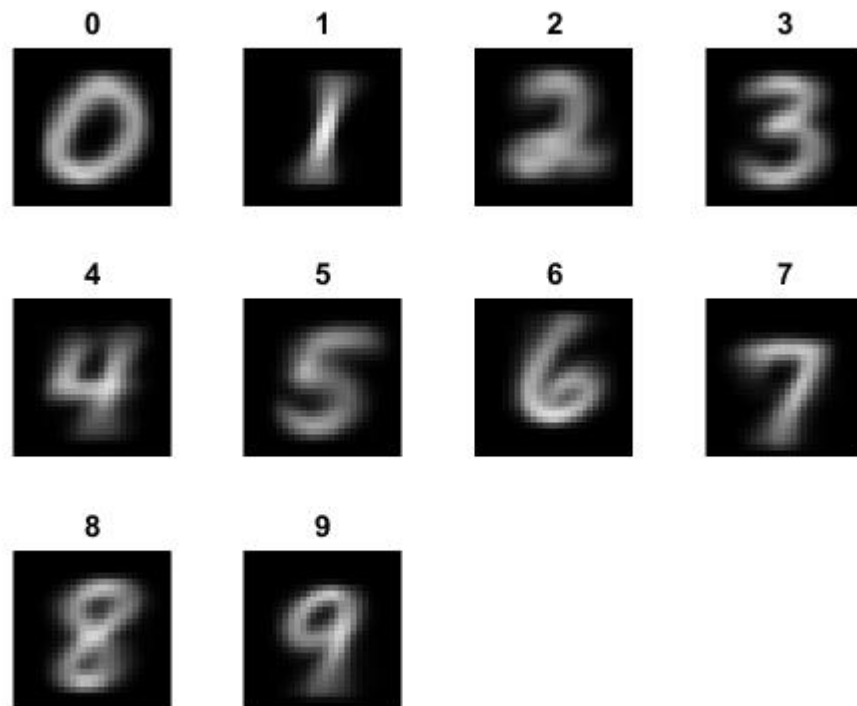


1 Task 1.

1.1 Solution



1.2 Code

8/8

```
close all, clear all, clc
load('Example_MNIST_digits.mat'); % Gives us; 'b' and 'labbb'

for i = 1:numel(unique(labb)) % from 1 to the different number of labels
    subplot(3,4,i) % create a sub plot of 3 by 4 and plot in the ith element
    imshow(uint8(reshape(mean(b(labb == i,:)), 28, 28)')); %finds the index
    % all rows which have label i, extracting only the rows which match and
    % calculate the mean of them column by column then reshaping the image
    % to make it square, transposing the image so it appears the right way
    % up, converting it to an image then plotting it before moving on to
    % the next one
    title(i-1); % puts a title to represent the number it is showing i-1
    % because the first number is zero not one and the last number is nine
    % and not ten
end
```



2 Task 2.

2.1 Code

```
clc, close all
```

```
D = b<120; % converts the image to binary, any number less than 120 gets converted
% to a 1 any number higher gets converted to a 0 meaning you would get an
% image of black (0) and white (1) pixels, the actual number would
% typically be the black pixels as those values would have been greater than
% 120 originally, the background would be white because it was less
r = 1-(mean(D',1)); % adds up all of the 1s and divides it by the total amount
% of pixels in that row gives the proportion of white, so we do minus-1 to
% get the proportion of black pixels
a = zeros(size(r)); % creates a blank vector to store the aspect ratio of the
% same size as r
for i = 1:size(D,1) % from 1 to the number of elements in a row
    c = reshape(D(i,:), 28, 28); % pulls out every row of that number individually
    [row, col] = find(not(c)); % finds the row and column index for every
    % element that is 0, ignoring 1 as that is the background
    a(1, i) = (max(row) - min(row)) / (max(col)-min(col));
    % left most point minus right most point
    % top most point minus bottom most point
    % divided totals together to get aspect ratio
end
Z = [a(:) r(:)]; % creates a dataset with 2 features 'a' aspect ratio
% r proportion of black pixels to white pixels
```

