condensed samples will be retuned

4. Build the new set of training data using condensed samples.

5. Call K-NN with new training data.

6. Verify the results to find the accuracy and time taken.

Here, the main function is 'condensedIdx = condensedata(trainX, trainY)'.

Functions

Function calculates the condensed set by removing data which is not needed to make decisions. These samples **defines the decision boundary**.

1. Out of all training samples, pick 1 sample at random.

2. Classify all the test data with above 1 training example.

3. Check how many samples were wrongly classified.

4. Pick a random sample from wrongly classified set.

5. Classify the test data with above two training samples.

6. Repeat the step 3 to step 5 until all the samples are correctly classified or length of condensed set is equal to actual training data

This function will return the indices of condensed samples.

Results

Below is illustration for small example of 10 samples.

```
Orginal : ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
condensed set [4]
Value of K= 1
knn     : ['I' 'I' 'I' 'I' 'I' 'I' 'I' 'I' 'I' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [0, 1, 2, 3, 5, 6, 7, 8, 9]

condensed set [4, 1]
Value of K= 1
knn     : ['O' 'O' 'I' 'O' 'I' 'I' 'O' 'O' 'I' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [0, 2, 3, 5, 6, 7, 8, 9]

condensed set [4, 1, 5]
Value of K= 1
knn     : ['O' 'O' 'I' 'E' 'I' 'E' 'O' 'O' 'I' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [0, 2, 3, 6, 7, 8, 9]

condensed set [4, 1, 5, 0]
Value of K= 1
knn     : ['B' 'O' 'I' 'E' 'I' 'E' 'O' 'O' 'I' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [2, 3, 6, 7, 8, 9]

condensed set [4, 1, 5, 0, 8]
Value of K= 1
knn     : ['B' 'O' 'I' 'E' 'I' 'E' 'O' 'L' 'L' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [2, 3, 6, 9]

condensed set [4, 1, 5, 0, 8, 3]
Value of K= 1
knn     : ['B' 'O' 'I' 'F' 'I' 'E' 'O' 'L' 'L' 'I']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [2, 6, 9]

condensed set [4, 1, 5, 0, 8, 3, 9]
Value of K= 1
knn     : ['B' 'O' 'L' 'F' 'I' 'E' 'O' 'L' 'L' 'L']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
Not classified  [6]

condensed set [4, 1, 5, 0, 8, 3, 9, 6]
Value of K= 1
knn     : ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
original: ['B' 'O' 'L' 'F' 'I' 'E' 'P' 'L' 'L' 'L']
```

```
Not classified  []
[4, 1, 5, 0, 8, 3, 9, 6]
```

Here indices of condensed set are [4, 1, 5, 0, 8, 3, 9, 6]

Time Taken						
k/#Training Samples	100	1000	2000	5000	10000	15000
1	2.41	18.16	36.78	118.84	345.53	670.28
3	2.01	19.36	35.65	114.19	355.24	720.95
5	1.99	17.21	35.34	126.12	367.38	777.63
7	1.99	17.4	36.16	125.98	360.88	738
9	2.08	17.06	35.67	116.75	335.64	736.25

