Assignment No: 5

Data Mining (IS590DT)

By: Meghna Shrivastava

Solutions:

Step 0 -- Familiarize yourself with the documentation (the .names file) so you understand the problem domain and how the data was constructed.

- For this step, I understood some basic outline of how the dataset is distributed. As per the problem specification, the following information made me explore the further questions with much clarity:

A) Number of Number of Instances

Optdigits.tra.csv Training 3823

Optdigits.tes.csv Testing 1797

B) Number of Attributes: 65

C) Class Distribution for both the testing and training datasets

D) Accuracy on the testing set with k-nn using Euclidean distance as the metric:

k = 1 : 98.00, k = 2 : 97.38 , k = 3 : 97.83, k = 4 : 97.61, k = 5 : 97.89, k = 6 : 97.77, k = 7 : 97.66, k = 8 : 97.66, k = 9 : 97.72, k = 10 : 97.55, k = 11 : 97.89

Step 1 – Visualization. Use the Python program to get a feel for how the numerical tuple representation corresponds to the actual image. Report a sample of image juxtaposted with its corresponding tuple and describe their relationship. Hint: You will need to download and install Python from python.org. Then, place the .py file in the same directory as the data files, open it in the IDLE text editor, and "Run Module".

File used:

optdigits_viewer.py to run the program using IDLE.

```
########
# Viewing images from raw data
# http://ftp.ics.uci.edu/pub/machine-learning-databases/optdigits/
#
# Make sure you have a file named optdigits.tra.csv in the same directory
#  as this Python program
#
# Note that this program was written in haste
#  -- the images look to be rotated and mirrored
#
# by Vetle Torvik 3/2017
# tested with Python 2.7.3 on Windows 7


from cImage import *

#blow up an image by a factor of x
def blowUp(image,x):
    w = image.getWidth()
    h = image.getHeight()
    newimage = EmptyImage(w*x,h*x)

    for row in range(h):
        for col in range(w):
            px = image.getPixel(col,row)
            for i in range(col*x,col*x+x):
                for j in range(row*x,row*x+x):
                    newimage.setPixel(i,j,px)
                    #print row,col,i,j
    return newimage

#training data
fn = 'optdigits.tra.csv'

#testing data
fn = 'optdigits.tes.csv'

fh = open(fn,'r')
#first line in file is just header, not data
header = fh.readline()
print (header)


#read all data into array then close file
lines = fh.readlines()
fh.close()
```

```
#check out instance number 873
# it was predicted 3 but actually an 8 (based on IBk)
lines = lines[871:874]

#images are all 8 by 8
sz = 8

cnt = 0
#loop over each image (row in dataset)
for line in lines:
    line = line.replace('\n','')
    cnt += 1
    x = line.split(',')
    print (cnt,x)

    #create a blank image
    digitImage = EmptyImage(sz,sz)
    i = -1
    #create pixels and populate image
    for row in range(sz):
        for col in range(sz):
            i += 1
            val = int(x[i])

            #change scale from 0-16 to 255-0
            intensity = int(((16-val)/16.0)*255)
            print(row,col,i,val,intensity)
            px = Pixel(intensity,intensity,intensity)
            digitImage.setPixel(row,col,px)
    digit = int(x[i+1])

    digitImage2 = blowUp(digitImage,50)
    #display image
    ti = 'Instance: ' + str(cnt) + 'Digit:' + str(digit)
    myWin = ImageWin(ti,sz*50,sz*50)
    digitImage2.draw(myWin)
    myWin.exitOnClick()
    dum = input('Hit Enter to continue\n')
```

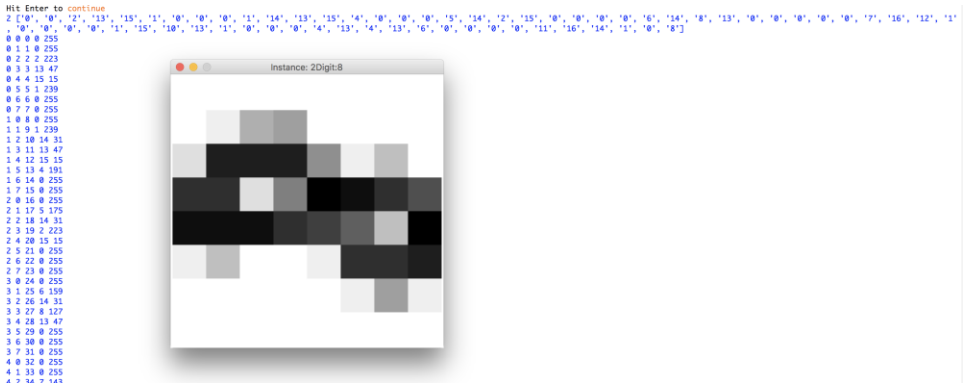
Fig: (Program implemented: optdigits_viewer.py)

We can see that the tuple consists of 5 numbers based on the command: print (row, col, i, val, intensity)

The output results were as follows: (Tuple and actual image placed side by side)

Instance No: 872, digit displayed: 6

Instance No: 873, digit displayed: 8



Instance No: 874, digit displayed: 4

C) Try to replicate the results of the original authors (i.e., use the instance-based classifier they reference and see if you get a similar accuracy). Hint: Remember to try different parameter values for the classifier and use the test set as provided for evaluation. Are there certain digits that are more difficult or easier to distinguish than others? If so, which ones? Optional: Identify an instance that was misclassified and display the corresponding image by modifying the provided Python program.