5/5


3 Task 3A.

3.1 Solution

```
errorRate =
0.3326
0.1467
0.3110
0.3637
0.3657
0.3309
0.3486
0.3593
0.3572
0.2941
```

3.2 Code

```
clc, close all
errorRate = zeros(10,1); % initialize matrix to store the error rates out of the loop
for i = 1:10 % each actual digit
    ltp = unique((labb(labb ~= i))); % labels to test against, reassigned everytime
    % we change the training data
    CM = zeros(2,2); % reset confusion matrix for the next digit
    for j = 1:9 % testing against
        ind = labb == i | labb == ltp(j); % index of the test and training data
        ZZ = Z(ind,:); % index used to extract corresponding aspect ratio
        % and proportion from dataset Z
        YY = labb(ind); % index used to extract corresponding labels
        NY = classifier(ZZ,YY); % extracted dataset and labels as input to
        % the function

        % confusion matrix 
        CM(1,1) = CM(1,1) + sum(YY==i & NY == i); % calculates elements which were
        % assigned positive and were positive (true positive)
        CM(1,2) = CM(1,2) + sum(YY==i & NY == ltp(j)); % calculates elements which
        % were actually negative but assigned positive (false positive)
        CM(2,1) = CM(2,1) + sum(YY==ltp(j) & NY == i); % calculates elements which
        % were actually positive but were assigned negative (false negative)
        CM(2,2) = CM(2,2) + sum(YY==ltp(j) & NY == ltp(j)); % calculates elements
        % which were assigned negative and were negative (true negative)
    end


    errorRate(i) = (CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the error rate by
    % adding up the elements which were wrongly assigned, which are false
    % positive and false negative and divided them by the total amount of
    % elements
end
```

4/7

4 Task 3B.

4.1 Code

```
function NewLabels = MyNMC(Ztr,Ytr,Zts)
% Ztr training set
% Ytr training labels
% Zts testing set

for i = 1:numel(unique(Ytr)) % for the amount of possible classifications
    ind = Ytr == unique(Ytr(i)); % index of all training points which match this label
    tstDm(i, :) = mean(Ztr(ind,:)); % calculates the mean x and mean y of...
    % these testing points, returning means for i clas 

    for j = 1:size(Zts,1) % for all of the points to be tested
        d1(j,i) = sqrt((Zts(j,1) - tstDm(i,1))^2 + (Zts(j,2) - tstDm(i,2))^2);
        % ... the euclidean distance: x1 of testing data minus x1 of the
        % mean of class i of the training data, then x2 of the testing data minus
        % x2 of the mean of class i of the training data, looping through every
        % object in the training data then against every mean of the class
        % storing the results in a j*i array, j being the amount of
        % objects, i being the amount of classes

        [M I] = min(d1(j,:)); % finds the index of the minimum value of each column
        % the minimum value is the distance from the object i of
        % the training data to the closest class
        NewLabels(j,1) = unique(Ytr(I)); % creates a new label vector for the labels
        % of the testing data using the index of the column for the value which
        % had the minimum distance
    end
end
end
```

