2015-2016

Machine Learning

CSD3939 – COURSEWORK 2

THOMAS-DANIEL BORG

This report presents algorithms being used to build a machine learning system to try to recognize UCI digits and categorizing them. There are two datasets provided on the UCI website. The training data which is used to train the system with provided data and the test data which is used to test this data against the training data.

The data consists of sixty-four values. The first sixty-three are input attributes of integers consisting of a range from 0 – 16 whereas the last attribute is the class code which in my report and code will be called the label. There are 10 labels in all with a range from 0 – 9.

Three algorithms were implemented in total. The Nearest Neighbor Algorithm, the k-Nearest Neighbor Algorithm and a simplified version of the K-Means Algorithm. These algorithms will perform a two-fold test and the results for the accuracy will be reported in this report. This was easily done by just swapping the test data with the training data when calling from the functions in the code.

Table of Contents

Algorithms Used

Three algorithms have been implemented in the categorizer application. These algorithms are nearest neighbor, K-nearest neighbor and a modified version of the K-means algorithm.

Nearest-Neighbor Algorithm

This was the first algorithm implemented for this coursework. Since this was the simplest of algorithms, it was used as a starting point as well as a building block for the other upcoming algorithms.

The core function of this algorithm is to find the distance between a training and data set. This distance is known as the Euclidean distance and can be calculated by using the formula below -

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

, where *a* is the training data and *b* is the test data.

The above formula takes into consideration one dataset from each of the available datasets, i.e. the training data and the test data. The first element from the training data is subtracted from the first element of the test data, then squared and added with a repetition of the other elements. After adding up all the elements, a square root of the answer is performed. This will then be repeater for all the training data. Therefore, one test data set is compared to all the training data and a distance will be given for each. The results will then be sorted and the least distance will have the highest probability of being the answer. The label is taken from the training data set with the least distance and compared to the label of the test data set. If these are identical, then the test data would have returned a correct value.

Therefore, each test data set is compared to all of the training data sets and results will be written in the class code for the training data containing the least distance. This will then be compared and a boolean value in the test data set class code is updated to 1 if

the label is correct or 0 if the label is incorrect. After all the test data sets are traversed, all the correct test data sets are counted and a percentage accuracy will be given. A general accuracy will be given as well as an accuracy for each label, that is a percentage for each label from 0 to 9.

K-Nearest-Neighbor Algorithm

This algorithm is an improved version of the previous algorithm. All functionality is the same up until finding the Euclidean distance but instead of taking the least distance, we take the least k amount of distances (k is inputted before applying the nearest neighbor code) and the most frequent occurrences is taken to be the highest probability of the recognized label.

To provide an example for the above, after performing a normal nearest neighbor algorithm, we are presented with 3000+ different distances. These are then sorted and if K is equal to five, we take the first 5 smallest distances. These are then converted into the corresponding label, ex: [1, 0, 0, 1, 0]. As can be seen, the 0 has a higher number of occurrences, therefore we take the 0 label as the resultant label and check this label against the test label to see whether this is correct.

This was also modified to take care of another situation which has given me a higher level of accuracy. Consider K is equal to five and we are given the following five label - [0, 1, 1, 0, 5]. As can be seen, the highest occurrence of label has 2 different labels this time. The 0 which has 2 occurrences and 1 which also has two. Strictly speaking, both values are good but the code has been modified so if this situation is encountered, apart from taking the label with the highest occurrence, it will then fall down to choose also the least distance. Since the 5 labels are sorted from lowest to highest distance and both 0 and 1 have the same number of occurrences, the logic will select 0 rather than the other label as a result as the 0 label still has the lowest Euclidean distance.

K-Means

This algorithm is a simplified version of the original k-means algorithm. This didn’t return a very accurate result. This is mainly because of 3 reasons.

1. K-Means assumes the variance of the distribution of each attribute is spherical.
2. All variables have the same Variance
3. The prior probability for all k clusters are the same. I.e. Each cluster has roughly equal number of observations.

If any one of these 3 assumptions are violated, then k-means will fail, hence returning the lowest accuracy in certain situations. Having said that, an accuracy of 89.37% was achieved which was still quite successful.

