TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

**SCHOOL OF ENGINEERING AND COMPUTER SCIENCE**

# Assignment Cover Sheet

**Full Name:** Tao Shi

**Student ID:** 300409943 **Course:** COMP422

**Assignment No.:** 1 **Due Date:** 22 August at 11:59pm

## Part 1: Preprocessing

### 1.1 Edge Detection

The procedure of Sobel edge detection is as follows:
1. Change the TIFF format of original image to numpy.ndarray in Python for further processing;
2. Convolution process by using row mask and column mask respectively;

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

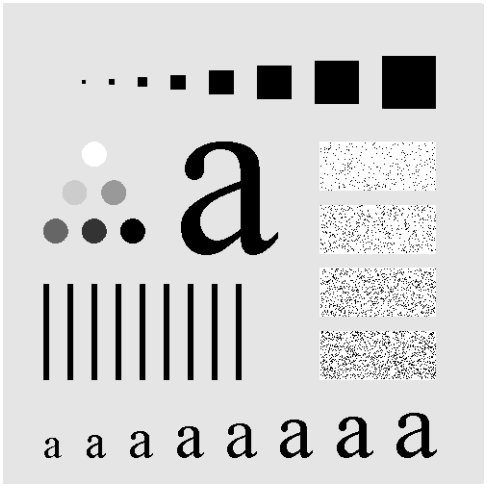
row mask

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

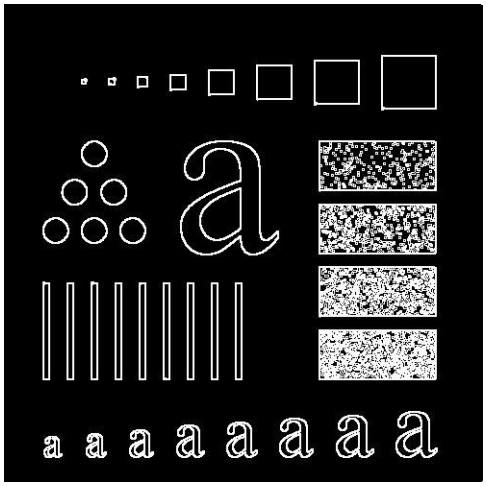
column mask

3. Assume the results from the two masks are s1 and s2, then the edge magnitude is $\sqrt{s1^2 + s2^2}$;
4. Apply thresholding (10 in the case) to enhance the edge;
5. Save the processed image as expected format (jpg in the case).

The original and the resulting image are as follows:



**The original image**



**The resulting image**

According to the resulting image, the edges from Sobel operator can represent the outline of squares, circles and letters of different sizes, even small objects.
However, the operator is less sensitive to noisy, for it is difficult to find a threshold which removes all noise pixels and at the same time retains the edges of the objects. On the other hand, if we hope that only corners are detected, we have to do a local analysis of detected interest points to determine the real corners. Obviously the Sobel operator cannot realize the goal.

### 1.2 Noise Cancellation

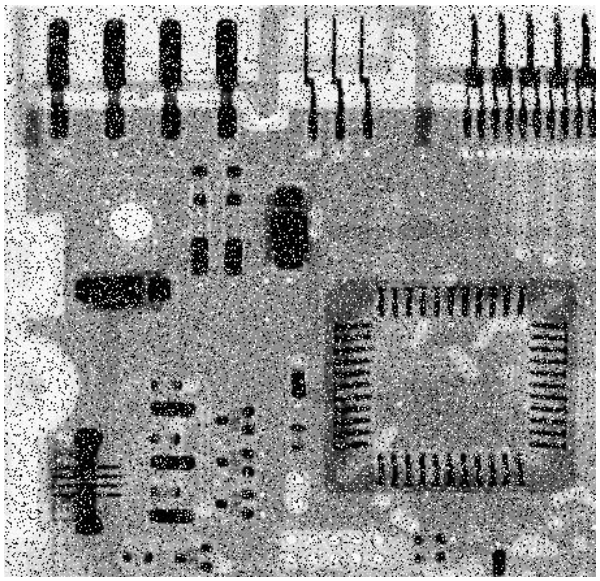
In order to eliminate the impulse noise in the original image, I applied *median filter* which replace the central pixel with the median value of the neighborhood and *mean filter* which is based on convolution, and the convolution mask is as follows:

