Assignment 3

Maks Epsteins

October 10, 2021

Exercise 1

1.1 Code for the Classify Function (Gaussian Bayes)

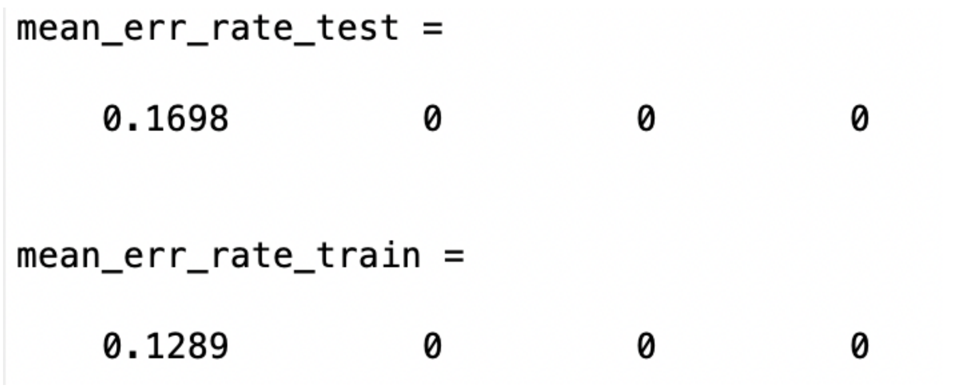
```
function y = classify(x, classification_data)

    nrfeatures = size(x,1);
    nrsamples = size(x,2);
    y = zeros(1,nrsamples);

    for s = 1:nrsamples
        y1_total = 0;
        y2_total = 0;

        for f = 1:nrfeatures

            mean1f = classification_data.class1(1,f);
            stdv1f = classification_data.class1(2,f);

            mean2f = classification_data.class2(1,f);
            stdv2f = classification_data.class2(2,f);

            y1 = normpdf(x(f,s), mean1f, stdv1f);
            y2 = normpdf(x(f,s), mean2f, stdv2f);

            y1_total = y1_total + log(y1);
            y2_total = y2_total + log(y2);

        end

        score1 = y1_total + log(classification_data.class1(:,nrfeatures + 1));
        score2 = y2_total + log(classification_data.class2(:,nrfeatures + 1));

        if score1 >= score2
            y(s) = -1;
        elseif score2 > score1
```

```
            y(s) = 1;
        end

    end

end
```

1.2 Code for the Class Train Function

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

    nrfeatures = size(X,1);
    nrsamples = size(Y,2);

    data = struct('class1', zeros(2,nrfeatures),'class2', zeros(2,nrfeatures));

    class1Y = [find(Y(1,:)==-1)];
    class1X = X(:,class1Y);

    class2Y = [find(Y(1,:)==1)];
    class2X = X(:,class2Y);

    for f = 1:nrfeatures
        param1 = class1X(f,:);
        param2 = class2X(f,:);
        data.class1(:,f) = transpose([mean(param1), std(param1)]);
        data.class2(:,f) = transpose([mean(param2), std(param2)]);
    end


    apriori1 = size(class1X,2)/nrsamples;
    apriori2 = size(class2X,2)/nrsamples;

    data.class1(:,nrfeatures+1) = apriori1;
    data.class2(:,nrfeatures+1) = apriori2;

    classification_data = data;