This algorithm is also quite similar to the Nearest Neighbor algorithm but the training data set has to first be grouped by their category. All datasets which have the same label have to be grouped together by taking their average and a single dataset with that label is created. An example below is shown explaining this step.

Training Data Set 1

1st Element	2nd Element	3rd Element	4th Element	Class Label
0	2	1	1	5

Training Data Set 2

1st Element	2nd Element	3rd Element	4th Element	Class Label
1	2	1	3	5

Training Data Set 3

1st Element	2nd Element	3rd Element	4th Element	Class Label
0	3	2	2	5

In the above data sets, there are 5 elements in total in each data set with the final element being the class label. This shows that all the datasets with a label of 5 are being grouped. The next step is to take the first element of each data set, add them and divide by the total number of datasets, in this case 3. (0 + 1 + 0)/3 = 0.33333

Next step is to do the same with the second element and so on up until the fourth element. The fifth element will not be processed as this is the label.

2nd element - (2+2+3)/3 = 2.33333
3rd element - (1+1+2)/3 = 1.33333
4th Element - (1+3+2)/3 = 2.00000

We then create a brand new dataset with each if the elements we have calculated together with the label for its fifth element.

1st Element	2nd Element	3rd Element	4th Element	Class Label
0.33333	2.33333	1.33333	2.00000	5

This step is complete for all labels of five. The same must be done for all the other 9 labels from 0 till 9.

After doing so, the new dataset will replace the old training data and the same process of the Nearest Neighbor algorithm will apply by calculating the Euclidean distance of the Test data set using the same formula against the new training data set. The smallest distance will be the resultant label and compared to the label of the test data sets to see whether this is correct.

Quality of Code

The code has been implemented in an object-oriented way. There are a total of 5 classes.

1. Menu.java - This is the main menu that is presented to the user to load the training and test data, aswell as the worker algorithms that have been discussed above.
2. DataSet.java - This is the class that holds the training data sets.
3. DataSetTest.java - This is the class that holds the test data sets. This differs from the above by having a boolean value of correct or not correct.
4. DataSetAvg.java - This is the class that holds the training data sets for the K-means algorithm.
5. Coursework2.java - This class holds all the algorithms which shows the results.

DataSet.java

This class implements the Comparable which is used to order the dataset by distance to return the smallest distance first. This distance is the Euclidean distance. There are then getters and setters for the following -

Label - The graphical representation of the data.
Data - The actual data in numerical form saved in a List of type Integer.
Distance - The Euclidean distance

There also is a method which implements Comparator that will order the dataset by distance as can be shown below.

```
public static class OrderByDistance implements Comparator<DataSet> {
    @Override
    public int compare(DataSet o1, DataSet o2) {
        return o1.getDistance() > o2.getDistance() ? 1 : (o1.getDistance() < o2.getDistance() ? -1 : 0);
    }
```

DataSetAvg.java

The DataSetAvg.java is exactly the same but keeps stored a new data set that is used for the K-Means algorithm. Only the values in the Data variable differs as the other variables are kept.

DataSetTest.java

The DataSetTest.java class differs from the other data set classes. The distance variable isn't needed as this is kept in the previously mentioned classes. Also this class does not implement the Comparable class since no comparison is being done in this class. There is a boolean variable named correct. This is set to true if the label value of the least distance in the DataSet class is equal to the current dataset being tested. If not, this is set to false. This value is used as a flag to calculate the accuracy of the algorithm.

```
public boolean getCorrect()
{
    return correct;
}

public void setCorrect (boolean correct)
{
    this.correct = correct;
}
```

Coursework2.java

This is the class that contains all the algorithms functionality Some of which will be described below.

