Lund University
Faculty of Engineering
Name: ZHANG Yifei
Email: yi4840zh-s@lu.se
Date: October 23, 2023

FMAN20 - Image Analysis, Autumn 2023

Assignment 3

Answer to Task 1 Your Own Classifier

I chose to use the svm model for the classification task.

First, the code defining the model structure and parameter training is performed:

```
function classification_data = class_train(X, Y)
% Training
% X is training sets the size is 361*200 every column values are 19*19 ...
    pictures
% Y are labels the size is 200*1 takes the value for -1 or 1

 % obtain the number of features(image pixel) and number of smaples
[n, m] = size(X);

% initialze variables
alpha = zeros(1, m);
b = 0;
lr = 0.01;  % learning rate
num_iterations = 1000;

% training SVM model
for iteration = 1:num_iterations
    for i = 1:m
        % calculate predict values
        y_pred = b;
        for j = 1:m
            y_pred = y_pred + alpha(j) * Y(j) * dot(X(:, i), X(:, j));
        end

        % update alpha
        if Y(i) * y_pred < 1
            alpha(i) = alpha(i) + lr;

        end
    end
end

% calculate weight vactors
w = zeros(n, 1);
for i = 1:m
    w = w + alpha(i) * Y(i) * X(:, i);
end

% calculate bias
b = mean(Y - w' * X);

% return svm model
svm_model.w = w;
svm_model.b = b;

classification_data.svm_model= svm_model;
end
```

The code for using an SVM model to classify face images is as follows

```
function y = classify(x, classification_data)
    % classification_data contains trained svm model
```

```
3
4       % extract svm model
5       svm_model = classification_data.svm_model;
6
7       % utlizing svm model to classify and perdict
8       scores = svm_model.w' * x + svm_model.b;
9       y = sign(scores);
10  end
```

Select one 19 x 19 image of a face and one 1919 image of a non-face from the test set. Utilize my SVM model to classify and predict the category of the image.

```
1    % Loop through all test examples
2   count_pos = 0;
3   count_neg = 0;
4   for j = 10 : nbr_test_examples
5       predictions_test(j) = classify(X_test(:, j), classification_data);
6       % If prediction is 1 and we haven't displayed two such images yet
7       if predictions_test(j) == 1 && count_pos < 2
8           count_pos = count_pos + 1;
9           % Reshape the image and display it
10          img = reshape(X_test(:, j), [19, 19]);
11          figure;
12          imagesc(img);
13          title(['Face ', num2str(count_pos)]);
14      % If prediction is -1 and we haven't displayed two such images yet
15      elseif predictions_test(j) == -1 && count_neg < 2
16          count_neg = count_neg + 1;
17          % Reshape the image and display it
18          img = reshape(X_test(:, j), [19, 19]);
19          figure;
20          imagesc(img);
21          title(['Non-Face ', num2str(count_neg)]);
22      elseif count_pos ≥ 2 && count_neg ≥ 2
23          break;
24      end
```

Average error rates (100 trials) for the classifier is as follows (first column).

```
mean_err_rate_test =

    0.0527    0.1560    0.0497    0.1605


mean_err_rate_train =

    0.0116    0.0117         0         0
```

Figure 1: error rate

The plot image are as follows.

