Image Analysis Assignment 3

Ruqayyah Nabage

September 2023

1 My Own Classifier

```matlab
function classification_data = class_train(X, Y)

% I have chosen to implement a k-Nearest Neighbour classifier, so
    in the training function all I do is store the values of X and
    Y in a cell classification_data which is what the function will
     return

classification_data = cell(1,2);
classification_data{1} = X.';
classification_data{2} = Y.';
```

Listing 1: My code for the training function

```matlab
function y = classify(x, classification_data)

% My chosen classifier is a nearest neighbour classifier.

 n = size(classification_data{2},2);
   dist = zeros(1,n);
   for i = 1: n
 % find the distance between x and every 'i' of the classification
    data)
 % find the min, what's the label of that min distance
       dist(i) = norm((x - classification_data{1}(:,i)));
   end
   [~, column] = min(dist);
   y = classification_data{2}(column);
```

Listing 2: My code for the classify function

- My average error rate for the training set was 0 and the average error rate for the test set was 0.1593 . I tried running the classifier several times and the error rates were only slightly different each time. I got 0 error in my classifier on the training set because the model had already learned this data and since my classifier is a nearest neighbour each point just finds itself and there's a zero distance between them so each point is always correctly classified. The error in the test data is to be expected since the model has never seen the data before, but since they're part of the same data set not a lot of error is to be expected which is why 0.1593 is okay as its not too high.

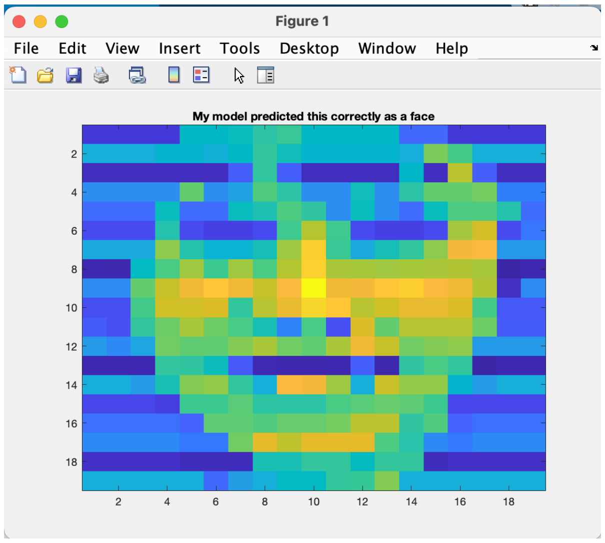
Figure 1
File Edit View Insert Tools Desktop Window Help
My model predicted this correctly as a face
2
4
6
8
10
12
14
16
18

Figure 1: Plot of 'face'; my model predicted this correctly as a face

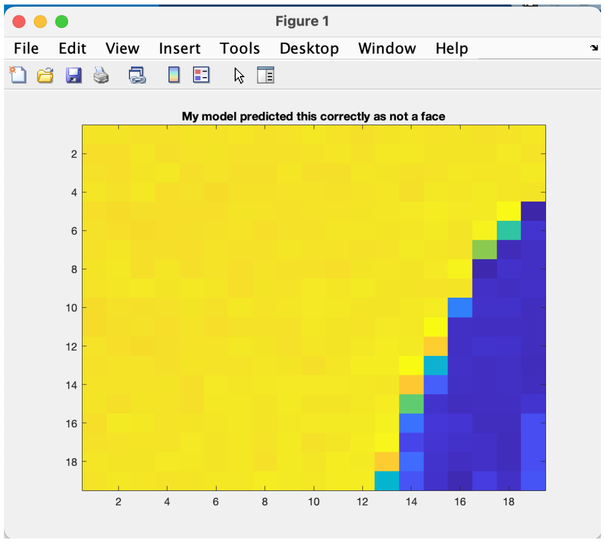
Figure 1
File Edit View Insert Tools Desktop Window Help
My model predicted this correctly as not a face
2
4
6
8
10
12
14
16
18

Figure 2: Plot of 'not face'; my model predicted this correctly as a face

2 Using pre-coded machine learning techniques

I think the errors on the training data for the built-in methods as well as my own classifier are as expected; 0. As this is the data that they were trained on

Table 1: Average Error Rates for 100 Trials

	My Model	**Tree Model**	**SVM Model**	**k-NN Model**
Test Set	0.1487	0.1542	0.0478	0.1487
Training Set	0	0.0428	0	0

so it is expected that the error would be equals to 0.