Ans: for this step, I first went through the '.names' file and determined the accuracy expected of KNN classifier mentioned:

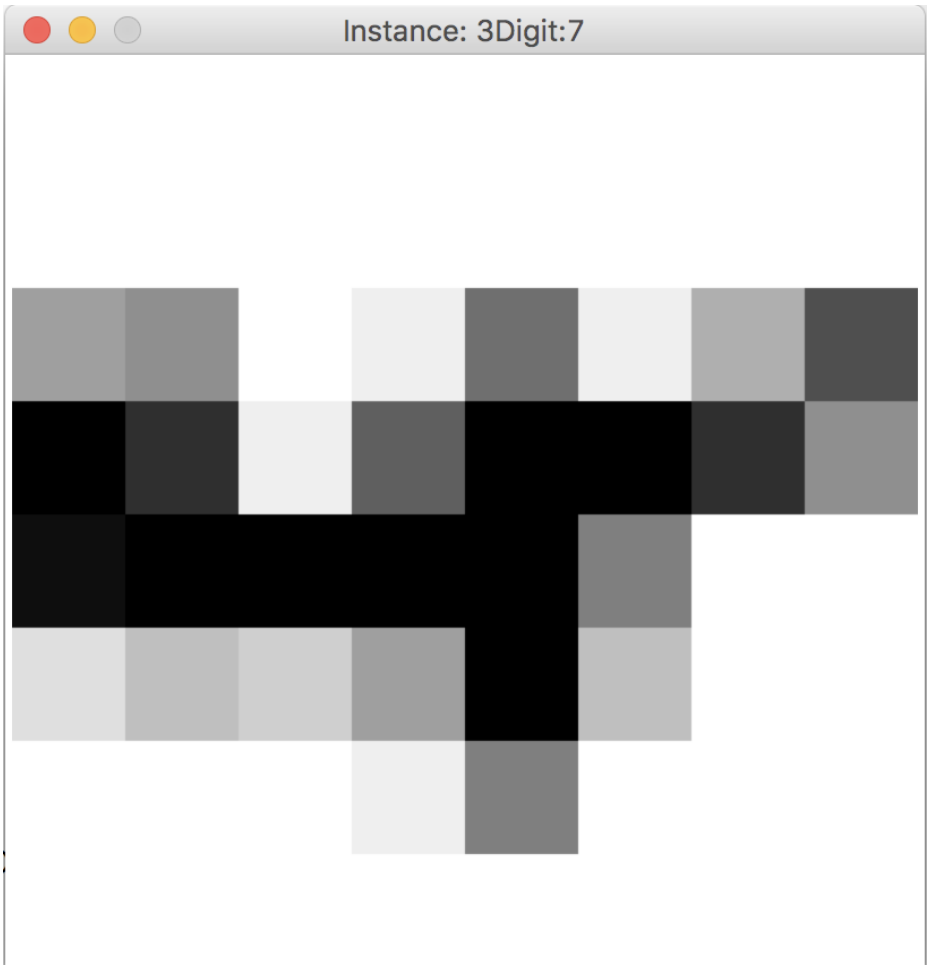
```
Accuracy on the testing set with k-nn
using Euclidean distance as the metric

 k =  1   : 98.00
 k =  2   : 97.38
 k =  3   : 97.83
 k =  4   : 97.61
 k =  5   : 97.89
 k =  6   : 97.77
 k =  7   : 97.66
 k =  8   : 97.66
 k =  9   : 97.72
 k = 10   : 97.55
 k = 11   : 97.89
```

I tried replicating this result using jupyter notebook. Please refer to the Jupyter notebook attachment for the implementation. You'll see that I have tried different parameter values for the classifier and derived almost precise results.

Are there certain digits that are more difficult or easier to distinguish than others? If so, which ones?

I personally found recognizing digit 7 a bit difficult as compared to other digits. Though, it still can be predicted that it is 7.