end
```

1.3 Mean Error Rates for the Data



```
mean_err_rate_test =

    0.1698         0         0         0


mean_err_rate_train =

    0.1289         0         0         0
```

Figure 1: Error rates for task 1

The error rate seems pretty adequate for both the test and the training data.

1.4 Image of face and nonface

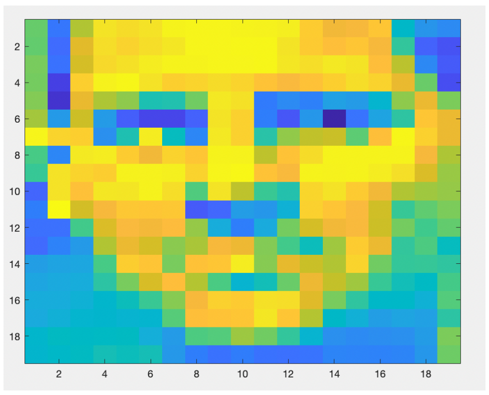


Figure 2: Example image of a face which was classified correctly.

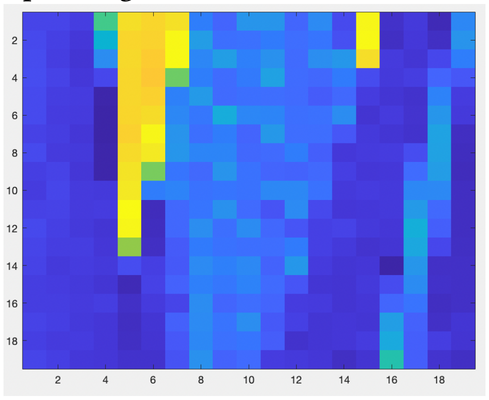
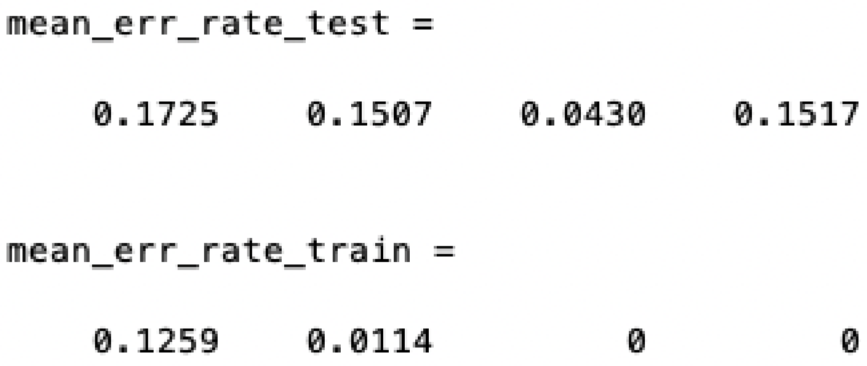


Figure 3: Example image of a non-face which was also classified correctly

Exercise 2

2.1 Error Rates

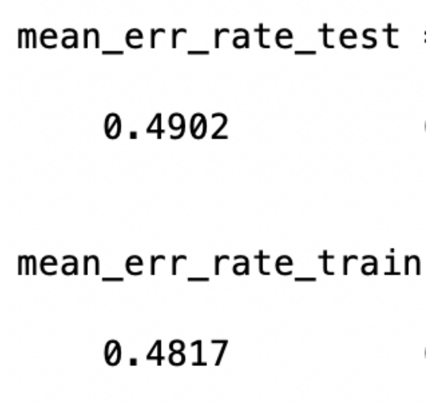


```
mean_err_rate_test =

    0.1725    0.1507    0.0430    0.1517


mean_err_rate_train =

    0.1259    0.0114         0         0
```

Figure 4: Error rates for task 2

All the results appear to be reasonable for each classifier algorithm and they all seem to work quite well. As expected the SVM classifier worked the best, however im surprised that the NN classifier worked better than our Gaussian Bayes. Also the low error rate for the training data also surprises me.

Exercise 3



```
mean_err_rate_test

    0.4902

mean_err_rate_train

    0.4817
```

Figure 5: Error rates

The high error rate for the CNN surprises me however that could be due to the low amount of layers in the network.

Exercise 4

4.1 Least Squares

4.1.1 Code for Least Squares

```
function leastsquares = ls(x,y)

    npoints = max(size(x));
    xsquared = x;

    for i = 1:npoints
        xsquared(i) = x(i)^2;
    end

    xy = x;

    for i = 1:size(x)
        xy(i) = x(i)*y(i);
    end

    sumx = sum(x);
    sumy = sum(y);
    sumxsquared = sum(xsquared);
    sumxy = sum(xy);

    k = (npoints*sumxy - sumx*sumy)/(npoints*sumxsquared - sumx^2);
    m = (sumy - k*sumx)/npoints;

    leastsquares = [k,m];

end
```

4.1.2 Code for Least Squares Error

```
function error = lserror(k,m,x,y)

    npoints = max(size(x));
    error = 0;

    for i = 1:npoints
        error = error + (y(i) - (k*x(i) + m))^2;
    end
end
```

4.1.3 Errors for LS

LS error = 19.5072, TLS error = 20.4984

The errors seem appropriate for the LS regression and it makes sense that the TLS error is larger than the LS error.

4.2 Total Least Squares

4.2.1 Code for Total Least Squares

```
function totalleastsquares = tls(x,y)
%TLS Summary of this function goes here
%   Detailed explanation goes here

    npoints = max(size(x));

    xbar = mean(x);
    ybar = mean(y);

    w = 0;

    for i = 1:npoints
        w = w + (y(i)-ybar)^2 - (x(i) - xbar)^2;
    end

    r = 0;

    for i = 1:npoints
        r = r + 2*(x(i) - xbar)*(y(i) - ybar);
    end

    k = (w + sqrt(w^2 + r^2))/r;

    m = ybar - k*xbar;


    totalleastsquares = [k,m];
end
```

4.2.2 Code for Total Least Square Error

```
function error = tlserror(k,m,x,y)

    npoints = max(size(x));
    error = 0;

    for i = 1:npoints
        x1 = x(i);
        y1 = y(i);