This figure shows the looping functionality present in all the algorithms. The outer loop goes through all the testing data set whereas the inner loop goes round all the training

data sets. Therefore every test dataset is ‘compared’ to all the test data sets.

```
1  counterTest = 0;
2  for(DataSetTest dsTest:dataTest)
3  {
4          counterTrain = 0;
5          for(DataSet dsTrain:dataTrain)
6          {
7           counterTrain++;
8          }
9
10 counterTest++;
11 }
```

Within the Inner for-loop (dsTrain on line 5), a recursive function is performed to solve the Euclidean distance which can be found in the below figure. The values of x refers to the element number in the data set. This is incremented to cover all the values from 0 till 64 (all 65 values, excluding the data set label). An if statement checks whether the value of x is less than the size of the test data array. If this is true, then the element found in position x of the training data set currently in loop is added to the same value found in position of the test data set currently in loop and the answer is squared. After doing so, the function recalls itself by passing a newly incremented value of x together with the tempResult. This is done for all the positions up until 64, which is then returned normally to the main method of the algorithms and a square root is applied on the returned value. This value is the Euclidean distance.

```
1  public int InnerCalculation(int x, int tempResult)
2  {
3      if (x < dataTest.get(counterTest).getData().size()-1)
4      {
5          tempResult += Math.pow((dataTrain.get(counterTrain).getData().get(x)-dataTest.get(counterTest).getData().get(x)),2);
6          return InnerCalculation(x+1, tempResult);
7      }
8
9      return tempResult;
10 }
```

The value is saved by using the Test data set class by setting the distance using the setter.

Next step was to sort the training data set by order of distance.

In case of the **Nearest Neighbor Algorithm**, check is then performed by checking if the current data set being tested has the same label as the training data set with the least Euclidean distance. If this is correct, the current data set being tested is set to true, else false. This is used to later find the accuracy by counting all the true values, dividing by the amount of test data sets and multiplied by 100 to give an accuracy in percentage form.

```
1  if(dataTrain.get(0).getLabel()==dataTest.get(counterTest).getLabel())
2  {
3      System.out.println("Correct");
4      dataTest.get(counterTest).setCorrect(true);
5  }
6  else
7  {
8      System.out.println("Incorrect");
9      dataTest.get(counterTest).setCorrect(false);
10 }
```

In case of the **K-Nearest Neighbor**, a k value is required by the user to input. All the functionality is the same like the Nearest Neighbor but differs after sorting all the distances by distance. The program will take into consideration a number of minimum distances, this amount being equal to the k value inputted by the user.

```
1  for(int i=0;i<kValue;i++)
2  {
3      System.out.println("Label - "+dataTrain.get(i).getLabel());
4      System.out.println("Distance - "+dataTrain.get(i).getDistance());
5      kValues.add(dataTrain.get(i).getLabel());
6  }
```

A loop is performed to loop for as many times as the kValue is and get the distance and label of the first few distances and the labels are added to a List of kValues. Next step was to find the mode of the labels, ie the number of occurrences of the label which might point towards the correct and most probable digit.

```
HashMode.put(0, Collections.frequency(kValues, 0));
                HashMode.put(1, Collections.frequency(kValues, 1));
                HashMode.put(2, Collections.frequency(kValues, 2));
                HashMode.put(3, Collections.frequency(kValues, 3));
                HashMode.put(4, Collections.frequency(kValues, 4));
                HashMode.put(5, Collections.frequency(kValues, 5));
                HashMode.put(6, Collections.frequency(kValues, 6));
                HashMode.put(7, Collections.frequency(kValues, 7));
                HashMode.put(8, Collections.frequency(kValues, 8));
                HashMode.put(9, Collections.frequency(kValues, 9));

                 int maxValueInMap=(Collections.max(HashMode.values()));  // This will return max value in the Hashmap
                    for (Map.Entry<Integer, Integer> entry : HashMode.entrySet()) {  // Itrate through hashmap
                        if (entry.getValue()==maxValueInMap) {
                            //System.out.println(entry.getKey());     // Print the key with max value
                        Mode.add(entry.getKey());
                        }
                    }
```