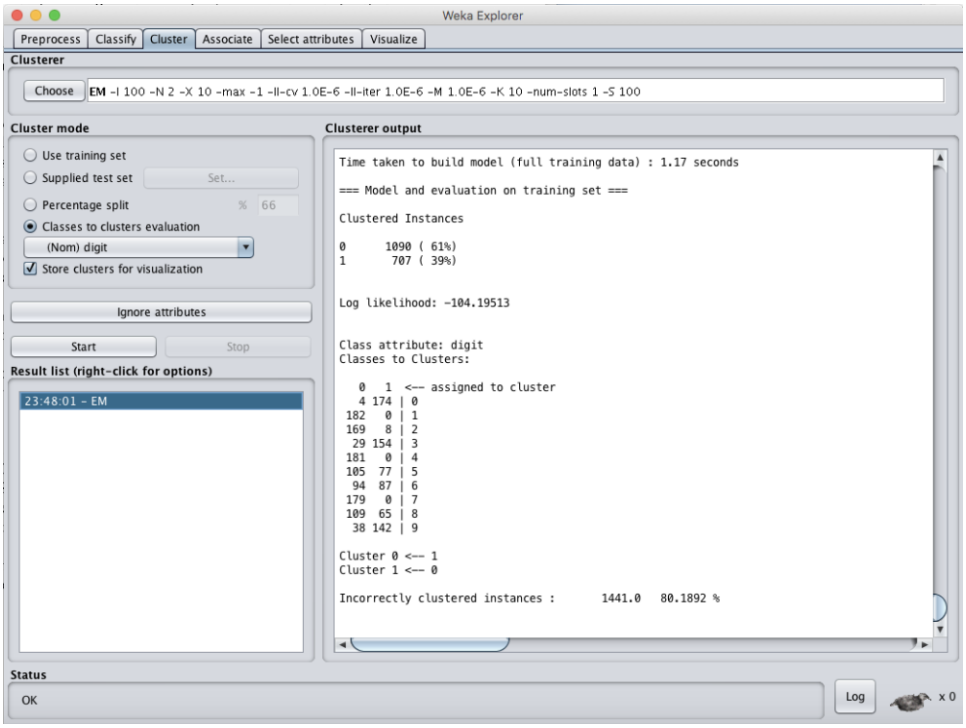


D) Test two non-instance-based classifiers that you know are capable of capturing complex patterns and compare the accuracy with two instance-based classifiers. Do you think instance-based classifiers will generally perform better in this problem domain? Why?

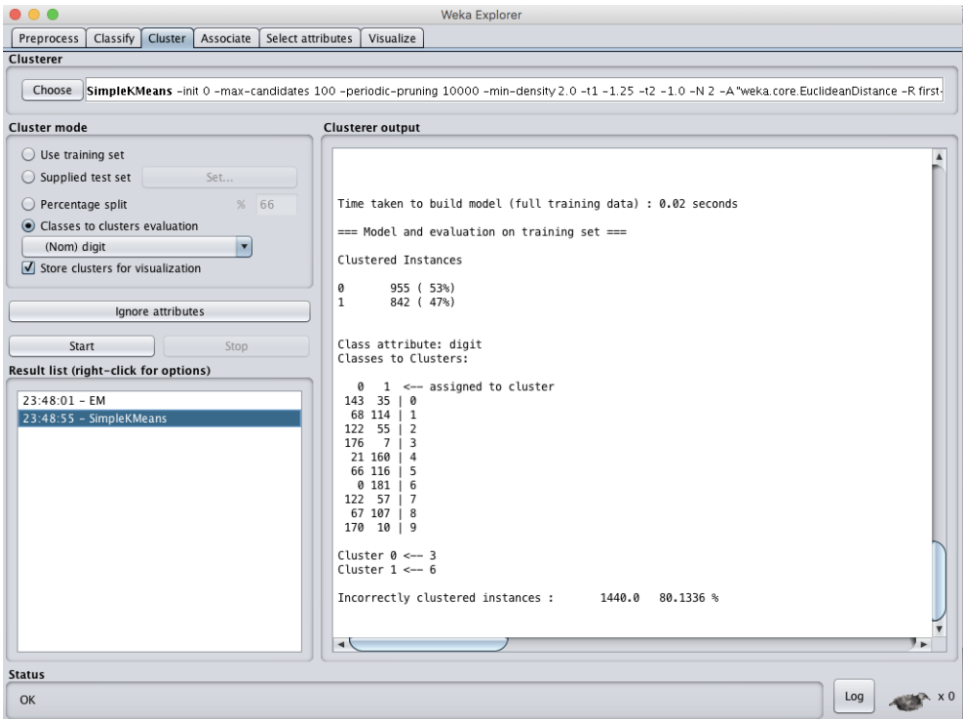
- I have implemented this step in the Jupyter Notebook. Please refer to various classifiers apart from KNN that are implemented.
- The explanation related to the performance is also written just after the implementation of Precision values for KNN (Please refer jupyter notebook for detailed execution)

E) Compare the performance of the classifiers to two clusterers. Hint: Remember to "ignore" the class attribute when clustering. Is the performance of the clusterers surprising? Why?

Result after performing clustering using EM and SimpleKMeans:



Fig; (a) Clustering using EM



Fig; (b) Clustering using SimpleKMeans

I did find the results for clustering somewhat surprising because the results for the classifiers tends to the higher accuracy level whereas the results based on clustering shows higher degree of incorrectly classified instances, EM: 80.1892% and SimpleKMeans: 80.1336%.

F) Pick the best (or your favorite) classifier so far and evaluate it using Weka's 10-fold cross- validation of the training set. Do you get better or worse accuracy when compared to using the provided test set? Why?

Ans:

I chose RandomForestClassifier for this step and performed classification on both Test and Training data:

