Andrew McClelland
October 19, 2018

ELEC 425 - Assignment 1

1 – Marginal and Conditional Numeric Distributions

Calculate and print out (or copy down) the following distributions. (Only show up to 4 decimal places)

1. *The marginal probability vector pC(x_1)*

   The output of the marginal probability vector can be found below in Figure 1, while it's code can be found below in Figure 2. The sum of the vector is 1, which validates the calculated values. To calculate it, x_2, x_3, and x_4 are summed out.

```
0.2039
0.1863
0.1943
0.2130
0.2026
```

Figure 1 - Marginal probability vector pC(x1)

```
%%% 1.1 - Marginal probability vector pC(x1)
pC_x1 = sum(sum(sum(pC, 2), 3), 4);
disp(pC_x1)
```

Figure 2 - Code for calculating marginal probability vector pC(x1)

2. *The conditional probability table pA(x_3, x_4 | x_1 takes its third value)*

   The conditional probability table of *pA(x_3, x_4 | x_1 = 3)* is stored in the variable *pA_x3_x4_given_x1_3rdValue* and can be found below in Figure 3. Its code can be found below in Figure 4. The sum of the table is 1, which validates the calculated values. To calculate it, x_2 was summed out, and x_1 was set to 3.

```
Q1 Part 1.2 - Conditional probability table pA(x3,x4 | x1 takes its third value) (x3 by x4 (columns are x4, rows are x3)
0.0411  0.0411  0.0411  0.0411  0.0411
0.0392  0.0392  0.0392  0.0392  0.0392
0.0381  0.0381  0.0381  0.0381  0.0381
0.0501  0.0501  0.0501  0.0501  0.0501
0.0611  0.0014  0.0442  0.0154  0.0356
```

Figure 3 - Conditional probability table pA(x3, x4 | x1 takes its third value)

```
%%% 1.2 - Conditional probability table pA(x3,x4 | x1 takes its third value)
pA_x2_x3_x4_given_x1_3rdValue = pA(3, :, :, :) / sum(sum(sum(pA(3, :, :, :), 2), 3), 4);
pA_x3_x4_given_x1_3rdValue = sum(pA_x2_x3_x4_given_x1_3rdValue, 2);
```

Figure 4 - Code for calculating conditional probability table

3. *The conditional probability vector pB(x_4 | x_2 takes its first value)*

   The *conditional probability vector pB(x_4 | x_2 = 1)* is stored in the variable *pB_x4_given_x2_1stValue* and can be found below in Figure 5. Its code can be found below in Figure 6. The sum of the table is 1, which validates the calculated values. To calculate it, x_1 and x_3 were summed out and x_2 was set to 1.

```
Q1 Part 1.3 - Conditional probability vector pB(x4 | x2 takes its first value
0.3300   0.1100   0.2200   0.0700   0.2700
```

Figure 5 - Conditional probability vector pB(x4 | x2 takes its first value)

```
%%% 1.3 - Conditional probability vector pB(x4 | x2 takes its first value)
pB_x1_x3_x4_given_x2_1stValue = pB(:, 1, :, :) / sum(sum(sum(pB(:, 1, :, :), 1), 3), 4);
pB_x4_given_x2_1stValue = sum(sum(pB_x1_x3_x4_given_x2_1stValue, 1), 2);
```

Figure 6 - Code for calculating conditional probability vector

For each distribution in {A,B,C} and each of the following statements, say whether the statement applies.

1. *x_1 is conditionally independent of x_2 given x_3* for distribution B, but not for distributions A or C. The output of the code for this question can be found below in Figure 7 while the code itself can be found below in Figure 8.

```
Part 2.1 - x1 is conditionally independent of x2 given x3 (which is p(x1, x2|x3) = p(x1|x3)p(x2|x3)
p(x1, x2|x3) ~= p(x1|x3)p(x2|x3) for pA distribution, so not conditionally independent
p(x1, x2|x3) == p(x1|x3)p(x2|x3) for pB distribution, so they are conditionally independent
p(x1, x2|x3) ~= p(x1|x3)p(x2|x3) for pC distribution, so not conditionally independent
```