1/9 1/9 1/9

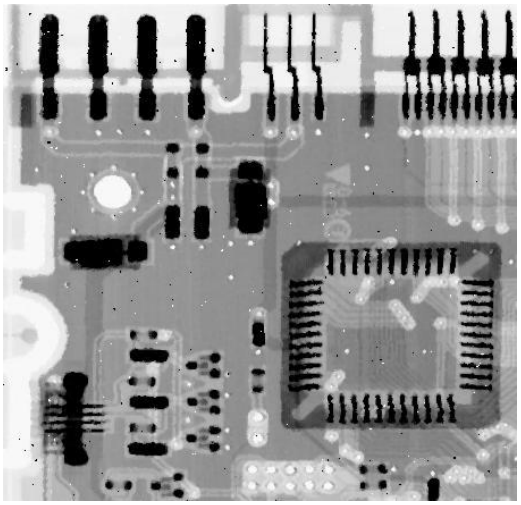
1/9 1/9 1/9

1/9 1/9 1/9

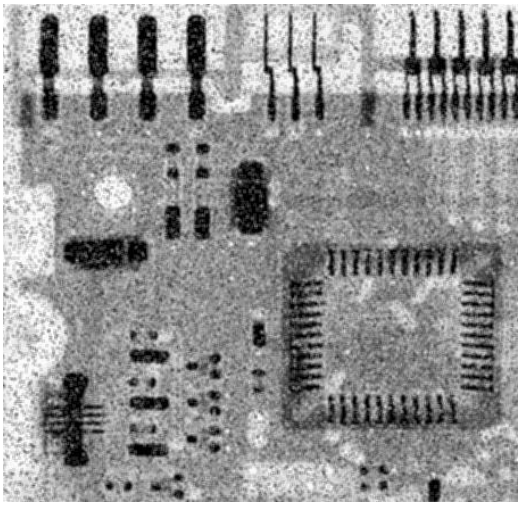
The original and the resulting images are as follows:



**The original image**



**The image processed by median filter**



**The image processed by mean filter**

Obviously, the two filters both reduce the impulse noise, but the effectiveness of median filter is much better.

The images containing impulse noise have dark pixels in bright regions and bright pixels in dark regions. This type of noise could be caused by analog-to-digital converter errors, bit errors in transmission and so on.

Because median filter can replace the central pixel of noisy image with the median

value of the neighborhood, the maximum value (bright noisy pixels) and minimum value (dark noisy pixels) will be cancelled, which is more effective than mean filter.

### 1.3 Image Enhancement

Because Laplacian filters can enhance images in all directions equally, I tried two typical convolution masks as Laplacian filters to deblur the original images. The two typical convolution operators and relevant resulting images are as follows:

**The original image**

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

| | | |
|---|---|---|
| 1 | -2 | 1 |
| -2 | 5 | -2 |
| 1 | -2 | 1 |

**The images processed by two typical Laplacian filter**

The convolution process is:
1. overlay the mask on the original image;
2. Perform the vector inner product operation;
3. Slide the mask cross the image, and perform the operation at each pixel location;
4. Put the summed result to the central pixel of the output image.
Obviously, the two filters both enhance the blurry image, but the effectiveness of left Laplacian filter is better.

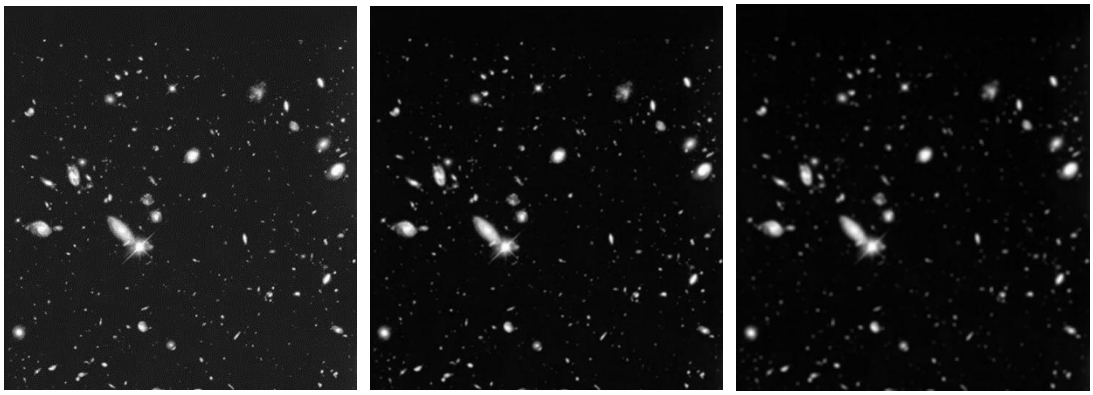
## Part 2: Mining Image Data

### 2.1 Mining Space Images

Firstly 2 kinds of the mean filter are applied, and the convolution masks are as follows:

| | |
|---|---|
| 1/9 1/9 1/9 | 1/25 1/25 1/25 1/25 1/25 |
| 1/9 1/9 1/9 | 1/25 1/25 1/25 1/25 1/25 |
| 1/9 1/9 1/9 | 1/25 1/25 1/25 1/25 1/25 |
| **3*3 mask** | 1/25 1/25 1/25 1/25 1/25 |
| | 1/25 1/25 1/25 1/25 1/25 |
| | **5*5 mask** |