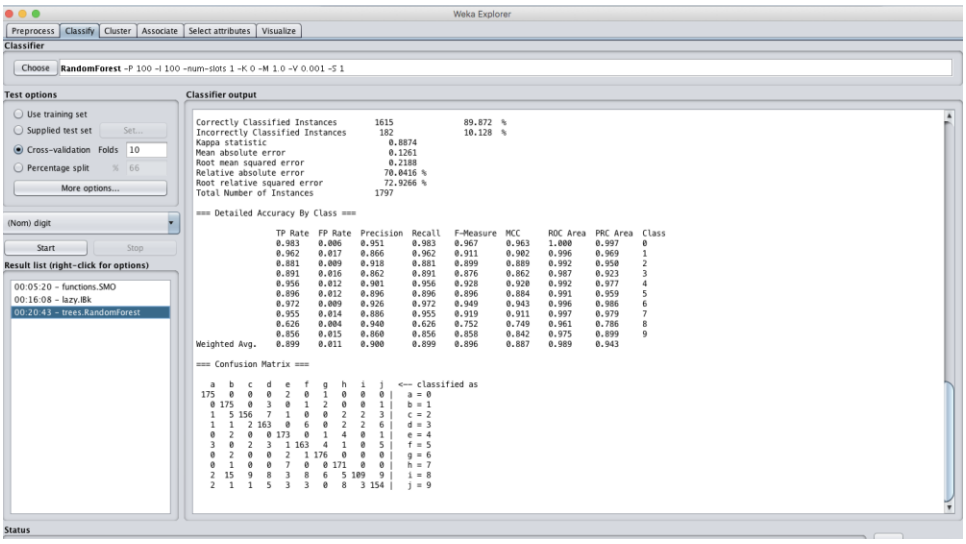


Fig: (a) For Test Data

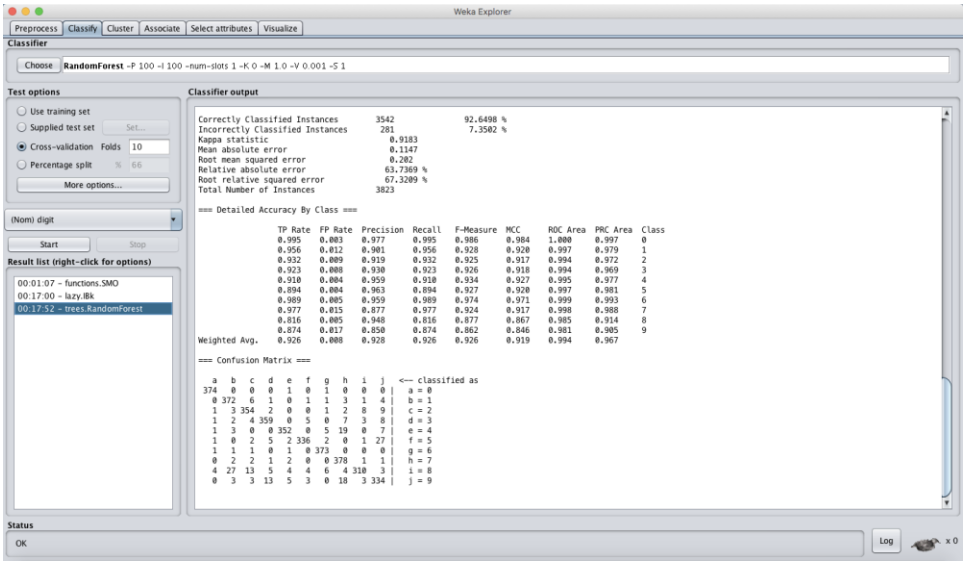


Fig: (b) For Training Data

Result: I got the better accuracy for the training set (Accuracy of 92.65%) as compared to the Training Set (Accuracy of 89.872%). This may be because of the total weightage of the incorrectly classified instances for the both test and training dataset respectively. Though numerically, the incorrectly classified instances for the Training data is more as compared to Test data, the total percentage it represents out of the overall dataset has resulted into the given result (281 incorrect classified instances accounts for only 7.35% and 182 incorrectly classified instances in test data accounts for 10.12% of the total instances)

G) Suppose you expanded the dataset to include additional characters such as lower-cased English alphabetical characters, or any ASCII character, or any UTF-8 character. To what degree would you expect the accuracy to be reduced? Hint: Test to the classifier accuracy for the given dataset restricted to only 2, 3, 4, etc. of the digits (i.e., removing all other instances in the dataset) and report the decrease in accuracy as a function of number of classes.