Figure 7 – Output of code for Question 1, part 2.1

```
tolerance = 1e-5;

%%% 2.1 - x1 is conditionally independent of x2 given x3 (which is p(x1, x2|x3) = p(x1|x3)p(x2|x3)

% Calculate p(x1, x2, x4 | x3) for pA, pB, pC
pA_x1_x2_x4_given_x3 = pA ./ sum(sum(sum(pA(:, :, :, :), 1), 2), 4);
pB_x1_x2_x4_given_x3 = pB ./ sum(sum(sum(pB(:, :, :, :), 1), 2), 4);
pC_x1_x2_x4_given_x3 = pC ./ sum(sum(sum(pC(:, :, :, :), 1), 2), 4);

% Calculate p(x1, x2|x3) for pA, pB, pC
pA_x1_x2_given_x3 = sum(pA_x1_x2_x4_given_x3, 4);
pB_x1_x2_given_x3 = sum(pB_x1_x2_x4_given_x3, 4);
pC_x1_x2_given_x3 = sum(pC_x1_x2_x4_given_x3, 4);

% Calculate p(x1|x3) for pA, pB, pC
pA_x1_given_x3 = sum(sum(pA_x1_x2_x4_given_x3, 2), 4);
pB_x1_given_x3 = sum(sum(pB_x1_x2_x4_given_x3, 2), 4);
pC_x1_given_x3 = sum(sum(pC_x1_x2_x4_given_x3, 2), 4);

% Calculate p(x2|x3) for pA, pB, pC
pA_x2_given_x3 = sum(sum(pA_x1_x2_x4_given_x3, 1), 4);
pB_x2_given_x3 = sum(sum(pB_x1_x2_x4_given_x3, 1), 4);
pC_x2_given_x3 = sum(sum(pC_x1_x2_x4_given_x3, 1), 4);

% Calculate p(x1|x3)p(x2|x3) for pA, pB, pC
pA_x1_x2_given_x3_Calculated = pA_x1_given_x3 .* pA_x2_given_x3;
pB_x1_x2_given_x3_Calculated = pB_x1_given_x3 .* pB_x2_given_x3;
pC_x1_x2_given_x3_Calculated = pC_x1_given_x3 .* pC_x2_given_x3;

% Now determine conditional independence of each probability distribution pA, pB, pC

pA_condIndependent = true;
pB_condIndependent = true;
pC_condIndependent = true;

for x3 = 1:5
    if(abs(pA_x1_x2_given_x3(:, :, x3) - pA_x1_x2_given_x3_Calculated(:, :, x3)) > tolerance)
        pA_condIndependent = false;
    end

    if(abs(pB_x1_x2_given_x3(:, :, x3) - pB_x1_x2_given_x3_Calculated(:, :, x3)) > tolerance)
        pB_condIndependent = false;
    end

    if(abs(pC_x1_x2_given_x3(:, :, x3) - pC_x1_x2_given_x3_Calculated(:, :, x3)) > tolerance)
        pC_condIndependent = false;
    end
end

fprintf("Part 2.1 - x1 is conditionally independent of x2 given x3 (which is p(x1, x2|x3) = p(x1|x3)p(x2|x3)\n")
if(~pA_condIndependent)
    fprintf("p(x1, x2|x3) ~= p(x1|x3)p(x2|x3) for pA distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3) == p(x1|x3)p(x2|x3) for pA distribution, so they are conditionally independent\n")
end

if(~pB_condIndependent)
    fprintf("p(x1, x2|x3) ~= p(x1|x3)p(x2|x3) for pB distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3) == p(x1|x3)p(x2|x3) for pB distribution, so they are conditionally independent\n")
end

if(~pC_condIndependent)
    fprintf("p(x1, x2|x3) ~= p(x1|x3)p(x2|x3) for pC distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3) == p(x1|x3)p(x2|x3) for pC distribution, so they are conditionally independent\n")
end
```

Figure 8 - Code for Question 1, part 2.1

2. *x_1 is conditionally independent of x_2 given x_3 and x_4* for distributions A and B, but not for distribution C. The output of the code for this question can be found below in Figure 9 while the code itself can be found below in Figure 10.

```
Part 2.2 - x1 is conditionally independent of x2 given x3 and x4 (which is p(x1, x2|x3, x4) = p(x1|x3,x4)p(x2|x3,x4)
p(x1, x2|x3, x4) == p(x1|x3,x4)p(x2|x3,x4) for pA distribution, so they are conditionally independent
p(x1, x2|x3, x4) == p(x1|x3,x4)p(x2|x3,x4) for pB distribution, so they are conditionally independent
p(x1, x2|x3, x4) ~= p(x1|x3,x4)p(x2|x3,x4) for pC distribution, so not conditionally independent
```