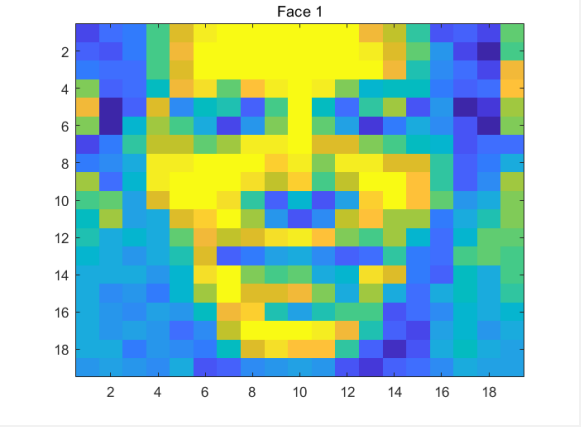


Figure 2: face

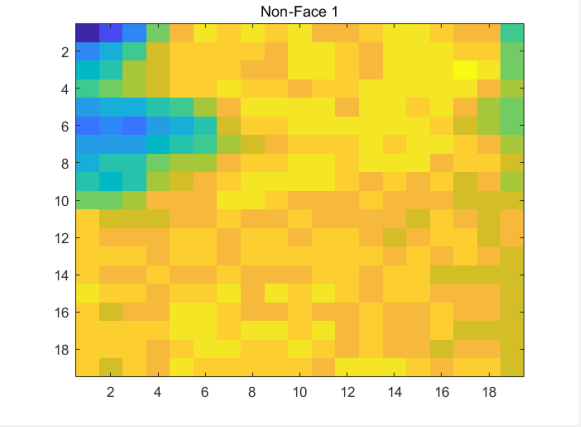


Figure 3: non-face

Answer to Task 2 Use pre-coded machine learning techniques

Average error rates (100 trials) for the classifier is as follows.

```
mean_err_rate_test =

    0.0527    0.1560    0.0497    0.1605


mean_err_rate_train =

    0.0116    0.0117         0         0
```

Figure 4: error rate

The error rates shows that on training set both pre-coded and my own classifier can performed quite well. However, in the test set, possibly due to the absence of hyperparameter tuning, the accuracy of k-nearest neighbors (k-NN) and decision trees is average, whereas SVM demonstrates higher accuracy.

Answer to Task 3 Testing a simple CNN model

Average error rates (100 trials) for the predefined CNN classifier is as follow.

```
mean_err_rate_test =

    0.0825


mean_err_rate_train =

     0

fx >>
```

Figure 5: CNN error rate

Convolutional Neural Networks (CNNs) perform well on both the test and training sets. However, a drawback is that due to their large number of parameters, they demand significant computational resources during training, resulting in longer training times and lower model interpretability.

Answer to Task 4 Line Fit

Code for line fit is as follow.

```
 % Load the data (xm and ym) from linedata.mat
load linedata.mat
data = [xm, ym];
N = length(xm);

% Calculate the mean of data points
mean_x = mean(xm);
mean_y = mean(ym);

% Calculate elements of the covariance matrix
N = length(xm);
Sxx = sum(xm.^2) - (1/N) * (sum(xm))^2;
Syy = sum(ym.^2) - (1/N) * (sum(ym))^2;
Sxy = sum(xm.*ym) - (1/N) * (sum(xm)) * (sum(ym));

% Construct the covariance matrix
A = [Sxx, Sxy; Sxy, Syy];

% Compute eigenvalues and eigenvectors
[V, D] = eig(A);

% Extract eigenvalues and the first eigenvector
eigenvalues = diag(D);
eigenvector = V(:, 1); % Select the first eigenvector

% Normalize the eigenvector to make A^2 + B^2 = 1
eigenvector = eigenvector / sqrt(eigenvector(1)^2 + eigenvector(2)^2);

% Extract coefficients A, B, and C of the line equation Ax + By + C = 0
a = eigenvector(1);
b = eigenvector(2);
c = -(a * mean_x + b * mean_y);

