Project 1

1. Image Preprocessing and Basic Operators

1.1 Edge Detection

Sobel operator:

In image processing, a kernel(or called mask/convolution matrix) is a small matrix useful for blurring, sharpening, embossing, edge-detection, and more. This is accomplished by means of convolution between a kernel and an image.

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. The computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

A is the source image, and **Gx** and **Gy** are two images which at each point contain the horizontal and vertical derivative approximations respectively.

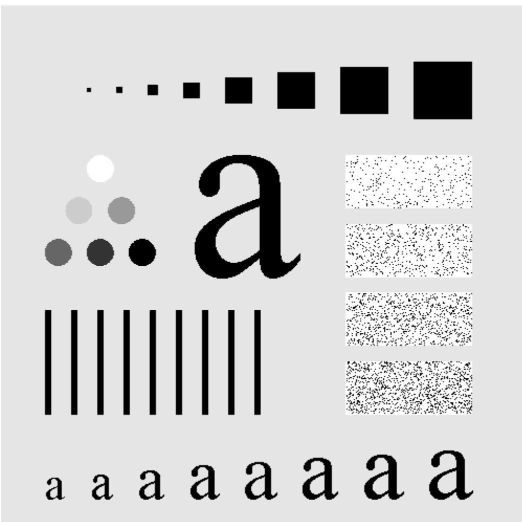
At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

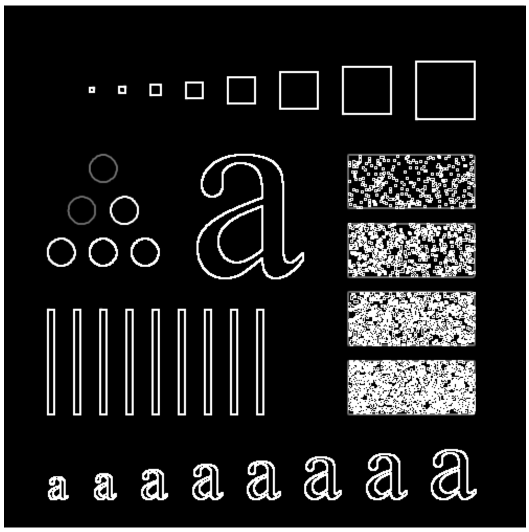
After getting the G, we set the value of center pixel A to G.

As we scan the image, we do the convolution over the image.

Input and Output:



Input Image



Sobel Operator

As shown in the original image, there are many noises, and the output shows that sobel operator can not handle the noise problem. So we need to find a better solution when there are noises in the image.

1.2 Noise Cancellation

Mean filtering:

The idea of mean filtering is simply to replace each pixel value in an image with the mean (‘average’) value of its neighbours, including itself.

Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighbourhood to be sampled when calculating the mean.

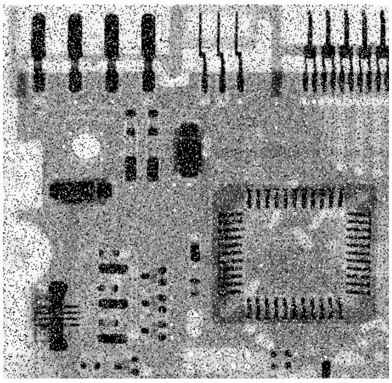
Often a 3×3 square kernel(As shown below) is used, although larger kernels (e.g. 5×5 squares) can be used for more severe smoothing.

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

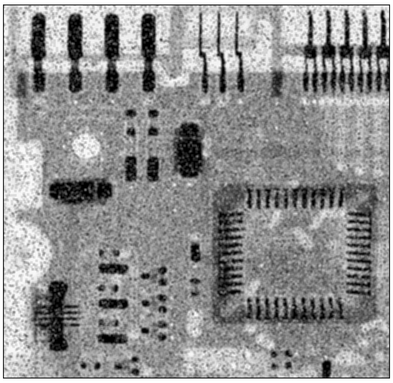
Median filtering:

The median filter replaces a target pixel's value with the median value of the neighbouring pixels(e.g. 3×3 squares)

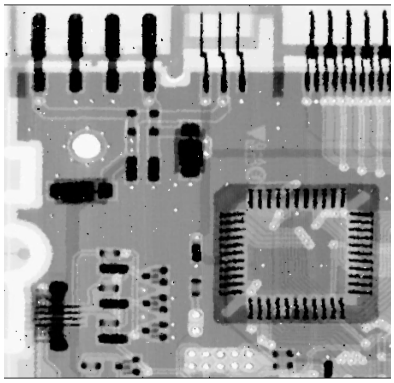
Input and Output:



Input Image



Mean Filter



Median Filter

The mean filter has effect to cancel noises in a image, but the result got from mean filter is not good enough.