Execution Steps performed: for the indexes range 1-10

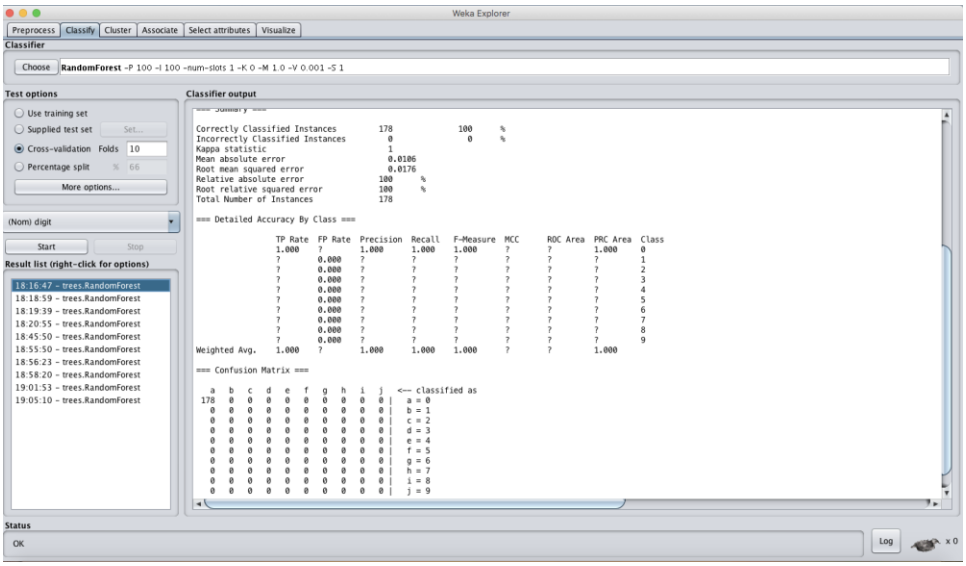


Fig: (a) Only 0s

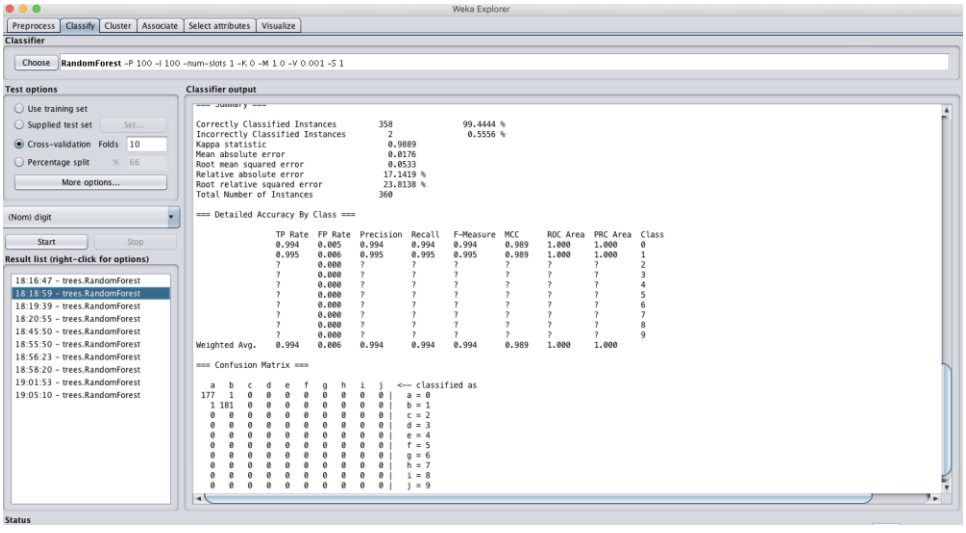


Fig: (b) 0s and 1s

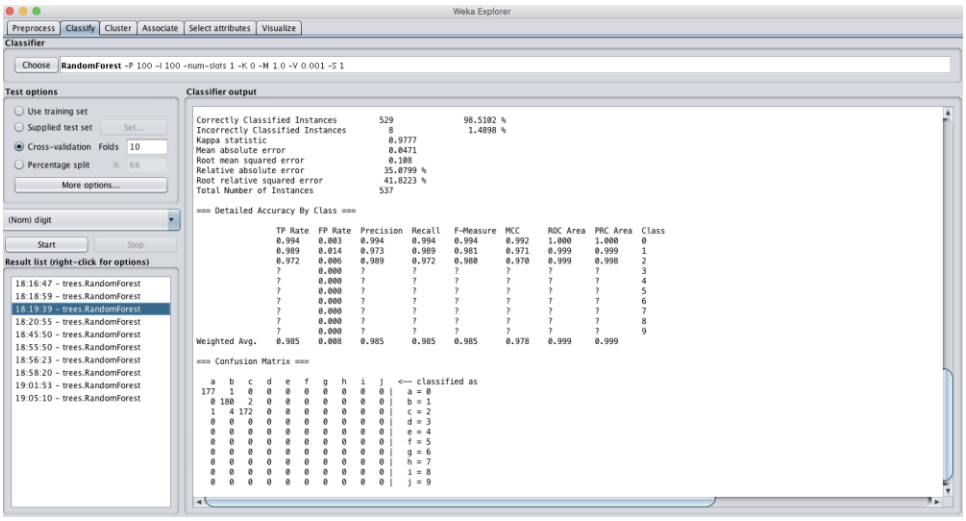


Fig: (c) 0s, 1s, 2s

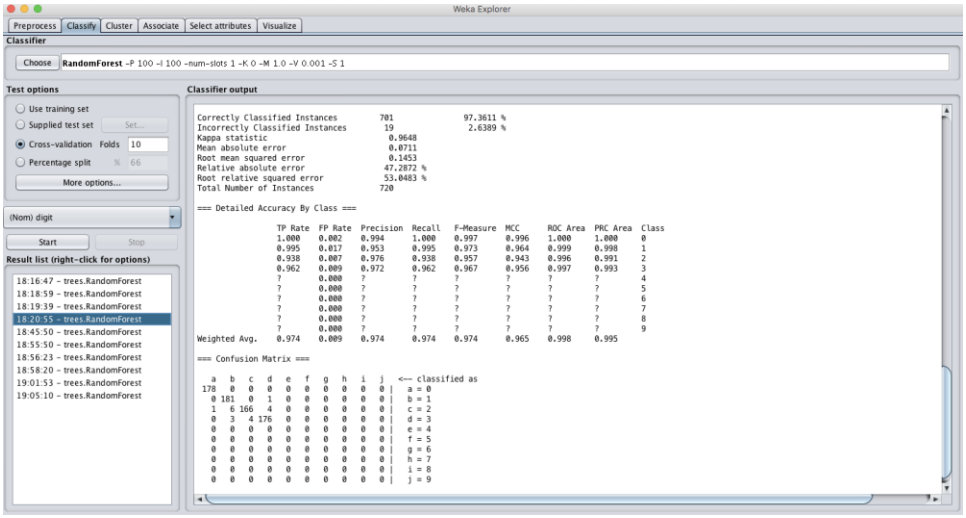


Fig: (d) 0s, 1s, 2s, 3s