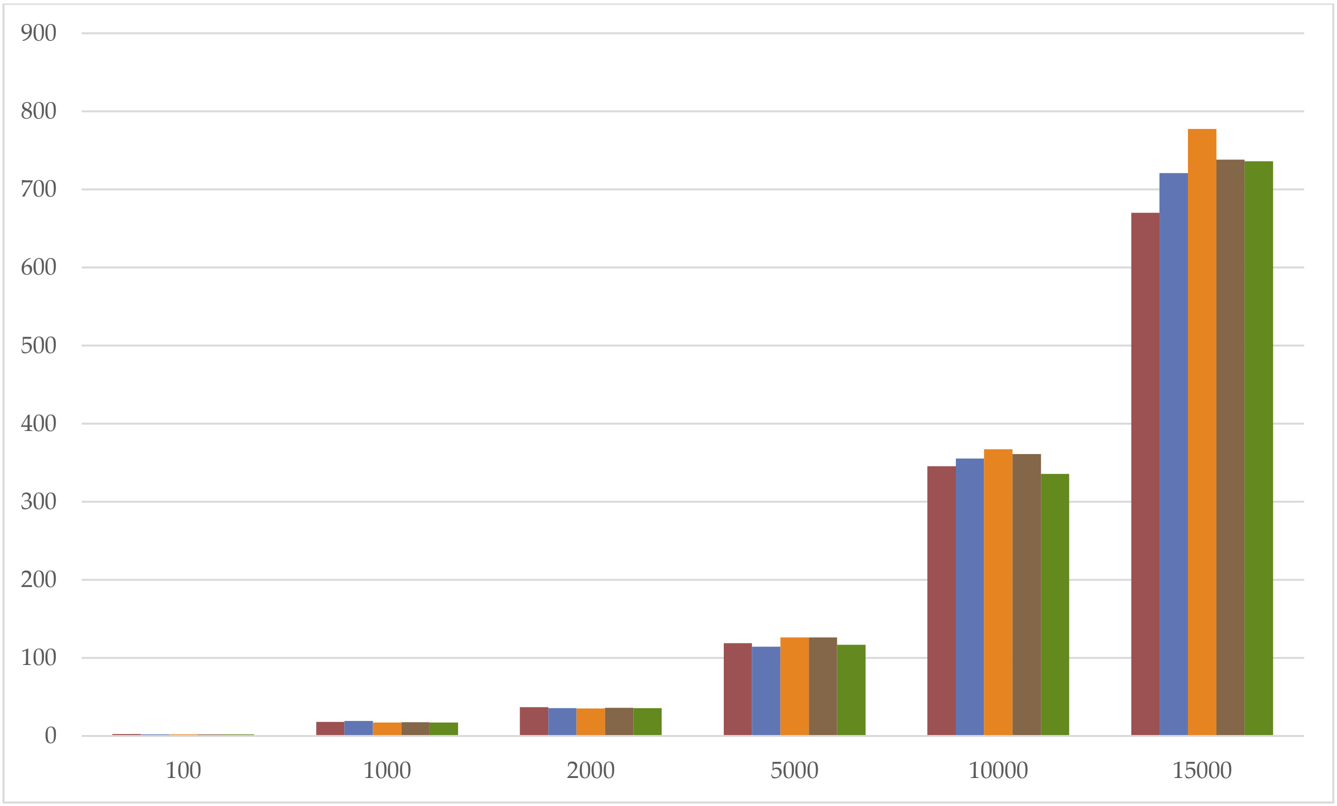

Figure 3: Graphical representation of accuracy by the condense K-NN algorithm for different training samples (X- Axis) and time taken in seconds (Y- Axis). Each color column represents the value for k.

Accuracy						
k/#Training Samples	100	1000	2000	5000	10000	15000
1	33.58	71.22	80.44	87.06	91	92.31
3	22.3	56.9	64.78	75.86	82.82	84.67
5	20.8	53.8	63.33	73.65	79.54	82.54
7	18.98	49.68	59.08	70.23	76.88	80.12
9	18.9	46.32	59.04	66.01	70.84	75.74