The median filter is effective for cancelling noises. As we can see from the output of median filter, noises are surely get rid of by applying median filtering.

1.3 Image Enhancement

Laplacian Filter:

```
 0  -1   0
-1   5  -1
 0  -1   0
```

Laplacian filters can enhance images in all directions equally, it will enhance the edges on the original image. Similar to **1.1**

Input and Output:

Input Image

Enhanced Filter

Laplacian Filter can sharpen the image, clarifies the blur images.

2. Mining Image Data

2.1 Mining Space Images

① Set a threshold manually or generate a threshold from the information of the input image by using Otsu's mathod.
② Compare each single pixel with the threshold iteratively. If the value of pixel is larger than threshold, set the value of that pixel to 255, otherwise, set the value of that pixel to 0. Completing this process will get the final result.

Threshold - Otsu's Method:

Otsu's method is used to automatically perform clustering-based image thresholding, or, the reduction of a gray level image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram, it then calculates the optimum threshold separating the two classes so that their combined spread is minimal, or equivalently, so that their inter-class variance is maximal.

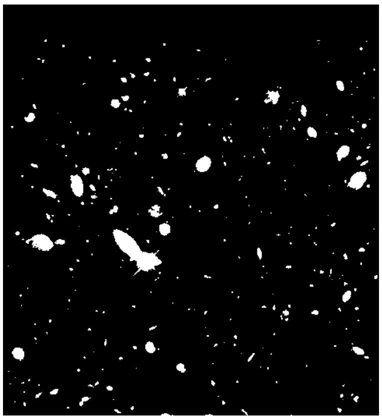
Weights W1 and W0 are the probabilities of the two classes separated by a threshold .
N1 is the number of pixel of the foreground object.
N2 is the number of pixel of the background.
M is the width and N is the height of the image.
M * N is the number of whole pixels in image.
μi is the mean value of grayscale in each class(foreground image and background).
μ is the mean value of grayscale for the whole image.
g is the intra-class variance.

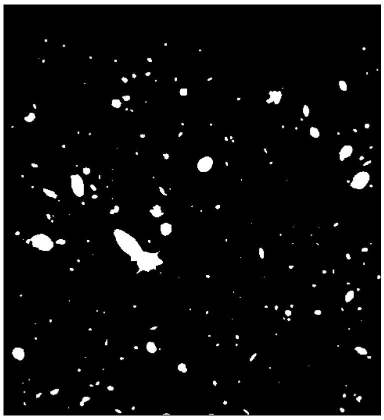
$$\omega_0 = \frac{N_0}{M * N}$$
$$\omega_1 = \frac{N_1}{M * N}$$
$$N_0 + N_1 = M * N$$
$$\omega_0 + \omega_1 = 1$$
$$\mu = \omega_0 * \mu_0 + \omega_1 * \mu_1$$
$$g = \omega_0(\mu_0 - \mu)^2 + \omega_1(\mu_1 - \mu)^2$$

Algorithm:
1. Count number of each gray scale value.
2. Calculate the probability of each gray scale.
3. Step through all possible thresholds t = 1, maximum = 255
    1. calculate ωi and ωi * μi
    2. Calculate μi
    3. Calculate g
    4. If the g is the maximum, then threshold = t
4. Return threshold

Input and Output:


Input Image


Otsu Only


Mean + Otsu

As shown at the above results, thresholding without apply a mean filter will get the map of all galaxies from the input image, including those tiny(far away from us) galaxies.

With mean filter applied before thresholding, the result map contains only large(closer-to-us) galaxies without those tiny galaxies.