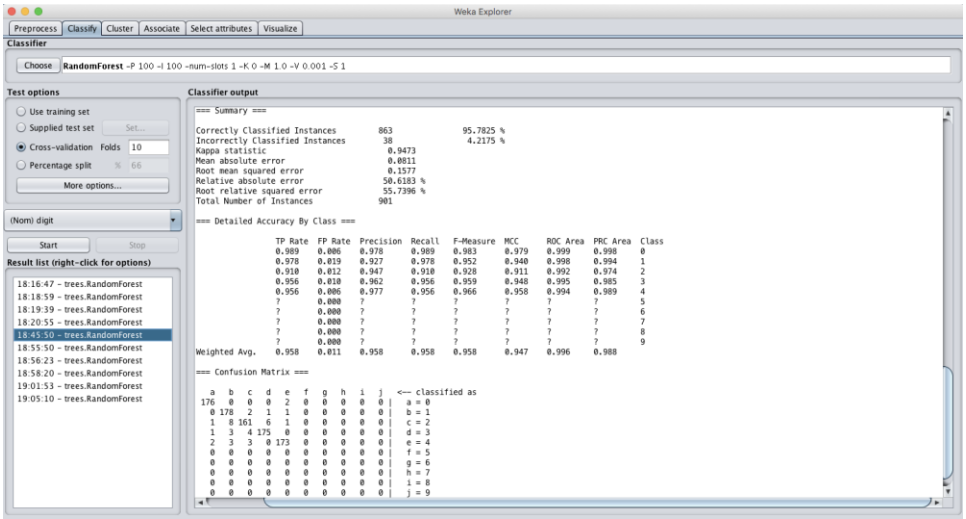


Fig: (e) 0s, 1s, 2s, 3s, 4s

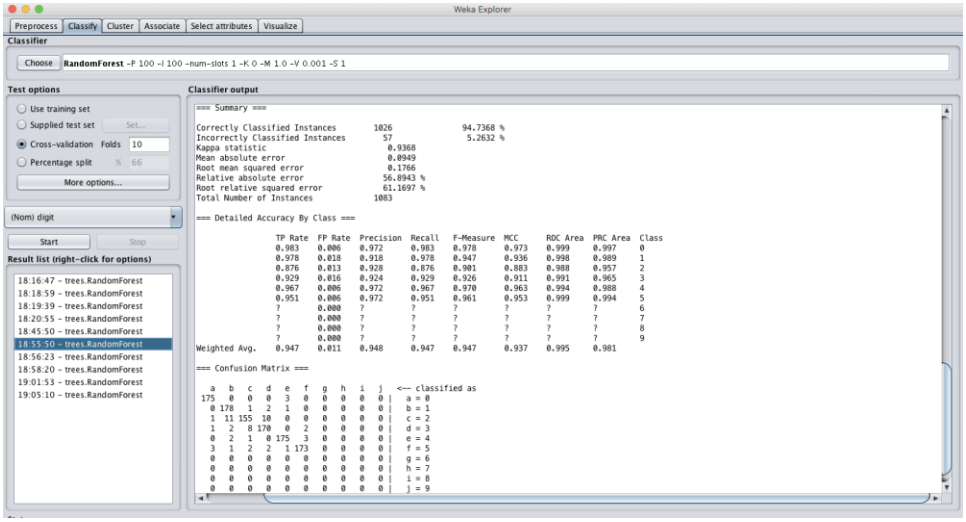


Fig: (f) 0s, 1s, 2s, 3s, 4s. 5s

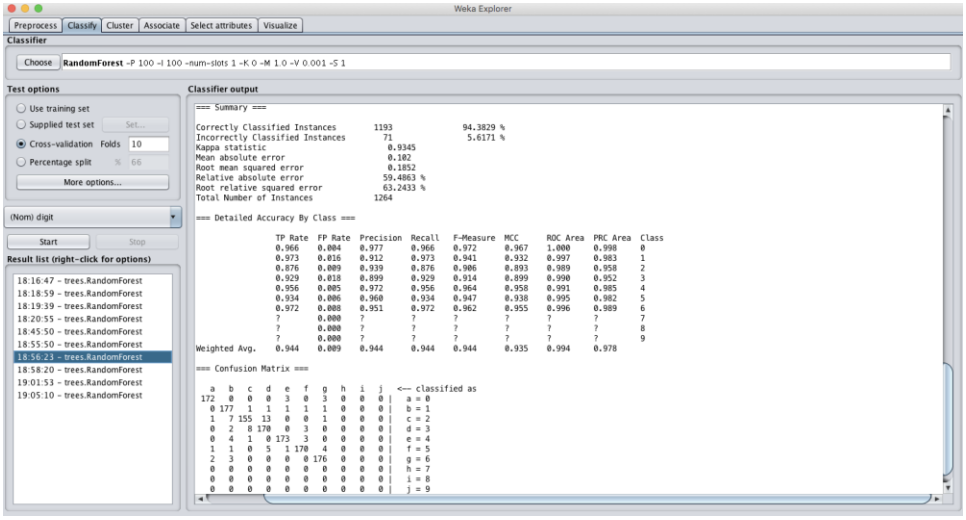


Fig: (g) 0s, 1s, 2s, 3s, 4s. 5s, 6s

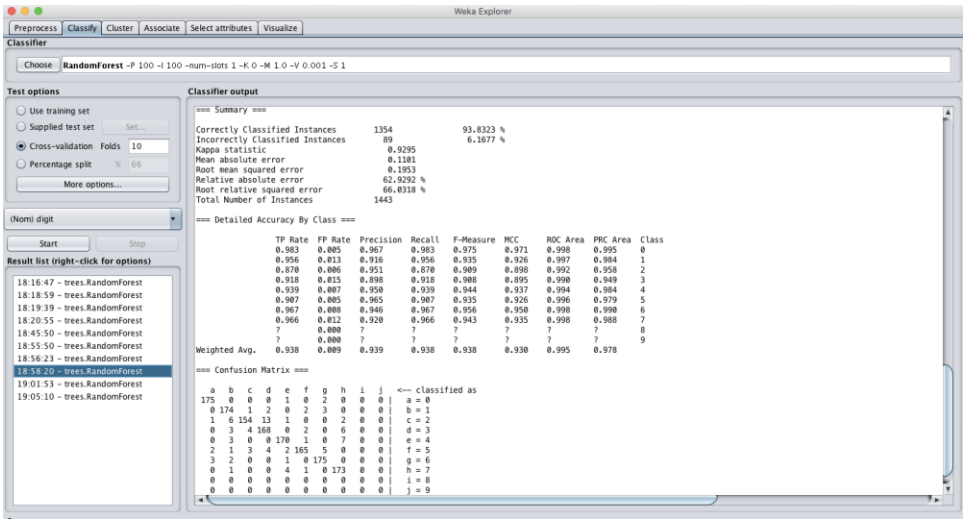


Fig: (h) 0s, 1s, 2s, 3s, 4s. 5s, 6s, 7s