        c = y(i) + k*x(i);

        x2 = (-k*x(i) + c -m)/k;
        y2 = -k*x2 + c;

        p = [x1,y1; x2,y2];
        dist = pdist(p,'euclidean');

        error = error + dist^2;

    end

end
```

4.2.3 Errors for TLS

LS error = 24.1030, TLS error = 24.8930

These results seem weird to me since the TLS error should be smaller than the LS error so either our TLS error function is incorrect or the TLS itself is. However the plot itself (see below) looks correct.

4.3 TLS and LS Regressions Plots

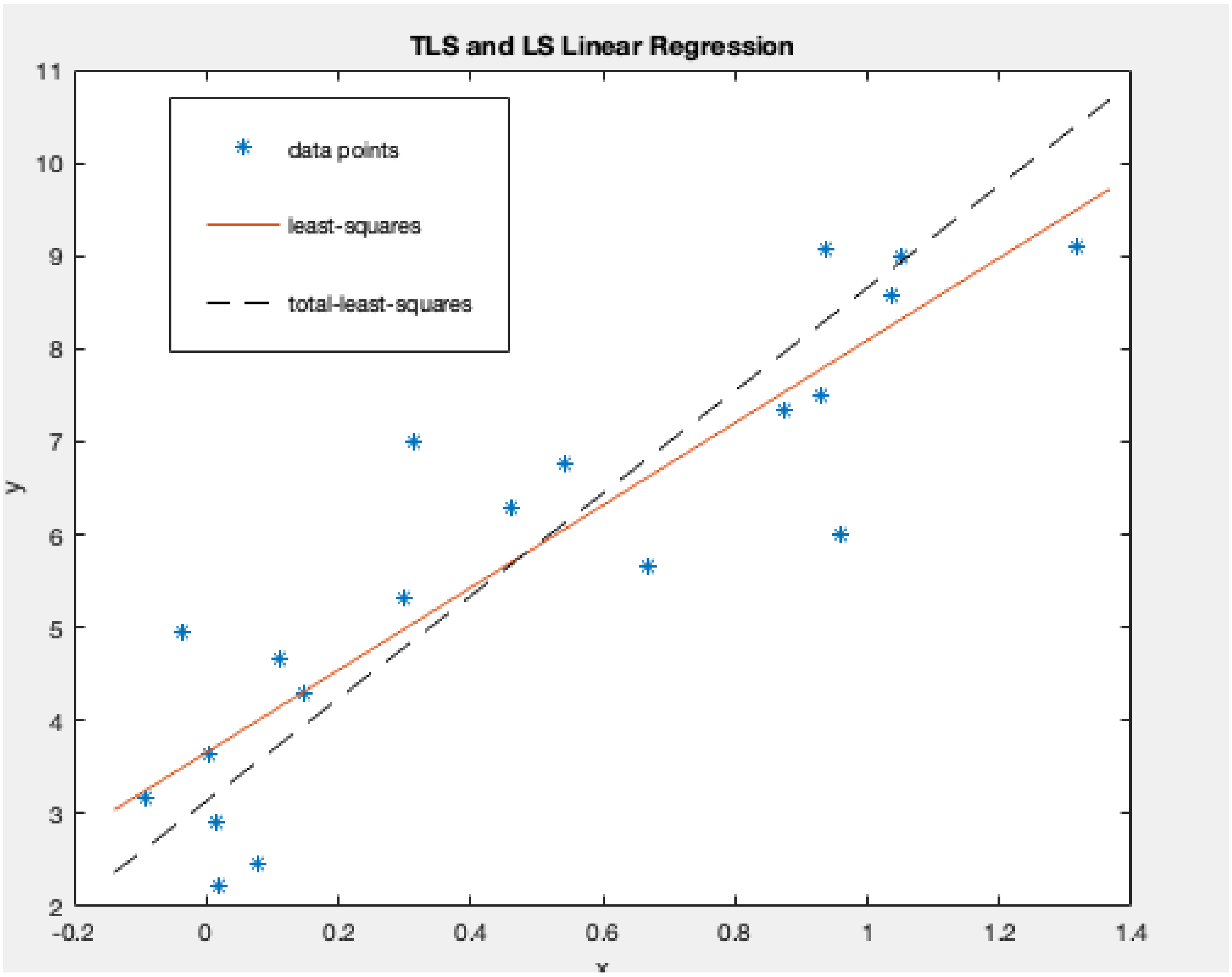


Figure 6: Plot for the TLS and LS Regression Lines

4.4 Difference Between the Two Methods

Both of the methods are used to calculate the approximate solutions for overdetermined systems, in our case to approximate a linear regression. The difference between the two methods is that LS attempts to minimize the vertical distance between the line and each datapoint, while TLS attempts to minimize the euclidean distance.

Exercise 5

5.1 Code for class_train Function

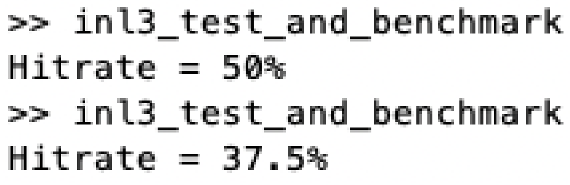
```
%The code was modified from the first exercise to accommodate for any number of
%classes

function classification_data = class_train(X, Y)
nrfeatures = size(X,1);
nrsamples = size(Y,2);
nrclasses = 10;

data = struct();