```

```
% Output the line equation
fprintf('Fitted line equation: %.2f*x + %.2f*y + %.2f = 0\n', a, b, c);

% Calculate p and c for the TLS line equation
p_tls = -a/b;
c_tls = -c/b;

% Plot the TLS line
plot(x_fine, p_tls * x_fine + c_tls);

% Display the TLS line equation
fprintf('TLS Line Equation: y = %.4fx + %.4f\n', p_tls, c_tls);

%% Fit a line to these data points using RANSAC and total least squares on the ...
    inlier set.

numIterations = 1000;
inlierThreshold = 0.1; % Adjust as needed
minInliers = 2; % Minimum number of inliers for a valid model

bestModel = struct('slope', 0, 'intercept', 0);
bestInliers = [];
bestInlierCount = 0;

for iteration = 1:numIterations
    % Randomly sample two data points
    sampleIndices = randperm(N, 2);
    xSample = xm(sampleIndices);
    ySample = ym(sampleIndices);

    % Calculate the line parameters (slope and intercept)
    slope = (ySample(2) - ySample(1)) / (xSample(2) - xSample(1));
    intercept = ySample(1) - slope * xSample(1);

    % Calculate distances to the line for all data points
    distances = abs(slope * xm + intercept - ym);

    % Find inliers (data points within the threshold)
    inliers = find(distances < inlierThreshold);
    inlierCount = length(inliers);

    % Update the best model if more inliers are found
    if inlierCount > bestInlierCount
        bestInlierCount = inlierCount;
        bestInliers = inliers;
        bestModel.slope = slope;
        bestModel.intercept = intercept;
    end
end

% Refit the best model using all inliers
xInliers = xm(bestInliers);
yInliers = ym(bestInliers);
bestModel.slope = (mean(xInliers) * mean(yInliers) - mean(xInliers .* yInliers)) / ...
                  (mean(xInliers)^2 - mean(xInliers.^2));
bestModel.intercept = mean(yInliers) - bestModel.slope * mean(xInliers);

% Plot the best-fit line
x_fine = [min(xm) - 0.05, max(xm) + 0.05];
plot(x_fine, bestModel.slope * x_fine + bestModel.intercept, 'k--');

% Display the best-fit line equation
fprintf('RANSAC Line Equation: y = %.4fx + %.4f\n', bestModel.slope, ...
    bestModel.intercept);
```

The visualization of the data points and the two fitted lines.

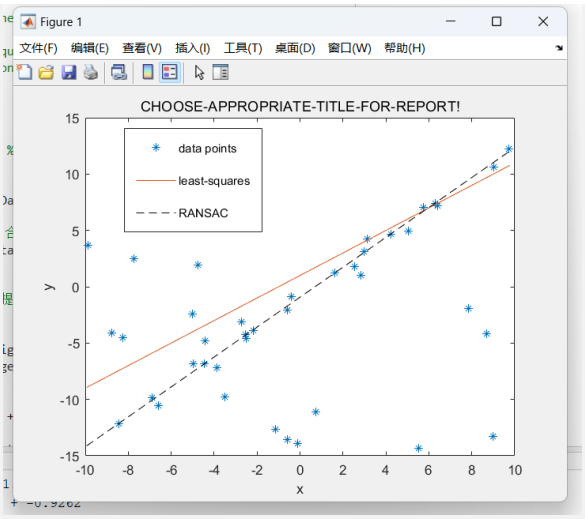


Figure 6: Line Fit

Calculate the line fit error. Code:

```
% Calculate coefficients A and B for the TLS line
A_ls = -p_ls;
B_ls = 1;

% Calculate coefficients A and B for the RANSAC line (inliers only)
A_ransac = -bestModel.slope;
B_ransac = 1;

% Calculate TLS error sum (sum of squared perpendicular distances from points to ...
    the fitted TLS line)
% tls_error_sum_ls = sum(abs(A_ls * xm + B_ls * ym + c_ls) ./ sqrt(A_ls^2 + ...
    B_ls^2)).^2;
tls_error_sum_ls = sum(((p_ls * xm + c_ls - ym).^2) / (1+p_ls^2));

% Calculate RANSAC TLS error sum (inliers only)
% tls_error_sum_ransac = sum(abs(A_ransac * xInliers + B_ransac * yInliers + ...
    bestModel.intercept) ./ sqrt(A_ransac^2 + B_ransac^2)).^2;
tls_error_sum_ransac = sum(((bestModel.slope * xInliers + bestModel.intercept - ...
    yInliers).^2) / (1+bestModel.slope^2));

% Calculate predicted values for the TLS line
y_ls = p_ls * xm + c_ls;

% Calculate predicted values for the RANSAC line (inliers only)
y_ransac = bestModel.slope * xInliers + bestModel.intercept;

% Calculate vertical errors (TLS line)
vertical_errors_ls = ym - y_ls;

% Calculate vertical errors (RANSAC line, inliers only)
vertical_errors_ransac = yInliers - y_ransac;

% Calculate error sums
ls_vertical_error_sum = sum(vertical_errors_ls.^2);

ransac_vertical_error_sum = sum(vertical_errors_ransac.^2);

% Output the four errors
fprintf('LS Vertical Error Sum: %.2f\n', ls_vertical_error_sum);
fprintf('LS Total Error Sum: %.2f\n', tls_error_sum_ls);
fprintf('RANSAC Vertical Error Sum (Inliers Only): %.2f\n', ...
    ransac_vertical_error_sum);
fprintf('RANSAC Total Error Sum (Inliers Only): %.2f\n', tls_error_sum_ransac);
```

