425 Assignment #1

Noah Marra - 20094614

Training Gaussian Classifier

Initializations

```
D = 64; %number of features
ak = 1/10; %all classes have the same # of features
mk = 700; % # of images for each training classes
K = 10;%total # of classes
M = mk*K; %# of all training data points over K classes
uk = [];
```

Here I am initializing the variables that I'm going to need for a Gaussian classifier training. There purposes are stated in comments in the code

Calculating the Means

```
for i = 1:1:K
        uk(:,i) = mean(digits_train(:,:,i),2);
end
```

Here I am going to use a loop to calculate the means (μ_{ki}) of each individual pixel of each class 'i' from 1 to 10. This will leave me with an array for uk that is 64 x 10. Where 64 is the mean of each pixel in a class and 10 is the number of classes for the dataset.

Calculating the Variance

```
for k = 1:1:K
    for j = 1:1:mk
        s(:,j,k) = (digits_train(:,j,k)-uk(:,k)).^2;
    end
end

variance = sum(s,"all")/(D*M)
varianceRoot = sqrt(variance);
```

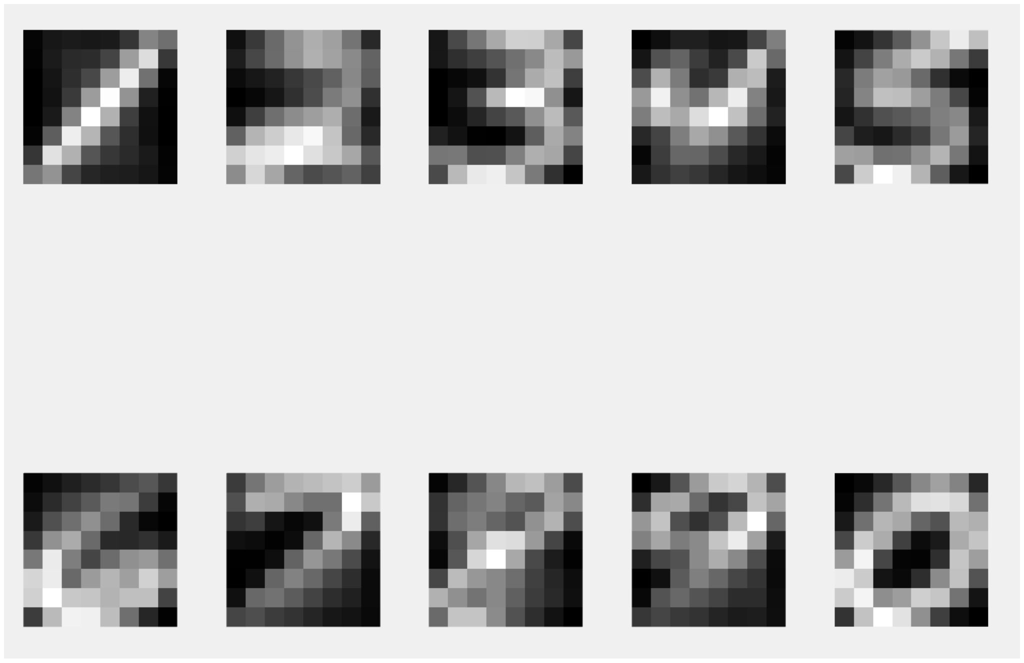
In this section I will need to calculate the variance using formula (4) provided to us on the assignment sheet. I used 2 loops to rotate through each of the images(mk) and each of the classes K. Since I know that I want each of the pixels to be subtracted by their respective means I can use the ':' to state that we want all these data points in the array. This just saves me an extra for loop through the 64 pixels. I then calculated the deviation from the mean for each of the pixels in each image for each class. Then I summed all the contents of that array to get the total sum and divide it by both D and M. Where M is the total number of training images which is 700 images for each class multiplied by the 10 classes hence it is 7000. This will then give me my variance (σ^2), which was 0.0634, and is shown below.

```
variance =


    0.0634
```

Plotting the Means

```
for i = 1:1:K
    subplot(2,5,i);
    imagesc(reshape(uk(:,i),8,8)'); axis equal; axis off; colormap gray;
end
```

Now that I have calculated the means I can use the simple line of code provided to us in the assignment to plot all the images I created. I used the subplot command to break the images up into 2 rows for easier viewing.