This might return more than one mode if any of the labels have equal occurrences. If this should happen, the program will then take into consideration the mode with the least Euclidean distance.

```
HashMap<Integer,List> hash = new HashMap<Integer,List>();
            hash.put(0, avgDataTrain0);
            hash.put(1, avgDataTrain1);
            hash.put(2, avgDataTrain2);
            hash.put(3, avgDataTrain3);
            hash.put(4, avgDataTrain4);
            hash.put(5, avgDataTrain5);
            hash.put(6, avgDataTrain6);
            hash.put(7, avgDataTrain7);
            hash.put(8, avgDataTrain8);
            hash.put(9, avgDataTrain9);



            for(int a=0; a<hash.size(); a++)
            {
            List<List>dataTemp = new ArrayList<List>();

                for(int b = 0; b<hash.get(a).size();b++)
                {
                    dataTemp.add((List)hash.get(a).get(b));
                }

            List <Double> avgData = new ArrayList<Double>();

                for(int x = 0;x<64;x++)
                {
                    double average = 0;
                    int sum = 0;

                    for(int y=0;y<dataTemp.size();y++)
                    {
                        sum = sum + (Integer)dataTemp.get(y).get(x);
                    }
                    average = (double)sum/dataTemp.size();
                    avgData.add(average);
                }
               DataSetAvg dSA = new DataSetAvg(a,avgData);
               dataTrainAvg.add(dSA);
               dataTemp.clear();


            }
```

The **K-Means** Algorithm differs from the 2 mentioned above only at the beginning stage. The first step to apply a K means Algorithm is to get all the training data set that have the same label(lines 1-11) and find the average(lines 24-36) upon each element and end up with one single data set. In total ending up with 10 unique-labeled training data sets ranging from 0 to 9. The same functionality will then be performed by finding the Euclidean distance by applying the same formula for the Nearest Neighbor and by taking the least distance.

Results

Two-Fold Test (A)

Training Data - optdigits.tra

Testing Data - optdigits.tes

Class Code	0	1	2	3	4	5	6	7	8	9	Average
NN	100	99.45054945	98.8700565	97.81420765	98.34254144	98.35164835	100	98.88268156	94.25287356	93.88888889	97.9966611
K-Means	98.31460674	74.72527473	88.70056497	88.52459016	91.16022099	93.95604396	96.68508287	94.97206704	79.31034483	87.22222222	89.37117418
k-NN (2)	100	99.45054945	98.8700565	97.81420765	98.34254144	98.35164835	100	98.88268156	94.25287356	93.88888889	97.9966611
k-NN (3)	100	99.45054945	97.74011299	98.36065574	98.34254144	98.35164835	100	96.08938547	93.10344828	97.77777778	97.9410128
k-NN (4)	100	100	98.30508475	98.36065574	98.34254144	98.35164835	100	97.76536313	94.25287356	97.22222222	98.27490262
k-NN (5)	100	100	98.30508475	96.72131148	98.89502762	98.35164835	99.44751381	96.64804469	93.67816092	96.66666667	97.8853645
k-NN (6)	100	100	98.8700565	97.81420765	98.89502762	98.35164835	100	97.76536313	93.67816092	96.66666667	98.21925431
k-NN (7)	100	99.45054945	98.30508475	97.26775956	98.34254144	98.35164835	100	97.76536313	91.95402299	96.66666667	97.82971619
k-NN (8)	100	100	97.74011299	97.81420765	98.34254144	98.35164835	100	97.76536313	92.52873563	96.66666667	97.9410128
k-NN (9)	100	98.9010989	97.74011299	97.26775956	98.89502762	98.35164835	100	97.76536313	91.95402299	96.66666667	97.77406789

The first run of the algorithms developed was done by using the original test data and training data given for this coursework. As can be seen from the table above, the average accuracy for the Nearest Neighbor was of 97.996%, the K means gave an overall accuracy of 89.371%. The best results are shown when using the K Nearest Neighbor Algorithm with a K-value of 4. Also it should be noted that 100% is returned for all the 0 label tests apart for the K Means which gave an accuracy of 98.315%. This can also be seen for the label of 6 which also returns 100% except for K-Means with an accuracy of 96.685% and K-Nearest Neighbor when k-value is equal to five which returned an accuracy of 99.448%.

Results are also shown for each label for Nearest Neighbor, K- Means and all K-Nearest Neighbors with k-values ranging from 2 till 9.