5/16

5 Task 3C.

5.1 Solution

errorRate =

0.4412

0.4295

0.3848

0.4722

0.4051

0.4026

0.4594

0.4247

0.4211

0.3718



5.2 Code

```

clc, close all

errorRate = zeros(10,1); % initialize matrix to store the error rates out of the loop
for i = 1:10 % each actual digit
    ltp = unique((labbb(labb ~= i))); % labels to test against, reassigned
    % everytime we change the training data
    CM = zeros(2,2); % reset confusion matrix for next digit
    training = Z; % a copy of the training data
    digitLabels = labbb; % a copy of the labels

    for j = 1:9 % testing against
        ind = digitLabels == i | digitLabels == ltp(j); % index of the test and
        % training data index used to extract corresponding aspect ratio
        % and proportion from dataset Z
        % index used to extract corresponding labels
        ZZ = training(ind,:); % a copy of the data to be tested using the ind
        % of all those which meet the requirement
        for k = 1:size(ZZ,1)-1 % for every element to be trained
            %removeIndex = find (labbb == i);
            ZZ = training(ind,:); % reset the training data
            YY = digitLabels(ind); % reset the labels
            testing = ZZ((k),:); % take the kth element from the data for testing
            testingL = YY((k)); % take the kth labe from the data for testing
            ZZ((k),:) = []; % delete that element from the training
            YY((k)) = []; % delete that label from the training
            NY = MyNMC(ZZ, YY, testing); % ZZ is the training data, YY is the
            % training labels 'testing' is the 1 element of the initial data which
            % was removed from zz and added here this changes every
            % iteration until it has gone through each possible element

            % confusion matrix
            CM(1,1) = CM(1,1) + sum(testingL==i & NY == i); % calculates elements
            % which were assigned positive and were positive (true positive)
            CM(1,2) = CM(1,2) + sum(testingL==i & NY == ltp(j)); % calculates elements
            % which were actually negative but assigned positive (false positive)
            CM(2,1) = CM(2,1) + sum(testingL==ltp(j) & NY == i); % calculates elements
            % which were actually positive but were assigned negative (false negative)
            CM(2,2) = CM(2,2) + sum(testingL==ltp(j) & NY == ltp(j)); % calcaules
            % elements which were assigned negative and were negative (true negative)
        end
    end

    errorRate(i) = (CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the error rate by
    % adding up the elements which were wrongly assigned, which are false
    % positive and false negative and divided them by the total amount of
    % elements
end
end

```

6 Task 4.

6.1 Solution

Original Data Error Rate =

0.0328

0.0314

0.0462

0.0555

0.0397

0.0577

0.0295

0.0379

0.0486

0.0536

DataSet D Error Rate =

0.0317

0.0312

0.0461

0.0541

0.0390

0.0570

0.0292

0.0379

0.0487

0.0539



The difference in the error rate between the original data set and data set D is very minimal however dataset D has a slightly lower error rate, this is due to the pixels being more distinguishable to each object as any pixel below the threshold was "lost" when the data was converted to binary, meaning there would be less noise in the data.

5/8

6.2 Code: Original Data Set

```

close all, clc

errorRate = zeros(10,1);
for i = 1:10 % each actual digit
    ltp = unique((labbb(labbb ~= i))); % labels to test against, reassigned everytime
    % we change the training data
    CM = zeros(2,2);
    for j = 1:9 % testing against
        ind = labbb == i | labbb == ltp(j); % index of the test and training data
        ZZ = b(ind,:); % index used to extract corresponding aspect ratio
        % and proportion from dataset Z
        YY = labbb(ind); % index used to extract corresponding labels
        NY = classifier(ZZ,YY); % extracted dataset and labels as input to
        % the function

        % confusion matrix
        CM(1,1) = CM(1,1) + sum(YY==i & NY == i); % calculates elements which were
        % assigned positive and were positive (true positive)
        CM(1,2) = CM(1,2) + sum(YY==i & NY == ltp(j)); % calculates elements
        % which were actually negative but assigned positive (false positive)
        CM(2,1) = CM(2,1) + sum(YY==ltp(j) & NY == i); % calculates elements
        % which were actually positive but were assigned negative (false negative)
        CM(2,2) = CM(2,2) + sum(YY==ltp(j) & NY == ltp(j)); % calculates elements
        % which were assigned negative and were negative (true negative)

    end

    errorRate(i) = (CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the error rate by
    % adding up the elements which were wrongly assigned, which are false
    % positive and false negative and divided them by the total amount of
    % elements
end