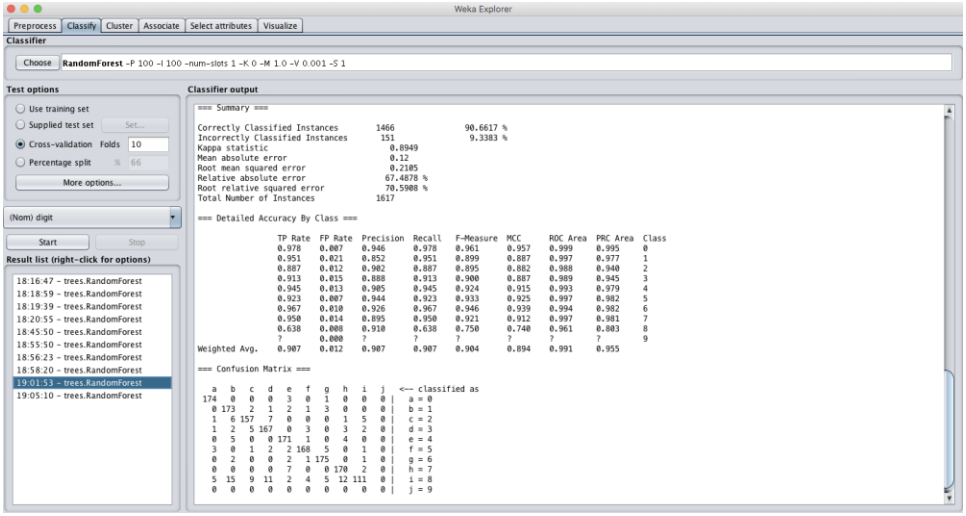


Fig: (i) 0s, 1s, 2s, 3s, 4s. 5s, 6s, 7s, 8s

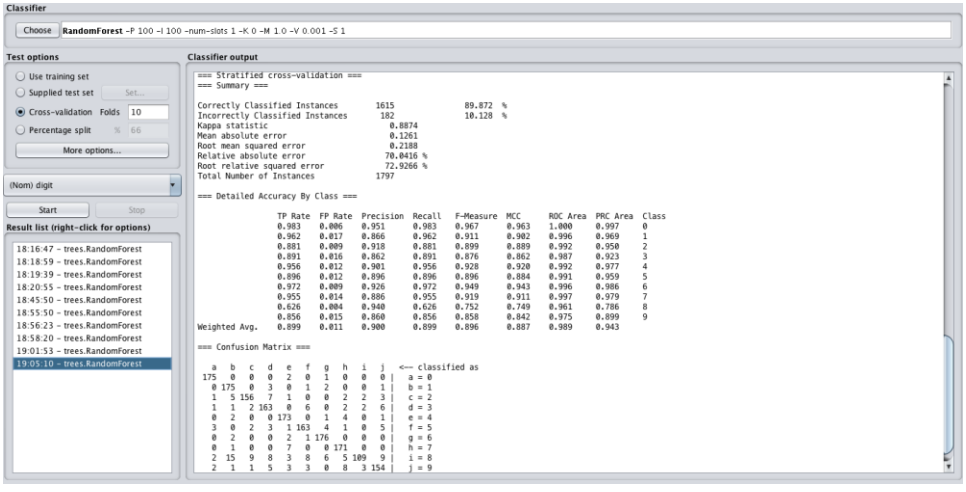


Fig: (j) 0s, 1s, 2s, 3s, 4s. 5s, 6s, 7s, 8s, 9s

Accuracy Table:

No. Of Digits Included (Index values)	2	3	4	5	6	7	8	9	10
Accuracy	99.44 %	98.51 %	97.36 %	95.78 %	94.74 %	94.38 %	93.83 %	90.66 %	89.88 %