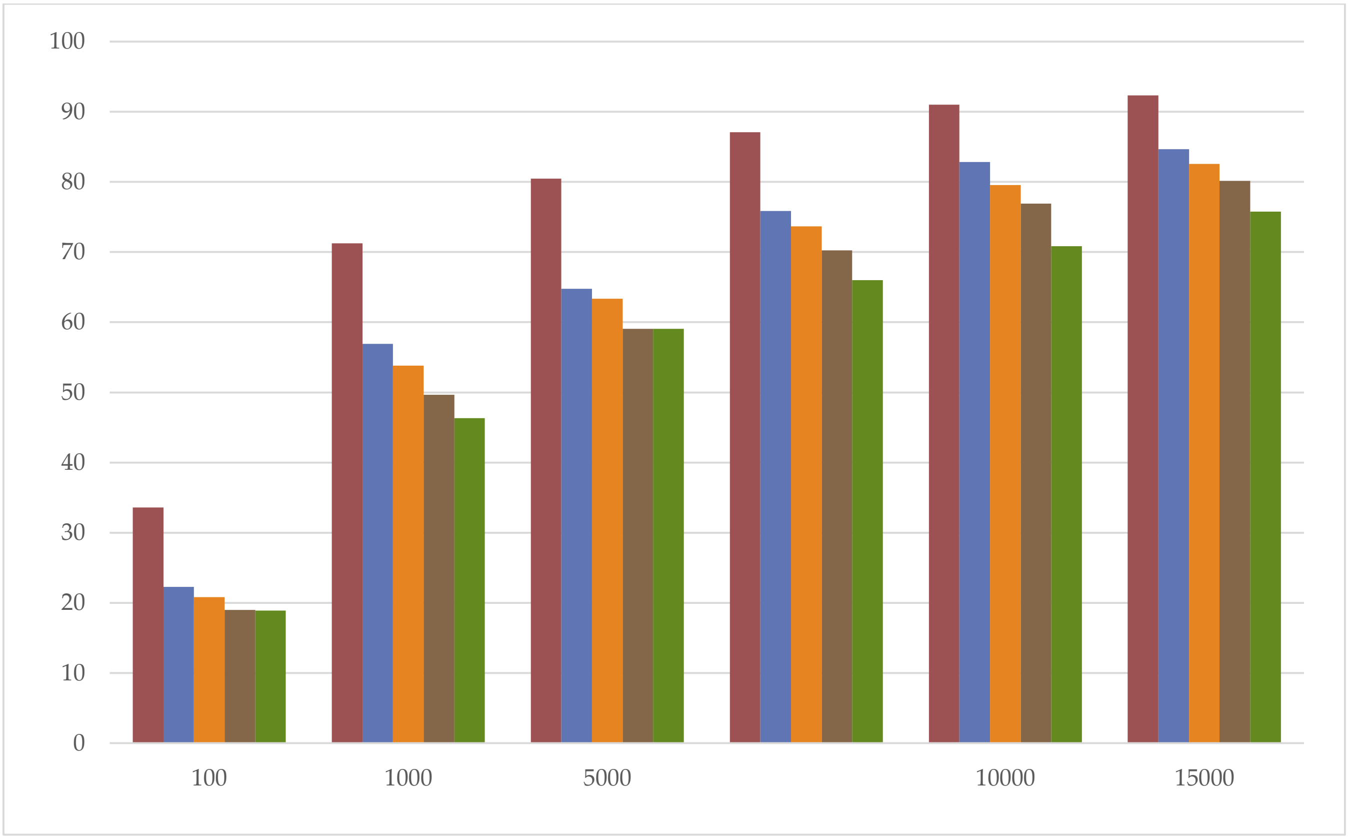

Figure 4: Graphical representation of accuracy by the condensed K-NN algorithm for different training samples (X- Axis) and accuracy in % (Y- Axis). Each color column represents the value for k.

Condensed K-NN vs K-NN algorithm

According to the 60 experiments, let us compare the result of Condensed K-NN vs K-NN algorithm. Comparison for time taken to complete the classification for 1000 samples is shown in the graph.