Result:

```
TLS Line Equation: y = 1.8100x + -2.3251
RANSAC Line Equation: y = 1.3276x + -0.9262
LS Vertical Error Sum: 3873.83
LS Total Error Sum: 905.90
RANSAC Vertical Error Sum (Inliers Only): 0.02
RANSAC Total Error Sum (Inliers Only): 0.01
```

Figure 7: Line Fit Error

- How many points do you minimally need to sample, in order to estimate the line model in RANSAC?

  To estimate the line model using RANSAC, you minimally need to sample two points. RANSAC randomly selects two data points and fits a line to them during each iteration.

- How is the line model estimated for the minimal dataset in the RANSAC approach?

  For the minimal dataset of two sampled points, RANSAC estimates the line model by fitting a line to these two points using the least squares method. It calculates the slope (p_sample) and y-intercept (q_sample) of the line that best fits these two points.

- What is the difference between the two methods (TLS and RANSAC)?

  The primary difference between the two methods lies in their approach to robust line fitting:

  TLS (Total Least Squares): TLS fits a line to all data points by minimizing the orthogonal distances between data points and the line. It is sensitive to outliers and assumes that all data points have the same weight in the fitting process.

  RANSAC (Random Sample Consensus): RANSAC is a robust method that randomly samples data points and fits a model (in this case, a line) to the sampled points. It then identifies inliers (data points close to the model) based on a threshold and refits the model using only the inliers. RANSAC is more robust to outliers and noisy data because it focuses on the inliers.

Answer to Task 5 OCR system construction and system testing

Hitrates for my OCR-system on short1 and home1:

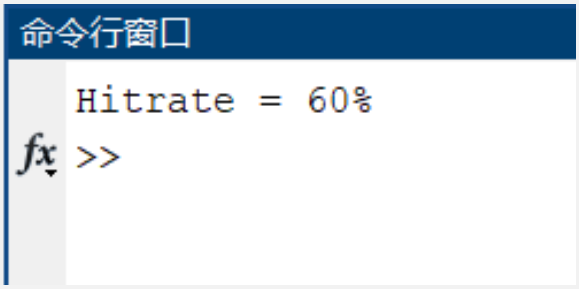


Figure 8: short

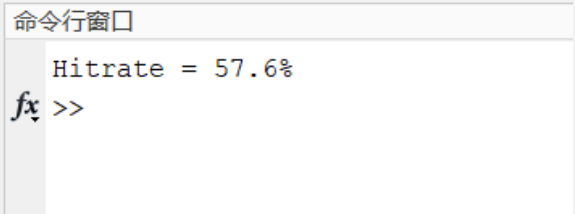


Figure 9: home

The entire OCR system can be divided into three components: segmentation, feature extraction, and classification.

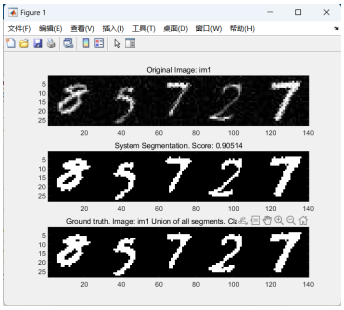
In the segmentation part, I employed an 8-connection algorithm to extract different components, where each component represents a single character or number. Several optimization techniques were also applied, such as filtering and the removal of smaller areas. The advantage of the 8-connection algorithm is its robustness to changes in scale.