Regarding the Decision Tree classifier; I believe inherently decision trees perform better when some tuning is performed on the model parameters to avoid overfitting to noise in the data. Also, decision trees are very sensitive to noisy data and this leads to incorrect splits and possibly a higher rate of error. That is why, random forests are mostly favoured over using single decision trees. As random forests takes a lot of decision trees and averages out the errors. But, overall even though it has the highest error for both the training and the test data set; the error is still within a negligible range.

As for my model and the inbuilt k-NN model; they have the same value of errors because they have the same implementation. Also, because of the nature of the k-NN classifer it overfits to the training data completely; that's why there's a 0 error on the training data, and it doesn't always predict new data with that much accuracy; because it finds it a little bit more difficult to generalize on new data. To remedy the problem of overfitting a higher 'k' number of neighbours is chosen.

3 Testing a simple CNN model

The CNN classifer gets a very low error rate on the test data as expected because it is a very robust method for classification and since the test data is a split from the training data; it is highly likely that the data is similar and so the classifier can easily classify the data correctly. It's performance on the training data is of course perfect as this is the data that it has trained on and it knows it perfectly.

In comparison with the performance of the other algorithms, only the Support Vector Machine model performs as competitively as the CNN model. I ran several iterations of the in-built models simultaneously with the CNN model and a few times, the SVM model outperformed the CNN model and sometimes the CNN model outperformed the SVM. This I believe is because they are both highly robust algorithms that are more computational in nature that are not prone to overfitting and therefore better performance on new data.

Table 2: Average Error Rates for 100 Trials with a CNN Classifier

Error on Training Data	0
Error on Test Data	0.0250

4 Line fit

To fit a line using the total least squares approach, I wrote it as a function since I would be reusing it when fitting the inlier points in the RANSAC approach as well. I also designed it so that it would return the total least squares error as well, since I need to provide that as well. Here is the function attached below.

```matlab
function [p_ls,error] = tls(xm,ym)
N = length(xm);
% creating a 2x2 matrix that has the eigen value solution of a and
    b for maximum and minimum
A = zeros(2);
A(1) = sum((xm.^2)) - (1/N)* (sum(xm))^2;
A(2) = sum(xm.*ym) - (1/N) * sum(xm) *sum(ym);
A(3) = A(2);
A(4) = sum((ym.^2)) - (1/N)* (sum(ym))^2;
%a_b has a in the row 1, and b in row 2
[a_b,~] = eig(A);
a = [a_b(1,:)];
b = [a_b(2,:)];

c = zeros(1,2);
c(1) = (-1/N)* (a(1)*sum(xm) + b(1)*sum(ym));
c(2) = (-1/N)* (a(2)*sum(xm) + b(2)*sum(ym));

% errors; to see which is the min and which is the max
error1 = zeros(1,N);
error2 = zeros(1,N);
for i = 1:N
    error1(i) = (a(1)*xm(i) + b(1)*ym(i) + c(1))^2;
    error2(i) = (a(2)*xm(i) + b(2)*ym(i) + c(2))^2;

end
sum_error1 = sum(error1);
sum_error2 = sum(error2);

p_ls = zeros(2,1);

% assigning the set with min error as p_ls; such that the predicted
     model will be ==> y = p_ls(1) * xm + p_ls(2). (From equation
    of a line y = mx + c; in terms of ax + by + c = 0 which is y =
    (-a/b)x - (c/b)
if sum_error1 < sum_error2
    p_ls(1) = -a(1)/b(1);
    p_ls(2) = -c(1)/b(1);
    error = sum_error1;
elseif sum_error1 > sum_error2
    p_ls(1) = -a(2)/b(2);
    p_ls(2) = -c(2)/b(2);
    error = sum_error2;
end
```