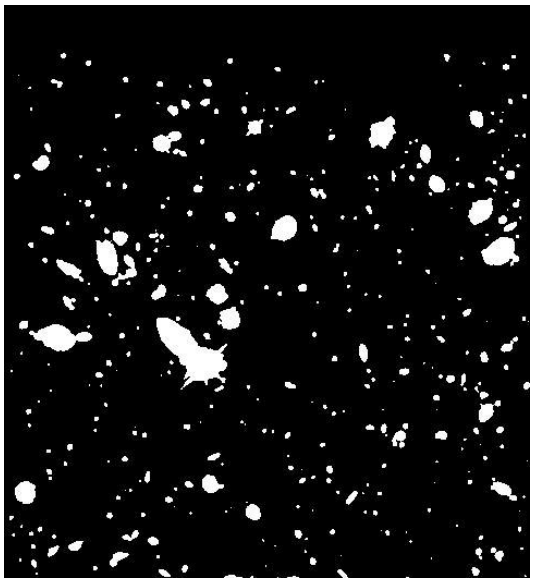
We can see that the image after 5*5 mask smoothing is more blurred than the image after 3*3 mask smoothing.



**The original image After 3*3 mask smoothing After 5*5 mask smoothing**

Then the image after 3*3 mask smoothing is applied different thresholding values, such as 50, 100 and 200. According to following resulting images, better performance is achieved (only large galaxies) when the thresholding is set as 200.
So greater thresholding value is helpful to detect only large galaxies, however if the thresholding value is too large, the detected objects will become too small. We can confirm proper thresholding value by testing and observation.
Lastly compare it with the resulting image after 5*5 mask smoothing and without

smoothing, whose thresholding values are both 200.



**The resulting image (after smoothing thresholding = 50)**

**The resulting image (after smoothing thresholding = 200)**

**The resulting image (after 5*5 mask smoothing thresholding = 200)**

**The resulting image (no smoothing thresholding = 200)**

In the resulting image after 5*5 mask smoothing, more small galaxies are ignored, on the contrary there are more small galaxies in the resulting image without smoothing, which means larger smoothing mask is helpful to detect only large galaxies.

### 2.2 Face Detection

### Feature Extraction

The following nine features are extracted:

| Features | Extraction method | Part of image |
|---|---|---|
| The difference between the colors of two eyeballs | The absolute value of difference of the minimum of two matrix | Upper left and upper right (both are 9*9 matrixes) |
| The difference between the colors of two eye whites | The absolute value of difference of the maximum of two matrix | Upper left and upper right (both are 9*9 matrixes) |
| The general difference between two sub-images | The absolute value of difference of the total of two matrix | Upper left and upper right (both are 9*9 matrixes) |
| The brightness of the nose tip | The maximum of the matrix | In the middle (5*5 matrix) |
| The location of the nose tip | the index of the maximum of the matrix % 5 | In the middle (5*5 matrix) |
| The difference of central information | The absolute value of difference of the median of two matrix | lower left and lower right (both are 9*9 matrixes) |
| The difference of horizontal and vertical information | $\left\|\frac{1}{16}\sum x_{ij} - \frac{1}{16}\sum x'_{ij}\right\|$ | lower left and lower right (both are 9*9 matrixes) |
| The difference of diagonal information | $\left\|\frac{1}{16}\sum y_{ij} - \frac{1}{16}\sum y'_{ij}\right\|$ | lower left and lower right (both are 9*9 matrixes) |
| The difference of other information | $\left\|\frac{1}{48}\sum z_{ij} - \frac{1}{48}\sum z'_{ij}\right\|$ | lower left and lower right (both are 9*9 matrixes) |

**Creating Tabular Data Sets**

The training data set and test data set are created as ARFF format to further steps. The file containing training examples is named as 'train'. 1450 face images and 1450 non-face images are converted to 2900 instance vectors, including 9 extracted features and 1 class label (1 means face and 0 means nonface).
The file containing test examples is named as 'tset'. 1451 face images and 1451 non-face images are converted to 2902 instance vectors, including 9 extracted features and 1 class label (1 means face and 0 means nonface).

**Object Classification and Performance Evaluation**

I use naive Bayes in WEKA to implement the object classification and performance evaluation on **training data** as follows:

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes

Relation: train

Instances: 2900

Attributes: 10

eyeball_diff

eyewhite_diff

upper_diff

noselip_bright

noselip_loc

lowerc_diff

lowerx_diff

lowery_diff

lowerz_diff

class

**Test mode: evaluate on training data**

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

Naive Bayes Classifier

| | Class | |
|---|---|---|
| Attribute | 0 | 1 |
| | (0.5) | (0.5) |
| ========== | ========== | ========== |
| eyeball_diff | | |
| mean | 17.041 | 11.2468 |
| std. dev. | 27.7719 | 12.4093 |
| weight sum | 1450 | 1450 |
| precision | 1.877 | 1.877 |
| | | |
| eyewhite_diff | | |
| mean | 20.6491 | 13.3369 |
| std. dev. | 29.7214 | 14.7488 |
| weight sum | 1450 | 1450 |
| precision | 1.6894 | 1.6894 |
| | | |
| upper_diff | | |
| mean | 1541.7196 | 1477.4747 |
| std. dev. | 2023.1393 | 1290.8671 |
| weight sum | 1450 | 1450 |
| precision | 8.5198 | 8.5198 |
| | | |
| noselip_bright | | |
| mean | 128.0087 | 202.6968 |
| std. dev. | 67.0354 | 46.1088 |
| weight sum | 1450 | 1450 |