As seen above this seems to have worked quite well and the Gaussian classifier has now been trained.

Training Naïve Bayes

Initialization

```
D = 64; %number of features
mk = 700; % # of images for each training classes
K = 10;%total # of classes
ak = 1/10; %all classes have the same # of features
P = zeros([64,10]);

X = (digits_train>0.5);%changes digits over 0.5 to 1 and under 0.5 to 0
```

The initializations are the generally the same as the Gaussian classifier; however, it also includes the addition of the transformation of the digits_train matrix to binary numbers.

Calculating $p(b_i = 1|C_k) = \eta_{ki}$

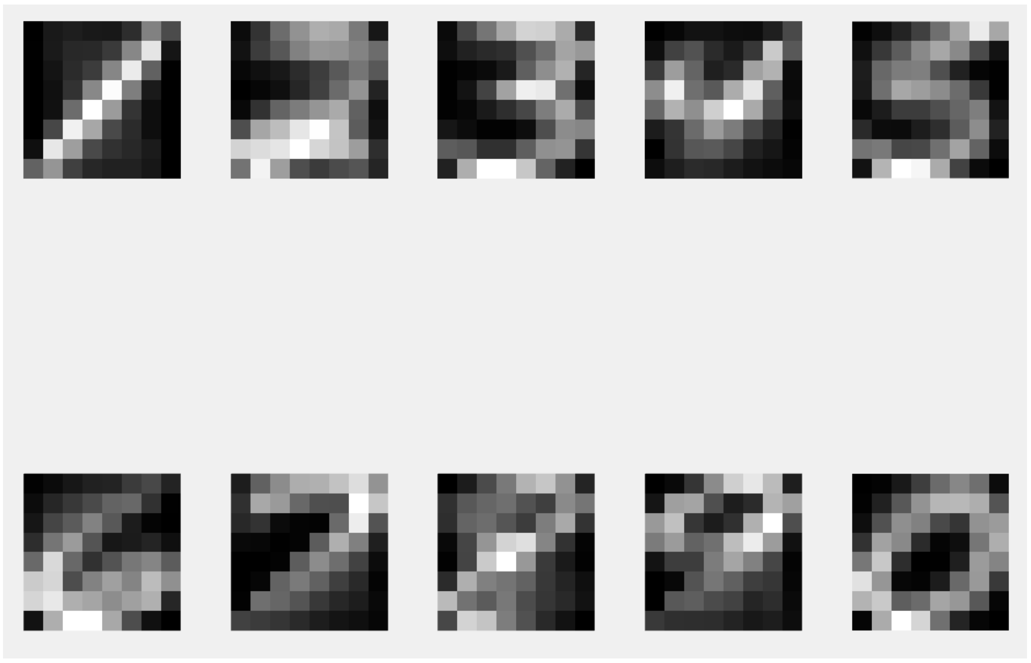
```
for k = 1:1:K
      for j = 1:1:mk
        for i = 1:1:D
           if X(i,j,k)> 0.5
               P(i,k) = P(i,k) + 1;
           end
        end
     end
end

nki = P./mk;
```

In this section I am for looping through all the classes, images, and pixels of the new binary data. I then check if the number is greater than 0.5 to check if it is a 1 or a 0. If it is a 1 than I add 1 to P(i,k). Where i is each pixel in the image. So what this function is doing is taking all the counting the number of times a 1 shows up in each pixel in each image. So to find nki or the mean of each number I need to divide by the number of images (mk) which I did outside the for loop.