X- axis : value of k

Y- axis : time taken in seconds

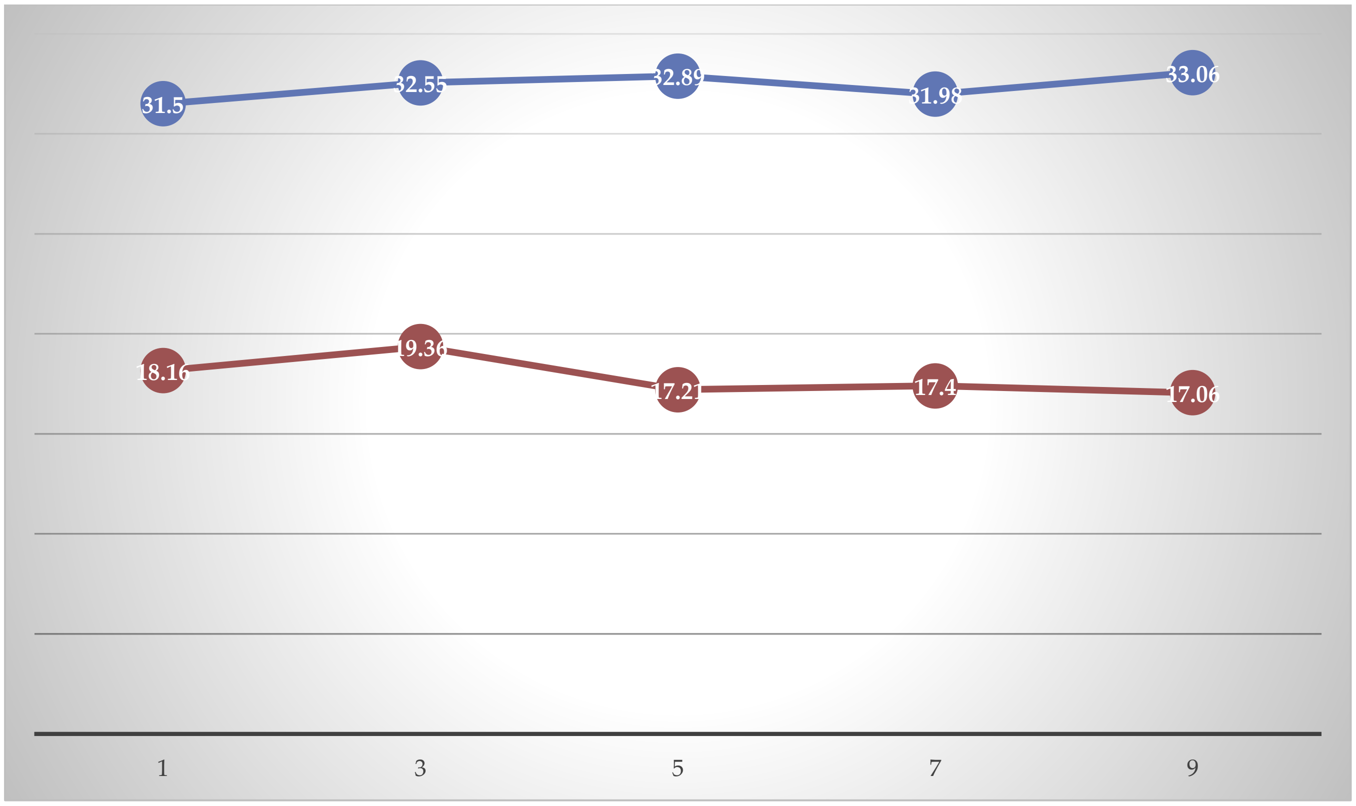

Figure 5: Blue line represents running time without condensing and brown line represents running time with condensing. Training data size was 1000

Statistics when training examples = 15000

```
K=1

CondenseSet : 2515

Accuracy : 92.44

Time : 670.286999941

K=3

CondenseSet : 2543

Accuracy : 84.32

Time : 720.950999975

K=5
```

```
CondenseSet : 2558

Accuracy : 82.54

Time : 777.634000063

CondenseSet : 2535

K=7

Accuracy : 80.12

Time : 738.003999949

K=9

CondenseSet : 2504

Accuracy : 75.74

Time : 736.259000063
```

Statistics when training examples = 10000

```
K=1

CondenseSet : 1976

Accuracy : 91.0

Time : 345.532999992

K=3

CondenseSet : 2002

Accuracy : 81.82

Time : 355.240999937

K=5

CondenseSet : 2022

Accuracy : 79.54

Time : 367.389000177

K=7

CondenseSet : 1963
```

```
Accuracy : 76.88

Time : 360.885999918

K=9

CondenseSet : 1960

Accuracy : 70.84

Time : 335.655999899
```

Confusion Matrix

Confusion matrix is a square matrix with dimensions of number of labels. Program outputs the matrix in a txt file with name ‘ConfusionMatrix.txt’. Below is the sample file ran with 100 training examples.

ConfusionMatrix.txt

The sum of ith row represents how many times label i has occurred. Column j represents how many times label i got classified as label j

Challenges and improvements

This codes implements efficient technique to calculate condensed data. This technique **avoids the repeated distance calculations** which makes condense algorithm work very fast. Below is the particular case of 1000 training examples -

Time taken when we re-use the K-NN function to find out a condense set = 1126.78 sec

Time taken when new technique is used to find out a condense set = 18.16 sec

This shows that the improvement is 1126-18/18 = **6155 time improvement.**

How the computation time is saved?

Imagine the condense algorithm run in traditional method

1. Pick a random sample from training and pass it as a training sample to K-NN algorithm

2. K-NN will return labels for all the testing data.

3. Verify the labels and calculate the how many of them are incorrectly classified.

5. Pick a random sample from those incorrect classified samples.

6. Now sample from step 1 and step 5 are the two training samples which are sent to K-NN and repeat step 2.

Here, we are calculating distance of all the testing sample from training example. But, we have already calculated the distance from previous training sample (sample from step 1). We will store this distance and is not needed to be recalculated.

By careful observation, we calculate the distance of 1st training sample 'n' times where n is the number of training samples.

Along with this, use of power function to calculate Euclidian distance instead of 'distance.euclidian' function saves 50% of the time.