| | | |
|---|---|---|
| precision | 1.0159 | 1.0159 |
| noselip_loc | | |
| mean | 1.7697 | 2.1172 |
| std. dev. | 1.6058 | 0.6998 |
| weight sum | 1450 | 1450 |
| precision | 1 | 1 |
| lowerc_diff | | |
| mean | 29.7952 | 26.9809 |
| std. dev. | 38.1838 | 23.1606 |
| weight sum | 1450 | 1450 |
| precision | 1.478 | 1.478 |
| lowerx_diff | | |
| mean | 11.8936 | 10.9389 |
| std. dev. | 15.9702 | 8.9051 |
| weight sum | 1450 | 1450 |
| precision | 0.1772 | 0.1772 |
| lowery_diff | | |
| mean | 10.8113 | 12.8095 |
| std. dev. | 13.7685 | 10.5644 |
| weight sum | 1450 | 1450 |
| precision | 0.1635 | 0.1635 |
| lowerz_diff | | |
| mean | 28.154 | 29.5328 |
| std. dev. | 37.2841 | 23.6083 |
| weight sum | 1450 | 1450 |
| precision | 0.1479 | 0.1479 |

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.05 seconds

=== Summary ===

| | | |
|---|---|---|
| Correctly Classified Instances | 2263 | 78.0345 % |
| Incorrectly Classified Instances | 637 | 21.9655 % |
| Kappa statistic | 0.5607 | |
| Mean absolute error | 0.2413 | |

Root mean squared error 0.4123
Relative absolute error 48.2628 %
Root relative squared error 82.4565 %
Total Number of Instances 2900

=== Detailed Accuracy By Class ===

| | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---|---|---|---|---|---|---|---|---|---|
| | 0.668 | 0.108 | 0.861 | 0.668 | 0.753 | 0.575 | 0.880 | 0.863 | 0 |
| | 0.892 | 0.332 | 0.729 | 0.892 | 0.802 | 0.575 | 0.880 | 0.886 | 1 |
| Weighted Avg. | 0.780 | 0.220 | 0.795 | 0.780 | 0.778 | 0.575 | 0.880 | 0.875 | |

=== Confusion Matrix ===

| a | b | <-- classified as |
|---|---|---|
| 969 | 481 | a = 0 |
| 156 | 1294 | b = 1 |

As the task of face detection, the appropriate evaluation should be TPF, which is the measure of sensitivity of desired objects (face) correctly classified by the classifier.
TPF = TP/(TP+FN) = 969/(969+156) = 86.1%.
The overall classification accuracy on training data is 78.0345%.

Then I use naive Bayes in WEKA to implement the object classification and performance evaluation on **test data** as follows:

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes
Relation: train
Instances: 2900
Attributes: 10
eyeball_diff
eyewhite_diff
upper_diff
noselip_bright
noselip_loc
lowerc_diff
lowerx_diff
lowery_diff
lowerz_diff
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) |
| ==== | ==== | ==== |
| eyeball_diff | | |
| mean | 17.041 | 11.2468 |
| std. dev. | 27.7719 | 12.4093 |
| weight sum | 1450 | 1450 |
| precision | 1.877 | 1.877 |
| eyewhite_diff | | |
| mean | 20.6491 | 13.3369 |
| std. dev. | 29.7214 | 14.7488 |
| weight sum | 1450 | 1450 |
| precision | 1.6894 | 1.6894 |
| upper_diff | | |
| mean | 1541.7196 | 1477.4747 |
| std. dev. | 2023.1393 | 1290.8671 |
| weight sum | 1450 | 1450 |
| precision | 8.5198 | 8.5198 |
| noselip_bright | | |
| mean | 128.0087 | 202.6968 |
| std. dev. | 67.0354 | 46.1088 |
| weight sum | 1450 | 1450 |
| precision | 1.0159 | 1.0159 |
| noselip_loc | | |
| mean | 1.7697 | 2.1172 |
| std. dev. | 1.6058 | 0.6998 |
| weight sum | 1450 | 1450 |
| precision | 1 | 1 |
| lowerc_diff | | |
| mean | 29.7952 | 26.9809 |
| std. dev. | 38.1838 | 23.1606 |
| weight sum | 1450 | 1450 |
| precision | 1.478 | 1.478 |

lowerx_diff

| | | |
|---|---|---|
| mean | 11.8936 | 10.9389 |
| std. dev. | 15.9702 | 8.9051 |
| weight sum | 1450 | 1450 |
| precision | 0.1772 | 0.1772 |

lowery_diff

| | | |
|---|---|---|
| mean | 10.8113 | 12.8095 |
| std. dev. | 13.7685 | 10.5644 |
| weight sum | 1450 | 1450 |
| precision | 0.1635 | 0.1635 |

lowerz_diff

| | | |
|---|---|---|
| mean | 28.154 | 29.5328 |
| std. dev. | 37.2841 | 23.6083 |
| weight sum | 1450 | 1450 |
| precision | 0.1479 | 0.1479 |