Plotting η_{ki}

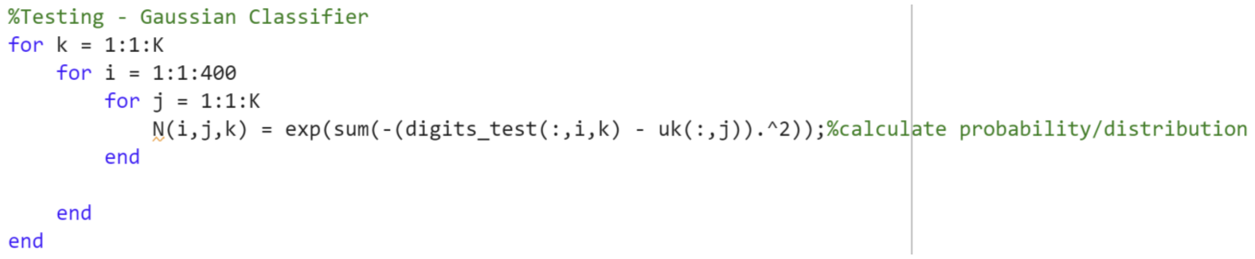
```
for i = 1:1:K
    subplot(2,5,i);
    imagesc(reshape(nki(:,i),8,8)'); axis equal; axis off; colormap gray;
end
```

Here, I did a very similar thing to plot ηki, as I did in the Gaussian classifier training. I then got the results as shown below for each of the digits.



Testing

Gaussian Classifier



```
%Testing - Gaussian Classifier
for k = 1:1:K
    for i = 1:1:400
        for j = 1:1:K
            N(i,j,k) = exp(sum(-(digits_test(:,i,k) - uk(:,j)).^2));%calculate probability/distribution
        end

    end
end
```

For the Gaussian classifier it uses the probability density function to find what class is most appropriate for the image. I decided to trim down the density function to be just what is shown in the above code this is because the variance is stated to be constant for all data points in this assignment as well as the number of pixels for each image. Because these numbers are constant, they are not needed in the math and will not have any affect on my classifier. I ran through 3 for loops in this section, one to go through all the actual classes for each of the test images, one for selecting each image, and the last one is to check through all the classes for the mean in order to determine what class the images is best suited for in terms of our training model.

```
[~,x] = max(N,[],2);%find the max probability at a certain index/class
```

In this next line of code, I found the maximum number in the distribution across its 2nd dimension, which in this case is the 400 images. What this does is it allows me to see what value for each of the means u_{ki}

is a maximum. And the [~,x] allows me to only save the maximum numbers index and not its actual value. I want the index as it is the class that the classifier is choosing for said image.

```
errorG(1:10) = 0;%initilize a 1x10 matrix of 0's
for k = 1:1:K
    for i = 1:1:400
        if x(i,1,k) ~= k
            errorG(k) = errorG(k) + 1; %add each time the algorithm classified a digit incorrectly
        end
    end
end
```

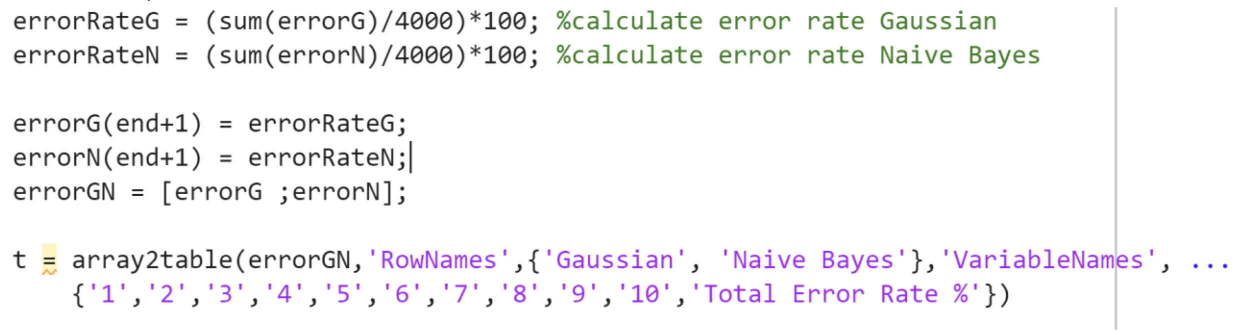
Lastly, I loop through the ‘x’ array that carries all the indices that our classifier guessed the digits were as well as carrying the data about what the correct class actually is. I can then easily loop through this array and compare the values to k. If it doesn’t equal the correct value than we add one to the error at the value of k. This way my errors are separated by class.