Two-Fold Test (B)

Training Data - optdigits.tes

Testing Data - optdigits.tra

Class Code	0	1	2	3	4	5	6	7	8	9	Average
NN	99.46808511	97.42930591	99.73684211	96.14395887	96.38242894	94.94680851	99.20424403	99.48320413	93.15789474	94.5026178	97.04420612
K-Means	99.20212766	86.88946015	93.68421053	92.80205656	86.82170543	79.78723404	98.67374005	98.70801034	89.73684211	84.03141361	91.02798849
k-NN (2)	99.46808511	97.42930591	99.73684211	96.14395887	96.38242894	94.94680851	99.20424403	99.48320413	93.15789474	94.5026178	97.04420612
k-NN (3)	99.46808511	97.68637532	99.73684211	98.20051414	95.86563307	94.94680851	99.46949602	98.96640827	92.36842105	95.54973822	97.2273084
k-NN (4)	99.73404255	97.42930591	99.73684211	97.94344473	96.38242894	94.68085106	99.20424403	98.96640827	92.36842105	95.02617801	97.14883599
k-NN (5)	99.20212766	97.94344473	99.21052632	98.45758355	96.12403101	95.21276596	99.20424403	98.70801034	93.15789474	92.93193717	97.01804865
k-NN (6)	99.20212766	97.94344473	99.73684211	98.71465296	96.38242894	95.4787234	99.20424403	98.96640827	92.10526316	94.2408377	97.20115093
k-NN (7)	99.20212766	97.94344473	98.94736842	98.71465296	95.60723514	95.21276596	99.20424403	98.96640827	91.84210526	93.45549738	96.91341878
k-NN (8)	99.20212766	97.42930591	99.21052632	97.94344473	95.60723514	95.21276596	99.20424403	98.70801034	92.63157895	92.93193717	96.80878891
k-NN (9)	99.20212766	97.68637532	99.21052632	97.94344473	95.34883721	95.21276596	99.20424403	98.96640827	92.36842105	91.09947644	96.62568663

The tests were tested again by swapping the training data sets with the test data sets to perform a Two-Fold test. The results in the above table were given.

Both the Nearest Neighbor and the K Nearest Neighbor have produced positive results but are slightly less accurate when performing the same tests with the original test and training data sets. This is clearly shown by the average percentage results when compared to the original which are less and also that no 100% accuracy is present. This can especially be seen when comparing labels 0 and 6 which have not produced a single 100% whereas the previous test resulted to all 100% accuracy for the 0 label and also the same for the sixth label.

The most accurate algorithm is the K-Nearest Neighbor when giving it a k-value of three. On the other hand, the K-Means algorithm has provided a much better average accuracy in this test, resulting to an accuracy of 91.028% when compared to the previous test which resulted to an accuracy of 89.371%.

Conclusion

This report has been finalized and presents all the work required by the coursework requirements to build a system to categorize the UCI digit tasks. The three algorithms developed in the system have all given positive results as explained above. All working processes of these algorithms were also explained.

The most successful algorithm was the K-Nearest Algorithm with a k-value of four and six. The Nearest Neighbor was the second best followed by the K-Means algorithm which gave the least accuracy.

A Two-Fold test was also supplied where the training set was swapped with the test data set. The accuracy was decreased in this process for the Nearest Neighbor and the K-Nearest Neighbor whereas the K-Means has shown an improvement in accuracy.

References

Wilcox Jones, P., Osipov, A. and Vladimir Rokhlin (2011) ‘Randomized approximate nearest Neighbors algorithm’, *Proceeding of the National Academy of Sciences of the United States of America,* 108(38).

Shengyi Jiang, Pang, G., Wu, M. and Kuang, L. (2012) ‘An improved K-Nearest Neighbor algorithm for text categorization’, *Expert Systems with Applications,* 39(1), pp. 1503-1509.

Belhaourai, S.B., Ahmed, S. and Mansour, S. (2014) ‘Optimized K-Means Algorithm’, *Mathematical Problems in Engineering,* 2014, pp. 4-18.