Time taken to build model: 0.01 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.05 seconds

=== Summary ===

| | | |
|---|---|---|
| Correctly Classified Instances | 2279 | 78.532 % |
| Incorrectly Classified Instances | 623 | 21.468 % |
| Kappa statistic | 0.5706 | |
| Mean absolute error | 0.239 | |
| Root mean squared error | 0.4027 | |
| Relative absolute error | 47.7922 % | |
| Root relative squared error | 80.5393 % | |
| Total Number of Instances | 2902 | |

=== Detailed Accuracy By Class ===

| | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---|---|---|---|---|---|---|---|---|---|
| | 0.700 | 0.129 | 0.844 | 0.700 | 0.765 | 0.579 | 0.872 | 0.829 | 0 |
| | 0.871 | 0.300 | 0.744 | 0.871 | 0.802 | 0.579 | 0.872 | 0.883 | 1 |
| Weighted Avg. | 0.785 | 0.215 | 0.794 | 0.785 | 0.784 | 0.579 | 0.872 | 0.856 | |

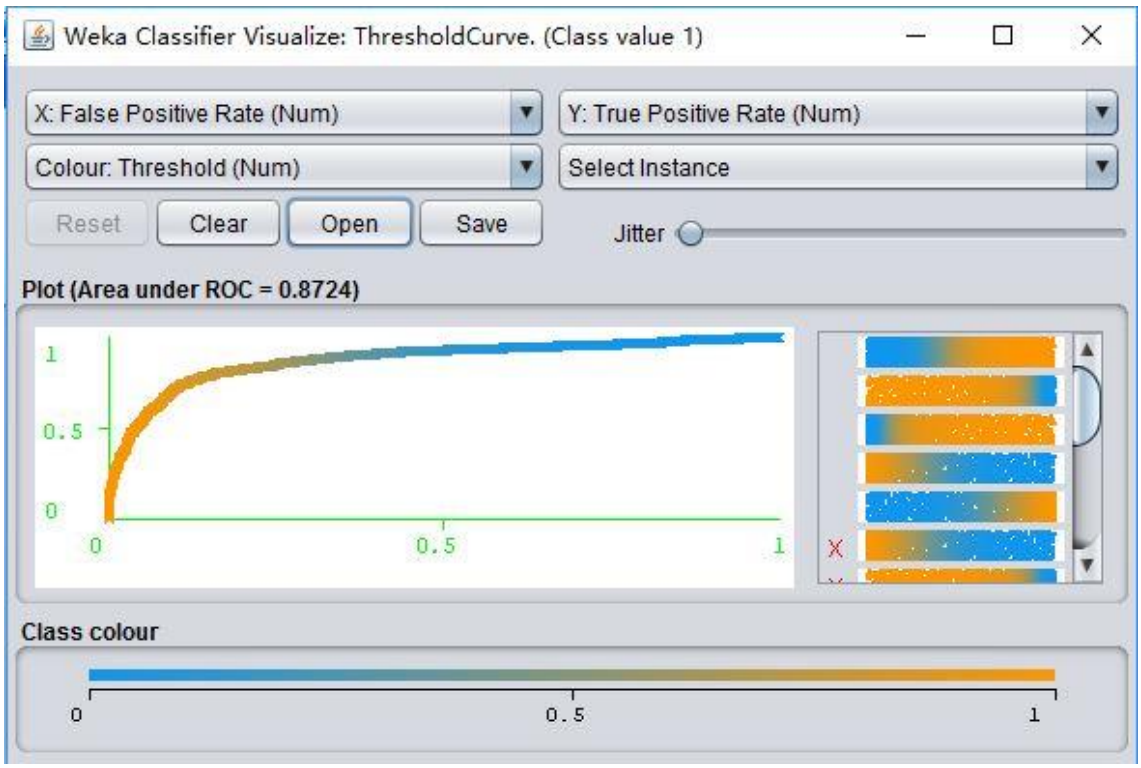
=== Confusion Matrix ===

```
    a     b    <-- classified as
 1015   436 |     a = 0
  187 1264 |     b = 1
```

TPF = TP/(TP+FN) = 1015/(1015+187) = 84.4%.
The overall classification accuracy on training data is 78.532%.

By extracting only nine features, the TPFs on training data and test data are both around 85%, and the overall classification accuracies are also close to 80%. I think the result is acceptable.

The ROC curve is as follows:




According to the information above, the AUC (area under the ROC curve) is 0.8724, which means the classifier has good discrimination capacity.

Because the classifier trained by extracted features has accepted performance, the quality of the features is good.