```


6.3 Code: Dataset D

```
close all, clc
```

```
errorRate = zeros(10,1);
for i = 1:10 % each actual digit
    ltp = unique((labbb(labbb ~= i))); % labels to test against, reassigned
    % everytime we change the training data
    CM = zeros(2,2);
    for j = 1:9 % testing against
        ind = labbb == i | labbb == ltp(j); % index of the test and training data
        ZZ = D(ind,:); % index used to extract corresponding aspect ratio
        % and proportion from dataset Z
        YY = labbb(ind); % index used to extract corresponding labels
        NY = classifier(ZZ,YY); % extracted dataset and labels as input to
        % the function


        % confusion matrix
        CM(1,1) = CM(1,1) + sum(YY==i & NY == i); % calculates elements
        % which were assigned positive and were positive (true positive)
        CM(1,2) = CM(1,2) + sum(YY==i & NY == ltp(j)); % calculates elements
        % which were actually negative but assigned positive (false positive)
        CM(2,1) = CM(2,1) + sum(YY==ltp(j) & NY == i); % calculates elements
        % which were actually positive but were assigned negative (false negative)
        CM(2,2) = CM(2,2) + sum(YY==ltp(j) & NY == ltp(j)); % calculates elements
        % which were assigned negative and were negative (true negative)

    end

    errorRate(i) = (CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the error rate by
    % adding up the elements which were wrongly assigned, which are false
    % positive and false negative and divided them by the total amount of
    % elements
end
```

7 Task 5.

7.1 Code



```


function [D] = MyRoc ( Z, Y )

% initialize empty variables outside of loop
Tp = Z*0; % true positive
Fp = Tp; % false positive
Tn = Tp; % true negative
Fn = Tp; % false negative
startP = [0,0]; % add the starting point
endP = [1,1]; % add the ending point

for i = 1:(numel(Z))-1 % 1 to all but the last element
    mid = ((Z(i)) + (Z(i+1))) / 2; % mid point between the current and next element
    for j = 1:numel(Z) % number of actual elements
        if Z(j) < mid & Y(j) == 1 % if the point is less than the mid and the
            % class is 1
            Tp(i) = Tp(i) + 1; % increment true positive
        elseif Z(j) < mid & Y(j) ~= 1 % if the point is less than the mid and the
            % class is 2
            Fp(i) = Fp(i) + 1; % increment false positive
        elseif Z(j) > mid & Y(j) == 2 % if the point is greater than mind and the
            % class is 2
            Tn(i) = Tn(i) + 1; % increment true negative
        else % if not the point is greater than mind and the
            % class is 1
            Fn(i) = Fn(i) + 1; % increment false negative
        end
    end
end
% stores sensitivity and specificity in a 2D array these are the
% coordinates for the roc curve
A(i, 1) = sum(Tp)/((sum(Tp)+sum(Fn))); % calculate the sensitivity
A(i, 2) = 1-(sum(Tn)/((sum(Tn)+sum(Fp)))); % 1 minus the sum of specificity
D = [startP; A; endP]; % put start and end points as well as the data points

end
end

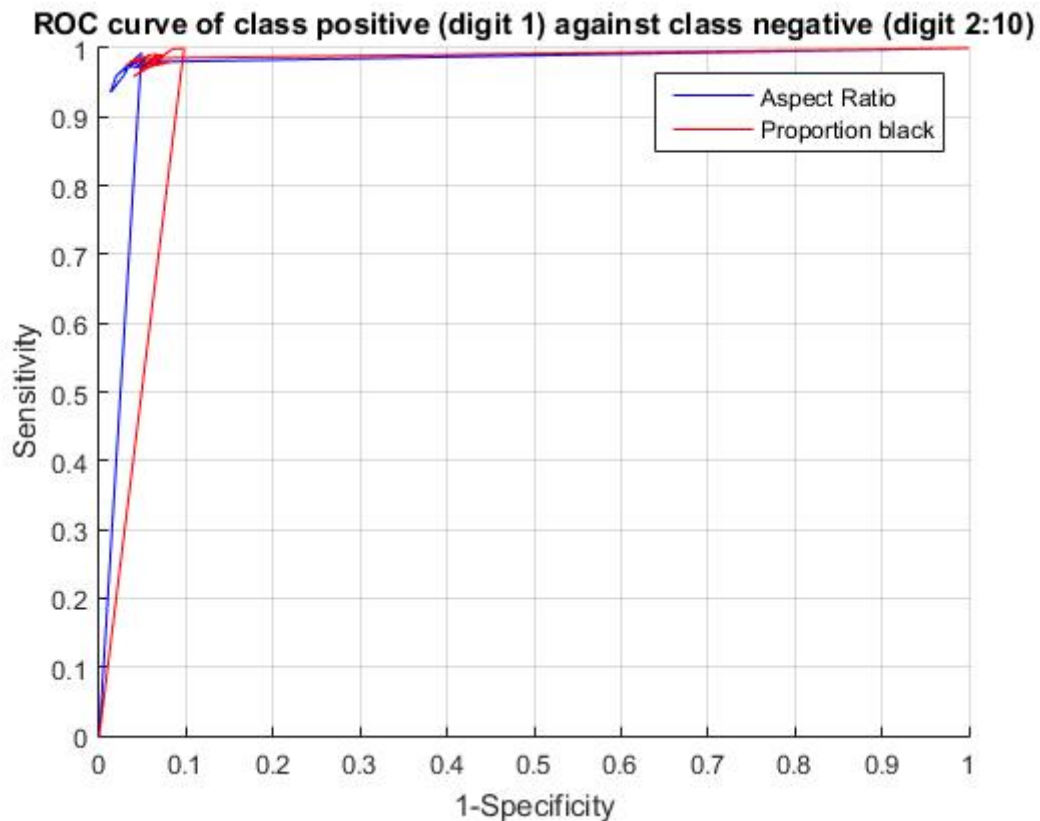
```