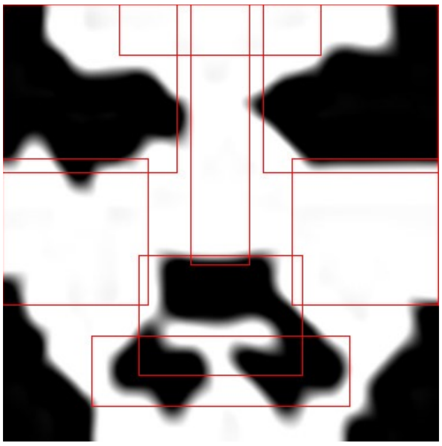
Mean filter can smooth the image, blur those tiny galaxies from a small white spot to be a small area(3*3 in my implementation) of dark gray, thus the threshold could ignore them.

Otsu's method also played a important role in my implementation. Instead of giving a threshold manually, Otsu generated a threshold automatically by the gray-level of the image. This will give a threshold which is more appropriate for the input image than those thresholds which are setup manually.

2.2 Face Detection

Feature Selection:

The original training image is in grayscale, the first step is to turn the image to black-white map using Otsu's method as we mentioned in the Question 2.1. Then, as we can see in FEATURE MAP as shown below, based on common understanding of human face, we select eight features: left eye, forehead, right eye, nose bridge, left cheek and right cheek, nose and mouth.



I use the proportion of the black area of the square to measure each feature.

For example: On the left corner is the left eye square, the proportion of the black area account for 70% of its square area. So we record this feature as 0.70.

Feature Extraction:

As every image is a matrix of 19 * 19 pixels, we could manually define every feature as a square. Then we calculate the proportion of black area of the square.
For example: On the left corner is the left eye square, it starts from row = 0 and column = 0 to row = 6 and column = 6.

Result:

Training:

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     training
Instances:    2900
Attributes:   9
              f1
              f2
              f3
              f4
              f5
              f6
              f7
              f8
              class
Test mode:    evaluate on training data
```

```
=== Classifier model (full training set) ===

Naive Bayes Classifier

                Class
Attribute           0       1
                (0.5)   (0.5)
===============================
f1
  mean           0.5827  0.6772
  std. dev.      0.3554  0.1787
  weight sum       1450    1450
  precision      0.0204  0.0204

f2
  mean           0.5188   0.113
  std. dev.      0.4272  0.2183
  weight sum       1450    1450
  precision         0.1     0.1

f3
  mean           0.5946   0.645
  std. dev.      0.3513  0.1723
  weight sum       1450    1450
  precision      0.0204  0.0204

f4
  mean           0.5312  0.2055
  std. dev.      0.3541  0.2118
  weight sum       1450    1450
  precision      0.0556  0.0556

f5
  mean           0.5321  0.2573
  std. dev.      0.3594  0.2359
  weight sum       1450    1450
  precision      0.0278  0.0278

f6
  mean           0.5405  0.2425
  std. dev.      0.3579  0.2343
  weight sum       1450    1450
  precision      0.0238  0.0238

f7
  mean           0.5484  0.5971
  std. dev.       0.407  0.2935
  weight sum       1450    1450
  precision      0.0833  0.0833

f8
  mean           0.5542  0.7108
  std. dev.      0.3809  0.2648
  weight sum       1450    1450
  precision        0.05    0.05
```

Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances	2637	90.931 %
Incorrectly Classified Instances	263	9.069 %
Kappa statistic	0.8186	
Mean absolute error	0.1108	
Root mean squared error	0.2653	
Relative absolute error	22.1641 %	
Root relative squared error	53.0624 %	
Total Number of Instances	2900	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.903	0.084	0.915	0.903	0.909	0.819	0.962	0.942	0
	0.916	0.097	0.904	0.916	0.910	0.819	0.962	0.966	1
Weighted Avg.	0.909	0.091	0.909	0.909	0.909	0.819	0.962	0.954	

=== Confusion Matrix ===

```
    a    b   <-- classified as
 1309  141 |    a = 0
  122 1328 |    b = 1
```

Test:

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes
Relation: training
Instances: 2900
Attributes: 9
f1
f2
f3
f4
f5
f6
f7
f8
class
Test mode: user supplied test set: size unknown (reading incrementally)

=== Classifier model (full training set) ===

Naive Bayes Classifier