%Iterate through all the classes
for c=1:nrclasses
    %Find all the data in our dataset that has class c
    classY = [find(Y(1,:) == c)];
    index = strcat('c', num2str(c));

    %Initialize data struct with empty array with space for mean and deviation
    %for each feature for each class.
    data.(index) = zeros(2, nrfeatures);

    %Iterate through each feature.
    for f = 1:nrfeatures
        %Get each data point for the current feature
        classX = X(:,classY);
        parameters = classX(f,:);
        %Add the mean and deviation to our data array
        data.(index)(:,f) = transpose([mean(parameters), std(parameters)]);
    end

    %Calculate our apriori and add it to our data.
    apriori = size(classY,2)/nrsamples;
    data.(index)(:,nrfeatures+1) = apriori;
end

classification_data = data;
end
```

5.2 Code for features2class Function using Gaussian Bayes

```
function y = features2class(x,classification_data)

nrfeatures = size(x,1);
nrsamples = size(x,2);
y = zeros(1, nrsamples);
nrclasses = 10;
```

```
%Iterate through each sample in the data set
for s = 1:nrsamples

    %Initialize the scores for each class to 0
    scores = zeros(10,1);

    %Iterate through each class (1-10)
    for c=1:nrclasses
        prob_total = 0;
        %The data is stored in a struct where the keys written as 'cn'
        %where n is the number to which the class corresponds
        index = strcat('c', num2str(c));

        %Iterate through each features
        for f = 1:nrfeatures
            %Get the calculated mean and standard deviation for each class
            mean = classification_data.(index)(1,f);
            stdv = classification_data.(index)(2,f);

            %If the standard deviation is 0 we only need to provide the mean to
            %normpdf
            if stdv ~= 0
                prob = normpdf(x(f,s), mean, stdv);
            elseif stdv == 0
                prob = normpdf(x(f,s), mean);
            end
            %Zero check to prevent crash if the probability is 0 since we
            %are using the log function.
            if prob ~= 0
               prob_total = prob_total + log(prob);
            end

        end

        %We assign the calculated score into our score array and add the apriori
        %probability
        scores(c,1) = prob_total + log(classification_data.(index)(1,nrfeatures+1));
    end

    %The index variable will correspond to the class with the highest probability
    [~, index] = max(scores);
    y(1,s) = index;
end
end
```

5.3 Classification Results



```
>> inl3_test_and_benchmark
Hitrate = 50%
>> inl3_test_and_benchmark
Hitrate = 37.5%
```

Figure 7: Hitrates for short1 and home1 datasets

5.4 Made modifications

The majority of the modifications that were made were in the segment2features function which I pretty much redid most of. Also had to change the threshold in the im2segment function which made the segmentation worse, but kept more of the features (the holes in the numbers were particularly sensitive).

The features that were added were the following:

```
%Density of white pixels in the middle 5 columns.
midcolumndensity = number(:,x_mid-2:x_mid+2);
midcolumndensity = sum(sum(midcolumndensity))/numel(midcolumndensity);

%Density of white pixels in the middle 5 rows.
midrowdensity = number(y_mid-2:y_mid+2,:);
midrowdensity = sum(sum(midrowdensity))/numel(midrowdensity);

%Feature extraction kernel in order to find horizontal features in the image.
%Will further try to improve this by adding some sort of thresholding.
hozkernel = 1/15*[0,0,0,0,0;
                  1,1,1,1,1;
                  1,1,1,1,1;
                  1,1,1,1,1;
                  0,0,0,0,0];

hozfeatures = conv2(number,hozkernel,'same');
hozfeatures = sum(sum(hozfeatures))/sum(sum(number));

%Feature extraction kernel in order to find vertical features in the image.
%Will also attempt to add some sort of thresholding to this one.
vertkernel = 1/15*[0,1,1,1,0;
                   0,1,1,1,0;
                   0,1,1,1,0;
                   0,1,1,1,0;
                   0,1,1,1,0];

vertfeatures = conv2(number,vertkernel,'same');
vertfeatures = sum(sum(vertfeatures))/sum(sum(number));
```