Explanation: Impact when the digits gets added:

For 0s --> 100%

For 1s --> 99.44% (Adding 1's to dataset decreases the accuracy by 0.56% from earlier)

For 2s --> 98.51% (Adding 2's to dataset decreases the accuracy by 0.93% from the last changes)

For 3s --> 97.36% (Adding 3's to dataset decreases the accuracy by 1.15% from the last changes)

For 4s --> 95.78% (Adding 4's to dataset decreases the accuracy by 1.58% from the last changes)

For 5s --> 94.74% (Adding 5's to dataset decreases the accuracy by 1.04% from the last changes)

For 6s --> 94.38% (Adding 6's to dataset decreases the accuracy by 0.36% from the last changes)

For 7s --> 93.83% (Adding 7's to dataset decreases the accuracy by 0.55% from the last changes)

For 8s --> 90.66% (Adding 8's to dataset decreases the accuracy by 3.17% from the last changes)

For 9s --> 89.88% (Adding 9's to dataset decreases the accuracy by 0.78% from the last changes)

h) Suppose you were given a handwritten note and you scanned it into a pixel-based image. What kind of pre-processing steps do you imagine need to be performed before you can apply a trained character classifier?

Answer: The importance of the preprocessing stage of a character recognition system lies in its ability to remedy some of the problems that may occur due to some of the factors presented in the given documentation. After learning the concepts through various sources on google, I came with the following set of steps as mentioned below:

1)Using Image enhancement techniques: Image enhancement improves the quality of images for human perception by removing noise, reducing blurring, increasing contrast and providing more detail.

2) Noise removal: By noise we refer to stochastic variations as opposed to deterministic distortions, such as shading or lack of focus. Noise in binary images takes the form of isolated pixels and the processing of removing this type of noise is called filling. In grey- level images or median filters and low-pass filters such as average or Gaussian blur filters proved to eliminate isolated pixel noise. Gaussian blur and average filters are a better choice to provide smooth texture to the image.

3) Skew detection/correction: Due to the possibility of rotation of the input image and the sensitivity of many document image analysis methods to rotation of the image, document skew should be corrected. Skew detection techniques can be roughly classified into the following groups: analysis of projection profile, Hough transform, connected components, clustering, and correlation between lines techniques.

4) Page segmentation: After image enhancement, noise removal and/or skew detection/correction, the next step in mixed content images or composite images

is to perform page segmentation in order to separate text from halftone images, lines, and graphs. The result of interest should be an image with only text; therefore, document/page segmentation. Document segmentation can be classified into three broad categories: top-down, bottom-up and hybrid techniques. The top-down methods recursively segment large regions in a document into smaller sub-regions. The segmentation stops when some criterion is met and the ranges obtained at that stage constitute the final segmentation results. On the other hand, the bottom-up methods start by grouping pixels of interest and merging them into larger blocks or connected components, such as characters which are then clustered into words, lines or blocks of text. The hybrid methods are the combination of both top-down and bottom-up strategies.

5) Character segmentation: Well, in this case there are many techniques developed for character segmentation and most of them are script specific and may not work with other scripts. Even in printed handwritten documents, character segmentation is required due to touching of characters when written by hand.

6) Image size normalization: The result from the character segmentation stage provides isolated characters which are ready to be passed into the feature extraction stage; therefore, the isolated characters are normalized into a specific size, decided empirically or experimentally depending on the application and the feature extraction or classification techniques used, then features are extracted from all characters with the same size in order to provide data uniformity.

7) Morphological processing: Segmentation results may cause some pixels to be removed producing holes to some parts of the images; this could be seen from characters having some holes in them where some of the pixels were removed during thresholding. Larger holes can cause characters to break into two or more parts/objects. On the other hand, the opposite can also be true, as segmentation can join separate objects making it more difficult to separate characters; these solid objects resemble blobs and are hard to interpret. The solution to these problems is Morphological Filtering. Useful techniques include erosion and dilation, opening and closing, outlining, and thinning and skeletonisation.

- Erosion and dilation: Erosion makes an object smaller by removing or eroding away the pixels on its edges; however, dilation makes an object larger by adding pixels around its edges.
- Opening and closing: The basic effect of an opening is somewhat like erosion in that it tends to remove some of the foreground pixels from the edges of regions of foreground pixels. However, it is less destructive than erosion in general. Closing is similar in some ways to dilation in that it tends to enlarge the boundaries of foreground regions in an image, but it is less destructive of the original boundary shape.
- Outlining: Outlining is a type of edge detection; it only works for binary images but produces better results than regular edge detectors.
- Thinning and skeletonisation: Skeletonisation is a process for reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while removing most of the original foreground pixels.

After performing the above pre-processing steps, a better result can be achieved after applying the classifier algorithm.

But I would like to conclude the above answer by stating that most of preprocessing techniques are application-specific and not all preprocessing techniques have to be applied to all applications. Each application may require different preprocessing techniques depending on the different factors that may affect the quality of its images, such as those introduced during the image acquisition stage. Image manipulation/enhancement techniques do not need to be performed on an entire image since not all parts of an image is affected by noise or contrast variations; therefore, enhancement of a portion of the original image maybe more useful in many situations. This is obvious when an image contains different objects which may differ in their brightness or darkness from the other parts of the image; thereby, when portions of an image can be selected, either manually or automatically according to their brightness such processing can be used to bring out local detail.