```
              Class
Attribute         0      1
```

```
               (0.5)   (0.5)
===============================
f1
  mean          0.5827  0.6772
  std. dev.     0.3554  0.1787
  weight sum      1450    1450
  precision     0.0204  0.0204

f2
  mean          0.5188   0.113
  std. dev.     0.4272  0.2183
  weight sum      1450    1450
  precision        0.1     0.1

f3
  mean          0.5946   0.645
  std. dev.     0.3513  0.1723
  weight sum      1450    1450
  precision     0.0204  0.0204

f4
  mean          0.5312  0.2055
  std. dev.     0.3541  0.2118
  weight sum      1450    1450
  precision     0.0556  0.0556

f5
  mean          0.5321  0.2573
  std. dev.     0.3594  0.2359
  weight sum      1450    1450
  precision     0.0278  0.0278

f6
  mean          0.5405  0.2425
  std. dev.     0.3579  0.2343
  weight sum      1450    1450
  precision     0.0238  0.0238

f7
  mean          0.5484  0.5971
  std. dev.      0.407  0.2935
  weight sum      1450    1450
  precision     0.0833  0.0833

f8
  mean          0.5542  0.7108
  std. dev.     0.3809  0.2648
  weight sum      1450    1450
  precision       0.05    0.05
```

Time taken to build model: 0 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.01 seconds

=== Summary ===

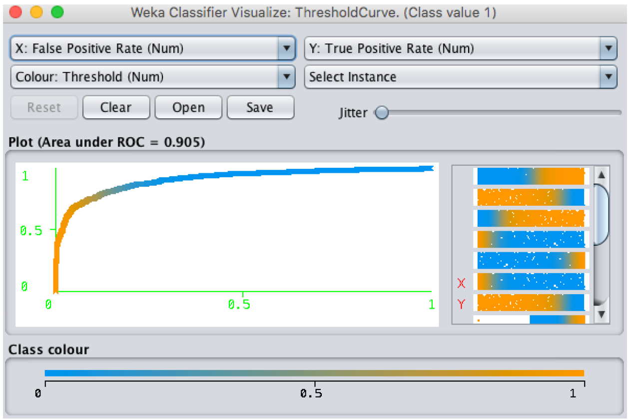
Correctly Classified Instances	2412	83.1151 %
Incorrectly Classified Instances	490	16.8849 %
Kappa statistic	0.6623	
Mean absolute error	0.1863	
Root mean squared error	0.3685	
Relative absolute error	37.2606 %	
Root relative squared error	73.7048 %	
Total Number of Instances	2902	

=== Detailed Accuracy By Class ===

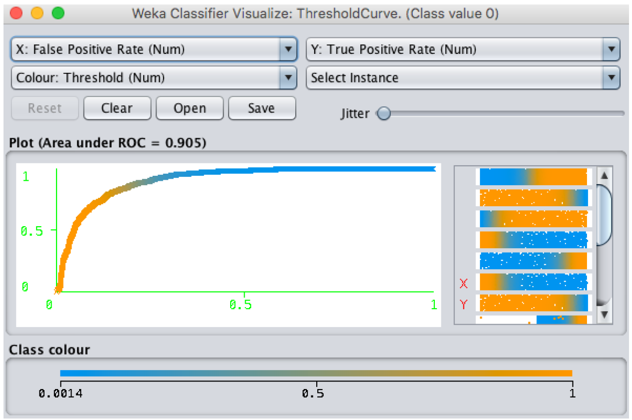
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.874	0.212	0.805	0.874	0.838	0.665	0.905	0.864	0
	0.788	0.126	0.862	0.788	0.824	0.665	0.905	0.916	1
Weighted Avg.	0.831	0.169	0.834	0.831	0.831	0.665	0.905	0.890	

=== Confusion Matrix ===

a	b	<-- classified as
1268	183 |	a = 0
307	1144 |	b = 1



Face Classifier



Non-face classifier

The ROC curves above are for the face classifier and non-face classifier with different threshold. The area under ROC is 0.905, which is acceptable.

The accuracy of training set is 90.931%, meanwhile the accuracy of testing set is 83.1151%, which can be found from the result.

Ostu's Method still works well for turning the images to black and white, which are used for getting features.

Overall, it is not a bad result, but still can be improved. Mean and standard deviation of particular regions as features may not be the best choice. Mean will count all pixels in that region while some of the pixel may not contains useful informations for classifier. Using median may get better solution. Also, the training set may not good enough for representing variety of faces, and 19*19 image size can not provide enough information for training.

3 Data Mining Using Extracted Features

Data Mining Using Extracted Features

What we need to do is as follow:
1. Put the features together in a file.
2. Generate labels for each number.
3. Generate labels for each feature.
4. Make the feature can be used in Weka.
5. Train a C4.5/J48 decision tree classifier using the dataset.
6. Constructed the decision tree and discuss about it.
7. Compare the result of using all feature with using only morphological feature.

Explain Methods:

The first 4 tasks are pure engineering problems that related to string feature handle ability.

As for these 4 tasks, we use shell script language to sort out the features.
The major tools we used are as follow:

sed
For example:
sed 's/ \+/ /g' mfeat-fac
This could replace multi-whitespace in the input file with one whitespace.

paste
For example:
paste mfeat_fac mfeat_kar > new_file
This example could paste two files column by column into the new_file.

The format for Weka is as follow:
@relation numbers
@attribute att01 numeric
...
@data

Implementation:

Running the run.sh script in Q3 folder, the sorted file will be generated in the result folder.

./result/
-mfeat-train.arff
-mfeat-test.arff
-mfeat-mor-train.arff
-mfeat-mor-test.arff

mfeat-train.arff and mfeat-test.arff contain all 649 features.
Each of the file contains 100 records for '0' ~ '9'. So the total records in each file is 1,000.

mfeat-mor-train.arff and mfeat-mor-test.arff contain only morphological
features. Each of the file contains 100 records for '0' ~ '9'. So the total records in each file is 1,000.

Results Analysis:

In the implementation section, we divide the data into training set and test set. As we use training set to feed the J48 classifier in Weka, the result are as follow:

=== Classifier model (full training set) ===

J48 pruned tree