Listing 3: Function to fit a line using Total Least Squares

```matlab
% Clear up
clc;
close all;
```

```matlab
clearvars;

% Begin by loading data points from linedata.mat
load linedata
N = length(xm); % number of data points

% Fitting a line to the data points with total least squares assuming the line has the form y = p_ls(1) * x + p_ls(2)).

[p_tls, tls_error] = tls(xm,ym);

% Fitting a line to the data points using RANSAC and total least squares on the inlier set.
distances = zeros(N,1);
threshold = 1.5;
bestModel = []; % Initialize the best model
noBestInliers = 0; % Initialize the number of inliers for the best model
noIterations = 1000;

for i = 1:noIterations % (no. of iterations) randomly choose 2 points by randomly generating an index and assigning the value from the index of xm and ym

    sampleIndices = randperm(size(xm, 1), 2);
    x = xm(sampleIndices, :);
    y = ym(sampleIndices, :);

    % now to fit them to a straight line model

     a = y(2) - y(1)/x(2) - x(1);
     c = y(1) - a * x(1);
     line = [a,c];

    % Find inliers (points that fit that particular model)
    for j = 1:N  % (since we're sampling 2 data points, check error for remaining 38 points in xm and ym)
        distances(j) = abs((a*xm(j)) + (-1*ym(j)) + c)/sqrt(a^2 + (-1)^2);
    end

    inliers_index = find(distances < threshold);

    % If the current model has more inliers than the best model so far, update it

    if numel(inliers_index) > noBestInliers
        bestModel = line;
        noBestInliers = numel(inliers_index);
        bestInliers = inliers_index;
        x_inliers = xm(inliers_index);
        y_inliers = ym(inliers_index);
    end

end

% now fit a line with the best inliers
```

```
[p_ransac, ransac_error] = tls(x_inliers,y_inliers);

 x_outliers = ones(40,1);
 x_outliers(bestInliers) = 0;
 x_outliers = xm(x_outliers == 1);

 y_outliers = ones(40,1);
 y_outliers(bestInliers) = 0;
 y_outliers = ym(y_outliers == 1);

ls_error = lsError(xm,ym,p_tls); %The LS error of the line fitted
    with TLS.
ransac_ls_error = lsError(x_inliers, y_inliers,p_ransac); %The LS
    error of the line fitted with RANSAC using the inlier set only.
```

Listing 4: Code for executing the TLS fitting and the RANSAC fitting

```
function ls_error = lsError(xm,ym, p_ls)
% predicted line
y_predicted = p_ls(1) * xm + p_ls(2);

%vertical errors
vertical_error = ym - y_predicted;
ls_error = sum(vertical_error.^2);
```