9/14

8 Task 6.

8.1 Solution



8.2 Code

```
close all, clc
```



```
[coords] = MyRoc(Z(:,1), labb); % aspect ratio with corresponding labels
% into roc function
[coords2] = MyRoc(Z(:,2), labb); % proportion of dots with corresponding labels
% into roc function
figure, hold on, grid on;

title('ROC curve of class positive (digit 1) against class negative (digit 2:10)')
xlabel('1-Specificity') % x-axis label
ylabel('Sensitivity') % y-axis label
plot(coords(:,1), coords(:,2), 'b'); % plotting aspect ratio
plot(coords2(:,1), coords2(:,2), 'r'); % plotting proportion
legend('Aspect Ratio', 'Proportion black')
```



7/12

9 Task 7.

9.1 Solution

```
CM2ensemble =  
196 213  
209 192
```

```
accuracyEnsemble =  
0.3580  
0.5308  
0.4567  
0.3827  
0.5185  
0.5926  
0.5432  
0.4938  
0.4320  
0.5308
```

```
1NMCAccuracy =  
0.4962  
0.4693  
0.5209  
0.4983  
0.5019  
0.4982  
0.5087  
0.4954  
0.4992  
0.4994
```

The differences in the accuracy between bagging and 1 NMC is on average more accurate for bagging but not by much.