```
------------------

pix48 <= 0
|   pix52 <= 1.524258
|   |   c108 <= 4: 4 (2.0)
|   |   c108 > 4: 1 (93.0/1.0)
|   pix52 > 1.524258
|   |   c185 <= 1052
|   |   |   pix229 <= 1
|   |   |   |   pix91 <= 3: 5 (3.0)
|   |   |   |   pix91 > 3: 2 (3.0)
|   |   |   pix229 > 1
|   |   |   |   c48 <= 8: 1 (2.0)
|   |   |   |   c48 > 8: 4 (98.0/1.0)
|   |   c185 > 1052
|   |   |   f71 <= 0.478731
|   |   |   |   c198 <= 1049
|   |   |   |   |   f11 <= 1.73218
|   |   |   |   |   |   c101 <= 713: 1 (6.0/1.0)
|   |   |   |   |   |   c101 > 713: 7 (97.0/1.0)
|   |   |   |   |   f11 > 1.73218: 3 (13.0)
|   |   |   |   c198 > 1049
|   |   |   |   |   pix201 <= 5
|   |   |   |   |   |   pix49 <= 3
|   |   |   |   |   |   |   zer32 <= 1: 7 (3.0/1.0)
|   |   |   |   |   |   |   zer32 > 1: 2 (96.0/1.0)
|   |   |   |   |   |   pix49 > 3: 3 (4.0)
|   |   |   |   |   pix201 > 5
|   |   |   |   |   |   pix116 <= 5: 3 (65.0)
|   |   |   |   |   |   pix116 > 5: 5 (4.0)
|   |   |   f71 > 0.478731
|   |   |   |   pix116 <= 2
|   |   |   |   |   pix98 <= 0: 3 (16.0)
|   |   |   |   |   pix98 > 0: 5 (2.0)
|   |   |   |   pix116 > 2: 5 (91.0)
pix48 > 0
|   pix50 <= 0: 0 (100.0)
|   pix50 > 0
|   |   c106 <= 4: 6 (99.0/1.0)
|   |   c106 > 4
|   |   |   pix48 <= 1
|   |   |   |   pix205 <= 2
|   |   |   |   |   pix111 <= 0: 1 (2.0/1.0)
|   |   |   |   |   pix111 > 0: 9 (99.0)
|   |   |   |   pix205 > 2
|   |   |   |   |   pix79 <= 4: 6 (2.0)
|   |   |   |   |   pix79 > 4: 8 (5.0)
|   |   |   pix48 > 1: 8 (95.0)