Moving on to the feature extraction part, I took into consideration that in the 'home1' test set, the scale may vary. To address this, I utilized scale-independent features. Firstly, Hoff circle detection was employed to identify the number of circles present in each component. Secondly, boundary values within the components were extracted, transformed into bounding boxes, and used as regions of interest (ROIs). These ROIs were then equally divided into 9 parts, and the ratio of white pixels to black pixels in each part was computed.

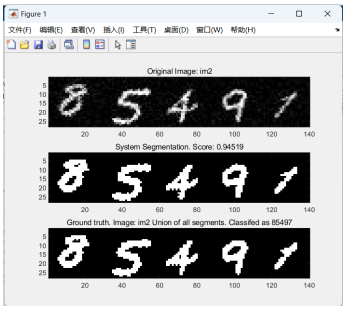
In the classification part, I opted for SVM as the classifier, as it demonstrated the best performance in task 2.

I also experimented with various other features and classification methods, but they did not yield satisfactory results. I plan to conduct further optimization in both feature selection and classification methods. I am considering trying Histogram of Oriented Gradients (HOG) and Backpropagation (BP) neural networks, but I need to address the scaling issue when implementing them.

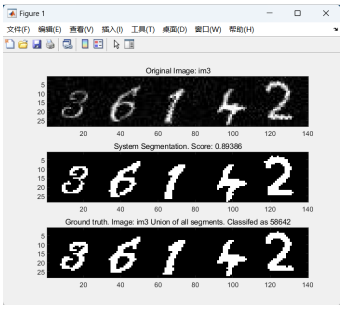
Here I have listed some examples of categorization on short.



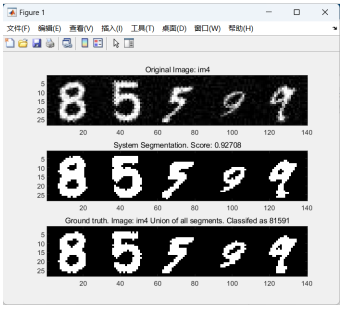
(a) figure 1



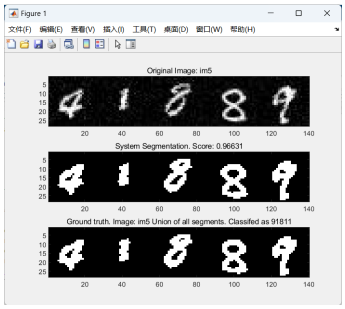
(b) figure 2



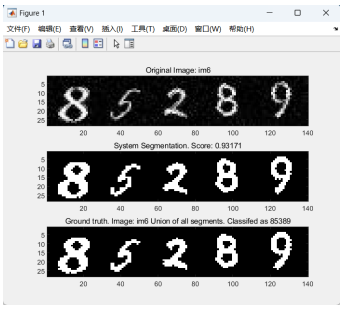
(c) figure 3



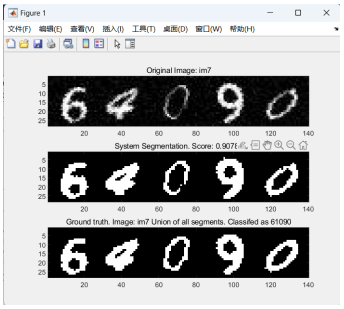
(d) figure 4



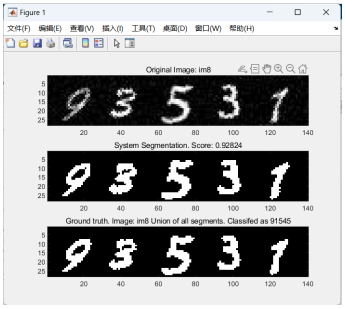
(e) figure 5



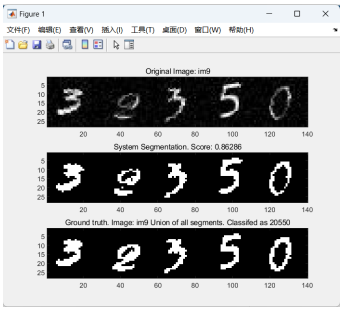
(f) figure 6



(g) figure 7



(h) figure 8



(i) figure 9

Figure 10: classification result