Listing 5: Code for function that finds the Least Squares Error

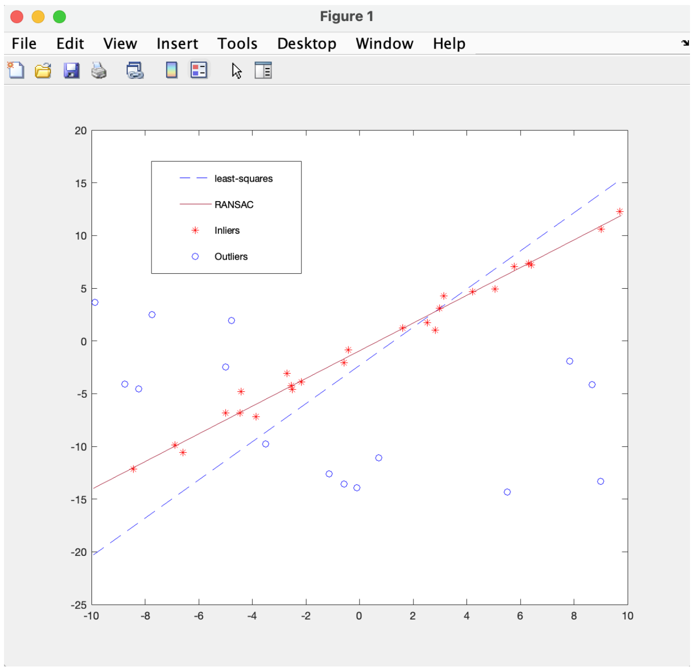

Figure 3: Plots of the data points and the two fitted lines

Table 3: TLS and LS errors of fitted lines

	Total Least Squares Line	RANSAC Line
TLS Error	905.898	5.4691
LS Error	3873.8346	14.9133

- I usually need to sample a minimum of two points in order to estimate the line model in RANSAC, I do this by generating 2 random numbers between 1 to 40 (the length of our dataset xm and ym); then assign the value at that index to be my x and y.
- The line model is estimated by calculating the slope and the intercept of a line between two points using the simple linear algebra formula;

  $$slope = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{and} \quad intercept = y_1 - slope \times x_1$$

  . These values are stored as the line model; line = [slope, intercept].
- The difference between the two methods is that the RANSAC method tries to find the best possible model for a particular set of point; inliers, and the outlier points don't have an effect on the final model of the line. Meanwhile, the Total Least Square method on all the points tries to find a model that minimizes the error on all the points in the dataset and therefore making it much more susceptible to the effect of outliers.
- Here are the errors in the table below:

The error on the RANSAC line is much lower than the error on the TLS line because like I mentioned above; the TLS tries to find the line that minimizes the error for all the points meanwhile; the RANSAC filters outliers from the data points. Also, the TLS Error is lower and is a better measure of the error because it measures the squared orthogonal errors (considers the orthogonal error of the data point to the line; i.e vertical and horizontal aspects of the error) and the LS error is higher because it assumes that the errors are only in the y-direction; which translates to counting only vertical offsets from the line as errors, near vertical lines lead to quite large values of the error.

5 OCR System

Table 4: Performance of OCR System

Performance on short1	64%
Performance on home1	10.1%

I re-used the classifier I designed in task1 which was a nearest neighbour classifier. The next step I took was to improve my segment2features function as it's performance in the previous assignment had only been acceptable and not very good. I chose some new features:

- I designed a for loop; that goes pixel by pixel and finds the longest horizontal line in the image and another that finds the longest vertical line in the image as well, with the idea being that the same numbers will have similar length of longest lines.
- The next new feature I chose was the position of the longest line (vertical and horizontal); with the idea being that the same number would have the the longest lines in similar positions.
- Another feature I chose was the count of how many sections of connected white pixels each row has; for the top part and the bottom part.

I also had previously only normalized my features by dividing each by the maximum value possible for that feature. I changed this to a Min-Max normalization. I also added polynomial combinations of some of my features that I felt were important (by adding their squared value to the feature vector).

However, even with these improved features I was only able to get a hitrate of 40%. I decided to run the benchmark using the mode 2 debug mode. I noticed that my system's reconstructed number segments were a lot 'thicker/fatter' than the ground truth's number segments. I realized I would need to adjust my im2segment function.

My methodology in designing that function had been to filter the image, threshold, perform dilation on the image to fill in any holes in case of faint images that would result in a single number being considered as separate segments. Previously, I had used a disk structural element of radius 2 for the dilation. Once I reduced the radius to 1; my hitrate went up to 64%!.

I think the reason I got a very low hitrate on the home1 dataset is because the images are completely new to my classifier and it being a nearest neighbour classifier; it doesn't have a very robust performance on unseen data or data that is not very similar to it's training data (it overfits on training data).