## Part 3: Data Mining Using Extracted Features

### 3.1 Object Recognition: Classification of Hand-Written Digits

The tabular data set is created as ARFF format to further steps, and the file is named as 'digitdata'. 649 attributes in 6 feature sets and 1 class label are converted to 2000 instance vectors.
I use WEKA to train one J48 decision tree classifier and test it on the test data, the

training data and test data are split from initial data set randomly(random seed of WEKA is 1), each of them have 1000 instance vectors.
The result is as follows:

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     digitdata
Instances:    2000
Attributes:   650
              [list of attributes omitted]
Test mode:    split 50.0% train, remainder test

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

J48 pruned tree
------------------

feature644 <= 0
|   feature648 <= 1.548576
|   |   feature209 <= 436
|   |   |   feature281 <= 401
|   |   |   |   feature260 <= 1030
|   |   |   |   |   feature624 <= 363.440548: 1 (187.0)
|   |   |   |   |   feature624 > 363.440548: 2 (2.0/1.0)
|   |   |   |   feature260 > 1030: 3 (2.0)
|   |   |   feature281 > 401: 4 (3.0/1.0)
|   |   feature209 > 436: 4 (8.0/1.0)
|   feature648 > 1.548576
|   |   feature105 <= 875
|   |   |   feature209 <= 309
|   |   |   |   feature487 <= 3: 2 (2.0)
|   |   |   |   feature487 > 3
|   |   |   |   |   feature498 <= 4
|   |   |   |   |   |   feature119 <= 25: 5 (5.0)
|   |   |   |   |   |   feature119 > 25: 1 (5.0)
|   |   |   |   |   feature498 > 4: 3 (2.0)
|   |   |   feature209 > 309: 4 (188.0/2.0)
|   |   feature105 > 875
|   |   |   feature7 <= 0.476178
|   |   |   |   feature217 <= 18
|   |   |   |   |   feature282 <= 393
|   |   |   |   |   |   feature419 <= 5: 3 (147.0/1.0)
|   |   |   |   |   |   feature419 > 5
```

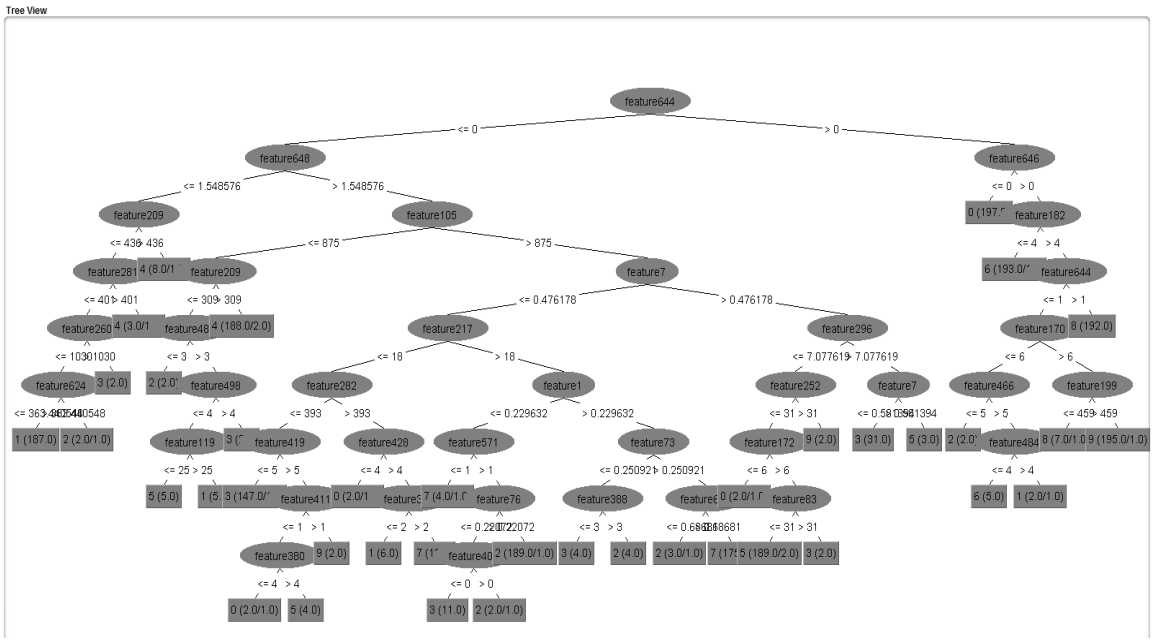
```
|   |   |   |   |   |   |   feature411 <= 1
|   |   |   |   |   |   |   |   feature380 <= 4: 0 (2.0/1.0)
|   |   |   |   |   |   |   |   feature380 > 4: 5 (4.0)
|   |   |   |   |   |   |   feature411 > 1: 9 (2.0)
|   |   |   |   |   feature282 > 393
|   |   |   |   |   |   feature428 <= 4: 0 (2.0/1.0)
|   |   |   |   |   |   feature428 > 4
|   |   |   |   |   |   |   feature375 <= 2: 1 (6.0)
|   |   |   |   |   |   |   feature375 > 2: 7 (19.0)
|   |   |   |   feature217 > 18
|   |   |   |   |   feature1 <= 0.229632
|   |   |   |   |   |   feature571 <= 1: 7 (4.0/1.0)
|   |   |   |   |   |   feature571 > 1
|   |   |   |   |   |   |   feature76 <= 0.22072
|   |   |   |   |   |   |   |   feature402 <= 0: 3 (11.0)
|   |   |   |   |   |   |   |   feature402 > 0: 2 (2.0/1.0)
|   |   |   |   |   |   |   feature76 > 0.22072: 2 (189.0/1.0)
|   |   |   |   |   feature1 > 0.229632
|   |   |   |   |   |   feature73 <= 0.250921
|   |   |   |   |   |   |   feature388 <= 3: 3 (4.0)
|   |   |   |   |   |   |   feature388 > 3: 2 (4.0)
|   |   |   |   |   |   feature73 > 0.250921
|   |   |   |   |   |   |   feature626 <= 0.68681: 2 (3.0/1.0)
|   |   |   |   |   |   |   feature626 > 0.68681: 7 (175.0)
|   |   |   feature7 > 0.476178
|   |   |   |   feature296 <= 7.077619
|   |   |   |   |   feature252 <= 31
|   |   |   |   |   |   feature172 <= 6: 0 (2.0/1.0)
|   |   |   |   |   |   feature172 > 6
|   |   |   |   |   |   |   feature83 <= 31: 5 (189.0/2.0)
|   |   |   |   |   |   |   feature83 > 31: 3 (2.0)
|   |   |   |   |   feature252 > 31: 9 (2.0)
|   |   |   |   feature296 > 7.077619
|   |   |   |   |   feature7 <= 0.581394: 3 (31.0)
|   |   |   |   |   feature7 > 0.581394: 5 (3.0)
feature644 > 0
|   feature646 <= 0: 0 (197.0)
|   feature646 > 0
|   |   feature182 <= 4: 6 (193.0/1.0)
|   |   feature182 > 4
|   |   |   feature644 <= 1
|   |   |   |   feature170 <= 6
|   |   |   |   |   feature466 <= 5: 2 (2.0)
|   |   |   |   |   feature466 > 5
```