9.2 Code: Bagging

```

clc
training = b; % using the original data
digitLabels = labb; % labels
M = zeros(2,2);
CM2 = zeros(2,2); % reset confusion matrix
errorRate = zeros(10,1); % initialize array outside loop
for i = 1:10 % each actual digit
    ltp = unique((labb(labb ~= i))); % labels to test against, reassigned everytime
    % we change the training data
    CM = zeros(2,2); % reset confusion matrix

    for j = 1:9 % testing against
        ind = digitLabels == i | digitLabels == ltp(j); % index of the test and
                                                    % training data

        ZZ = training(ind,:); % index used to extract corresponding aspect ratio
        for k = 1:9 % from 1 to 9 samples to classify with
            rndIDX = randperm(size(ZZ,1)); % randomizes the indexes in the dataset

            newSample = ZZ(rndIDX(k), :); % takes a the kth sample out of the randomly
            % assorted data set (takes a random sample)

            YY = digitLabels(ind); % index used to extract corresponding labels
            nL = YY(rndIDX(k)); % takes the kth label to match the k sample taken
            NY = MyNMC(ZZ, YY, newSample); % extracted dataset and labels as input to
            % the function to test against sample k

            % confusion matrix which resets per digit
            CM(1,1) = CM(1,1) + sum(nL==i & NY == i); % calculates elements
            % which were assigned positive and were positive (true positive)
            CM(1,2) = CM(1,2) + sum(nL==i & NY == ltp(j)); % calculates elements
            % which were actually negative but assigned positive (false positive)
            CM(2,1) = CM(2,1) + sum(nL==ltp(j) & NY == i); % calculates elements
            % which were actually positive but were assigned negative (false negative)
            CM(2,2) = CM(2,2) + sum(nL==ltp(j) & NY == ltp(j)); % calculates elements
            % which were assigned negative and were negative (true negative)

            % confusion matrix to save
            CM2(1,1) = CM2(1,1) + sum(nL==i & NY == i); % calculates elements
            % which were assigned positive and were positive (true positive)
            CM2(1,2) = CM2(1,2) + sum(nL==i & NY == ltp(j)); % calculates elements
            % which were actually negative but assigned positive (false positive)
            CM2(2,1) = CM2(2,1) + sum(nL==ltp(j) & NY == i); % calculates elements
            % which were actually positive but were assigned negative (false negative)
            CM2(2,2) = CM2(2,2) + sum(nL==ltp(j) & NY == ltp(j)); % calculates elements
            % which were assigned negative and were negative (true negative)
        end
    end

    accuracy(i) = 1-(CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the accuracy by
    % 1 minus the adding up of the elements which were wrongly assigned,
    % which are false positive and false negative and divided them by
    % the total amount of elements
end

```

end

9.3 Code: 1NMC

```

clc
training = b;
digitLabels = labb;

errorRate = zeros(10,1);
for i = 1:10 % each actual digit
    ltp = unique((labb(labb ~= i))); % labels to test against, reassigned everytime
    % we change the training data
    CM = zeros(2,2);

    %testing = Z(newSample);

    for j = 1:9 % testing against
        ind = digitLabels == i | digitLabels == ltp(j); % index of the test and
                                                    % training data

        ZZ = training(ind,:); % index used to extract corresponding aspect ratio

        % and proportion from dataset Z
        YY = digitLabels(ind); % index used to extract corresponding labels

        NY = MyNMC(ZZ, YY, ZZ); % extracted dataset and labels as input to
        % the function

        % confusion matrix
        CM(1,1) = CM(1,1) + sum(YY==i & NY == i); % calculates elements
        % which were assigned positive and were positive (true positive)
        CM(1,2) = CM(1,2) + sum(YY==i & NY == ltp(j)); % calculates elements
        % which were actually negative but assigned positive (false positive)
        CM(2,1) = CM(2,1) + sum(YY==ltp(j) & NY == i); % calculates elements
        % which were actually positive but were assigned negative (false negative)
        CM(2,2) = CM(2,2) + sum(YY==ltp(j) & NY == ltp(j)); % calculates elements
        % which were assigned negative and were negative (true negative)

    end

    accuracy1(i) = (CM(1,2) + CM(2,1)) / sum(CM(:)); % gets the accuracy rate by
    % 1 minus the adding up of the elements which were wrongly assigned,
    % which are false positive and false negative and divided them by the
    % total amount of elements
end

```



8/20

4 points bonus for LaTeX + correct colours of the MATLAB code + making your text readable by observing the "no-broken-lines" rule