Number of Leaves  :   24

Size of the tree :   47
```

As we can see from the decision tree. It does not use all 649 features, but it picks 24 features that could already separate 10 numbers.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
100	0	0	0	0	0	0	0	0	0	a = 0
0	100	0	0	0	0	0	0	0	0	b = 1
0	1	98	0	0	0	0	1	0	0	c = 2
0	0	0	98	1	0	0	1	0	0	d = 3
0	0	0	0	99	0	1	0	0	0	e = 4
0	0	0	0	0	100	0	0	0	0	f = 5
0	0	0	0	0	0	100	0	0	0	g = 6
0	1	1	0	0	0	0	98	0	0	h = 7
0	0	0	0	0	0	0	0	100	0	i = 8
0	1	0	0	0	0	0	0	0	99	j = 9

This confusion matrix not only provide correct rate but also shows where the data is misclassified.

For example:
The last row shows the result of number '9', 99% of number '9' can be successfully classified. 1% is misclassified as number '2' for some reason.

Then we could use it to classify the test dataset.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
97	0	0	0	0	2	0	1	0	0	a = 0
0	92	0	3	3	0	0	2	0	0	b = 1
0	2	97	0	0	1	0	0	0	0	c = 2
0	0	2	87	1	7	0	3	0	0	d = 3
0	5	2	2	90	1	0	0	0	0	e = 4
0	0	0	5	2	93	0	0	0	0	f = 5
0	1	0	0	2	0	97	0	0	0	g = 6
0	1	2	4	0	0	0	93	0	0	h = 7
0	0	0	0	0	0	0	0	98	2	i = 8
0	5	0	0	1	2	0	0	0	92	j = 9

This is the result that we use classifier to classify the test dataset.
The accuracy remain above 90% overall. Although number '3' is frequently being classified as number '5'.

Then we use only morphological features to train the classifier.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
100	0	0	0	0	0	0	0	0	0	a = 0
0	95	0	0	1	0	1	3	0	0	b = 1
0	1	89	4	0	2	1	3	0	0	c = 2
0	3	4	74	11	6	0	2	0	0	d = 3
0	2	1	6	86	0	1	4	0	0	e = 4
0	0	19	10	5	66	0	0	0	0	f = 5
0	0	0	0	0	0	100	0	0	0	g = 6
0	1	1	0	2	0	0	96	0	0	h = 7
0	0	0	0	0	0	0	0	100	0	i = 8
0	0	0	0	0	1	99	0	0	0	j = 9

As we can observe from the confusion matrix, some of the numbers could be successfully classified. For example: number '1', '6' and '8'. Some of the numbers can mostly be classified, but for number '5' , it always misclassified as '2' and '3'. There is one particular case, number '9' could not be classified by morphological features, as it will be classified as '6'.

If we use this classifier to classify the test sample. We would gain a similar result.

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	<-- classified as
97	0	1	0	0	2	0	0	0	0	a = 0
0	93	1	0	3	0	0	3	0	0	b = 1
0	0	70	4	5	16	1	4	0	0	c = 2
0	0	8	53	21	8	0	10	0	0	d = 3
0	5	3	9	65	4	0	14	0	0	e = 4
0	0	15	15	2	66	0	2	0	0	f = 5
0	1	0	1	0	0	98	0	0	0	g = 6
0	3	6	3	10	0	0	78	0	0	h = 7
0	0	0	0	0	0	2	0	98	0	i = 8
0	1	3	0	0	1	95	0	0	0	j = 9

3.3 Conclusion:

Although by using morphological features, the J48(decision tree) classifier could successfully classify most of the hand written numbers, but for particular case such as number '9', the shape information does not sufficiently provide a good result. If we use all 649 features, although the classifier does not necessarily use all the features, it only select a set of features which could also provide a reasonably good result.