```
|   |   |   |   |   |   feature484 <= 4: 6 (5.0)
|   |   |   |   |   |   feature484 > 4: 1 (2.0/1.0)
|   |   |   |   feature170 > 6
|   |   |   |   |   feature199 <= 459: 8 (7.0/1.0)
|   |   |   |   |   feature199 > 459: 9 (195.0/1.0)
|   |   |   feature644 > 1: 8 (192.0)

Number of Leaves  : 39

Size of the tree :  77


Time taken to build model: 1.86 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.08 seconds

=== Summary ===

Correctly Classified Instances         935               93.5      %
Incorrectly Classified Instances        65               6.5      %
Kappa statistic                          0.9277
Mean absolute error                      0.014
Root mean squared error                  0.1118
Relative absolute error                  7.767   %
Root relative squared error             37.2086 %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
                 0.975    0.000    1.000      0.975    0.987      0.986    0.987     0.978     0
                 0.948    0.014    0.875      0.948    0.910      0.901    0.969     0.873     1
                 0.909    0.010    0.899      0.909    0.904      0.895    0.961     0.854     2
                 0.876    0.023    0.832      0.876    0.853      0.835    0.924     0.719     3
                 0.928    0.002    0.975      0.928    0.951      0.947    0.963     0.910     4
                 0.916    0.011    0.897      0.916    0.906      0.896    0.948     0.814     5
                 0.961    0.003    0.970      0.961    0.966      0.962    0.977     0.936     6
                 0.903    0.006    0.949      0.903    0.925      0.917    0.983     0.898     7
                 0.972    0.002    0.981      0.972    0.977      0.974    0.985     0.957     8
                 0.958    0.001    0.989      0.958    0.973      0.971    0.982     0.960     9
Weighted Avg.    0.935    0.007    0.937      0.935    0.936      0.929    0.968     0.890
```

=== Confusion Matrix ===

| a | b | c | d | e | f | g | h | i | j | <-- classified as |
|---|---|---|---|---|---|---|---|---|---|---|
| 115 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | \| a = 0 |
| 0 | 91 | 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | \| b = 1 |
| 0 | 2 | 80 | 2 | 0 | 0 | 0 | 2 | 1 | 1 | \| c = 2 |
| 0 | 3 | 1 | 99 | 1 | 8 | 0 | 1 | 0 | 0 | \| d = 3 |
| 0 | 4 | 0 | 2 | 77 | 0 | 0 | 0 | 0 | 0 | \| e = 4 |
| 0 | 0 | 1 | 7 | 0 | 87 | 0 | 0 | 0 | 0 | \| f = 5 |
| 0 | 1 | 0 | 1 | 1 | 0 | 98 | 1 | 0 | 0 | \| g = 6 |
| 0 | 1 | 5 | 4 | 0 | 0 | 0 | 93 | 0 | 0 | \| h = 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 104 | 0 | \| i = 8 |
| 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 91 | \| j = 9 |

In terms of the information above, the classification accuracy on test data is 93.5%. There are only 38 features in the decision tree, which are required for prediction. The constructed decision tree is as follows:




Then I use WEKA to train another J48 decision tree classifier and test it on the test data by using only the 6 morphological features, the training data and test data are also split from initial data set randomly(random seed of WEKA is 1), both of them have 1000 instance vectors.
The result is as follows:

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2

Relation: digitdata-weka.filters.unsupervised.attribute.Remove-R1-643

Instances: 2000

Attributes: 7

feature644

feature645

feature646

feature647

feature648

feature649

class

Test mode: split 50.0% train, remainder test

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