Naïve Bayes

```
%Testing Naive Bayes
Y = (digits_test>0.5); %Make test data Binary

for k = 1:1:K
      for i = 1:1:400
          for j = 1:1:K
             P(i,j,k) = prod((nki(:,j).^Y(:,i,k)).*(1-nki(:,j)).^(1-Y(:,i,k))); %calculate probabilit
          end
     end
end

[~,y] = max(P,[],2); %find the max probability at a certain index/class

errorN(1:10) = 0;
for k = 1:1:K
    for i = 1:1:400
        if y(i,1,k) ~= k
            errorN(k) = errorN(k) + 1; %add each time the algorithm classified a digit incorrectly
        end
    end
end
```

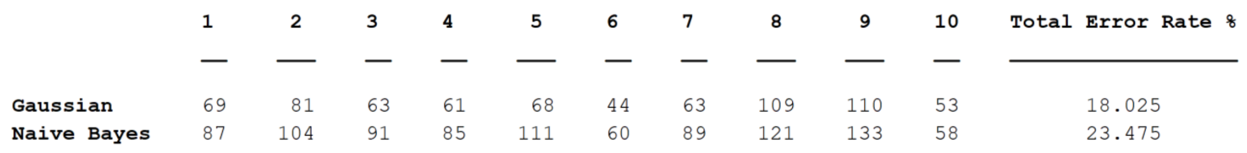
The testing of Naïve Bayes is very similar to that of the Gaussian classifier as seen and explained above. However, the main difference is the function that it uses to make a guess of the class that the images will be in. For Naïve bayes because it is binary it is a product function that uses nki or 1-nki depending on if the pixel is a 1 or a 0. So here I must loop through all our actual classes k, as well as each of the 400 images I, and lastly another loop through our classes to compare each of the nki for all classes against the test images. This is much like the Gaussian classifier. To choose a class the classifier must look at each of the nki probabilities for each class against the test images and choose the maximum value as the most likely class. The bottom for loop remains the same from the Gaussian classifier, and allows us to see the errors for each of the classes.

Table Comparison



```
errorRateG = (sum(errorG)/4000)*100; %calculate error rate Gaussian
errorRateN = (sum(errorN)/4000)*100; %calculate error rate Naive Bayes

errorG(end+1) = errorRateG;
errorN(end+1) = errorRateN;
errorGN = [errorG ;errorN];

t = array2table(errorGN,'RowNames',{'Gaussian', 'Naive Bayes'},'VariableNames', ...
    {'1','2','3','4','5','6','7','8','9','10','Total Error Rate %'})
```

Here I am setting up the test comparisons. I first start off by calculating the total error as a percentage for both the Gaussian and the Naïve Bayes classifiers. This is done by summing all the errors and dividing it by 4000 which is the total number of test cases. Next, I decided to add the total error rate to the end of my error arrays in order to have everything neatly in one table as seen below. The last line of code is setting up my table with row names and column names.



```
                    1      2      3      4      5      6      7      8      9     10     Total Error Rate %
                   __     ___    __     __     ___    __     __     ___    ___    __     __________________

    Gaussian       69      81    63     61      68    44     63     109    110    53           18.025
    Naive Bayes    87     104    91     85     111    60     89     121    133    58           23.475
```

Finally, we have finished testing our classifiers and we can see the results in the above table. Overall the Gaussian classifier worked better as it only had an error rate of 18.025% where as the Naïve Bayes had a higher error rate of 23.475%. Both classifiers seemed to definitely be working and classifying the large majority of images to the right numbers.