J48 pruned tree

---

Omit…

Number of Leaves : 69

Size of the tree : 137

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

| | | |
|---|---|---|
| Correctly Classified Instances | 696 | 69.6 % |
| Incorrectly Classified Instances | 304 | 30.4 % |
| Kappa statistic | 0.6626 | |
| Mean absolute error | 0.0664 | |
| Root mean squared error | 0.2027 | |
| Relative absolute error | 36.8514 % | |
| Root relative squared error | 67.4505 % | |
| Total Number of Instances | 1000 | |

=== Detailed Accuracy By Class ===

| TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
|---|---|---|---|---|---|---|---|---|
| 0.975 | 0.000 | 1.000 | 0.975 | 0.987 | 0.986 | 0.987 | 0.978 | 0 |
| 0.927 | 0.015 | 0.864 | 0.927 | 0.894 | 0.884 | 0.977 | 0.877 | 1 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0.625 | 0.045 | 0.573 | 0.625 | 0.598 | 0.558 | 0.868 | 0.487 | 2 |
| | 0.327 | 0.025 | 0.627 | 0.327 | 0.430 | 0.407 | 0.816 | 0.475 | 3 |
| | 0.675 | 0.055 | 0.528 | 0.675 | 0.593 | 0.556 | 0.875 | 0.523 | 4 |
| | 0.632 | 0.049 | 0.577 | 0.632 | 0.603 | 0.560 | 0.909 | 0.515 | 5 |
| | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.929 | 0.492 | 6 |
| | 0.845 | 0.035 | 0.737 | 0.845 | 0.787 | 0.763 | 0.950 | 0.726 | 7 |
| | 0.981 | 0.000 | 1.000 | 0.981 | 0.991 | 0.990 | 0.998 | 0.988 | 8 |
| | 0.968 | 0.113 | 0.474 | 0.968 | 0.637 | 0.635 | 0.935 | 0.465 | 9 |
| Weighted Avg. | 0.696 | 0.032 | 0.649 | 0.696 | 0.658 | 0.640 | 0.926 | 0.663 | |

=== Confusion Matrix ===

| a | b | c | d | e | f | g | h | i | j | <-- classified as |
|---|---|---|---|---|---|---|---|---|---|---|
| 115 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | a = 0 |
| 0 | 89 | 0 | 1 | 1 | 0 | 0 | 5 | 0 | 0 | b = 1 |
| 0 | 1 | 55 | 7 | 4 | 15 | 0 | 4 | 0 | 2 | c = 2 |
| 0 | 1 | 11 | 37 | 31 | 25 | 0 | 8 | 0 | 0 | d = 3 |
| 0 | 5 | 3 | 4 | 56 | 3 | 0 | 12 | 0 | 0 | e = 4 |
| 0 | 1 | 21 | 6 | 5 | 60 | 0 | 2 | 0 | 0 | f = 5 |
| 0 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 98 | g = 6 |
| 0 | 4 | 3 | 3 | 6 | 0 | 0 | 87 | 0 | 0 | h = 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 105 | 2 | i = 8 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 92 | j = 9 |

According to the confusion matrix above, I calculate the accuracies of every class as follows:

| Class | No. of right classification | No. of wrong classification | Accuracy |
|---|---|---|---|
| 0 | 115 | 3 | 97.5% |
| 1 | 89 | 7 | 92.7% |
| 2 | 55 | 35 | 62.5% |
| 3 | 37 | 76 | 32.7% |
| 4 | 56 | 27 | 67.5% |
| 5 | 60 | 35 | 63.2% |
| 6 | 0 | 102 | 0.0% |
| 7 | 87 | 16 | 84.5% |
| 8 | 105 | 2 | 98.1% |
| 9 | 92 | 3 | 96.8% |
| Sum | 696 | 304 | 69.6% |

Therefore the classifier is not good at predicting all classes, only the classification accuracies of 0, 1, 7, 8 and 9 are greater than 80%, which means the 5 hand-written digits are easier to predict, and other 5 classes (2, 3, 4, 5, 6) are difficult to distinguish from each other.

The accuracy of decision tree classifier using all features is 93.5%, which is far above the accuracy of decision tree classifier using only morphological features (69.6%). The result demonstrates that it is not enough to distinguish hand-written digits only according to morphological features. For example, the accuracy of digit 6 from this classifier is 0, and 96.1% (98 / 102) of its instances are classified as 9. I suppose it is because 6 and 9 have the same morphological features but different directions. That is why we require more types of features to distinguish hand-written digits.