Figure 9 - Output of code for Question 1, part 2.2

```
%%% 2.2 - x1 is conditionally independent of x2 given x3 and x4 (which is p(x1, x2|x3, x4) = p(x1|x3,x4)p(x2|x3,x4)

% Calculate p(x1, x2|x3, x4) for pA, pB, pC
pA_x1_x2_given_x3_x4 = pA ./ sum(sum(pA(:, :, :, :), 1), 2);
pB_x1_x2_given_x3_x4 = pB ./ sum(sum(pB(:, :, :, :), 1), 2);
pC_x1_x2_given_x3_x4 = pC ./ sum(sum(pC(:, :, :, :), 1), 2);

% Calculate p(x1|x3,x4) for pA, pB, pC
pA_x1_given_x3_x4 = sum(pA_x1_x2_given_x3_x4, 2);
pB_x1_given_x3_x4 = sum(pB_x1_x2_given_x3_x4, 2);
pC_x1_given_x3_x4 = sum(pC_x1_x2_given_x3_x4, 2);

% Calculate p(x2|x3,x4) for pA, pB, pC
pA_x2_given_x3_x4 = sum(pA_x1_x2_given_x3_x4, 1);
pB_x2_given_x3_x4 = sum(pB_x1_x2_given_x3_x4, 1);
pC_x2_given_x3_x4 = sum(pC_x1_x2_given_x3_x4, 1);

% Calculate p(x1|x3,x4)p(x2|x3,x4) for pA, pB, pC
pA_x1_x2_given_x3_x4_Calculated = pA_x1_given_x3_x4 .* pA_x2_given_x3_x4;
pB_x1_x2_given_x3_x4_Calculated = pB_x1_given_x3_x4 .* pB_x2_given_x3_x4;
pC_x1_x2_given_x3_x4_Calculated = pC_x1_given_x3_x4 .* pC_x2_given_x3_x4;

% Now determine conditional independence of each probability distribution pA, pB, pC

pA_condIndependent = true;
pB_condIndependent = true;
pC_condIndependent = true;

for x3 = 1:5
    for x4 = 1:5
        if(abs(pA_x1_x2_given_x3_x4(:, :, x3, x4) - pA_x1_x2_given_x3_x4_Calculated(:, :, x3, x4)) > tolerance)
            pA_condIndependent = false;
        end

        if(abs(pB_x1_x2_given_x3_x4(:, :, x3, x4) - pB_x1_x2_given_x3_x4_Calculated(:, :, x3, x4)) > tolerance)
            pB_condIndependent = false;
        end

        if(abs(pC_x1_x2_given_x3_x4(:, :, x3, x4) - pC_x1_x2_given_x3_x4_Calculated(:, :, x3, x4)) > tolerance)
            pC_condIndependent = false;
        end
    end
end

fprintf("\nPart 2.2 - x1 is conditionally independent of x2 given x3 and x4 (which is p(x1, x2|x3, x4) = p(x1|x3,x4)p(x2|x3,x4)\n")
if(~pA_condIndependent)
    fprintf("p(x1, x2|x3, x4) ~= p(x1|x3,x4)p(x2|x3,x4) for pA distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3, x4) == p(x1|x3,x4)p(x2|x3,x4) for pA distribution, so they are conditionally independent\n")
end

if(~pB_condIndependent)
    fprintf("p(x1, x2|x3, x4) ~= p(x1|x3,x4)p(x2|x3,x4) for pB distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3, x4) == p(x1|x3,x4)p(x2|x3,x4) for pB distribution, so they are conditionally independent\n")
end

if(~pC_condIndependent)
    fprintf("p(x1, x2|x3, x4) ~= p(x1|x3,x4)p(x2|x3,x4) for pC distribution, so not conditionally independent\n")
else
    fprintf("p(x1, x2|x3, x4) == p(x1|x3,x4)p(x2|x3,x4) for pC distribution, so they are conditionally independent\n")
end
```

Figure 10 - Code for Question 1, part 2.2

3. *x_1 is marginally independent of x_2* for distributions A, B and C. The output of the code for this question can be found below in Figure 11 while the code itself can be found below in Figure 12.

```
Part 2.3 - x1 is marginally independent of x2 (which is p(x1, x2) = p(x1)p(x2))
p(x1, x2) == p(x1)p(x2) for pA distribution, so is marginally independent
p(x1, x2) == p(x1)p(x2) for pB distribution, so is marginally independent
p(x1, x2) == p(x1)p(x2) for pC distribution, so is marginally independent
```

Figure 11 - Output of code for Question 1, part 2.3

```
%%% 2.3 - x1 is marginally independent of x2 (which is p(x1, x2) = p(x1)p(x2))

% Calculate p(x1, x2, x3, x4) for pA, pB, pC
pA_x1_x2_x3_x4 = pA ./ sum(sum(sum(sum(pA(:, :, :, :), 1), 2), 3), 4);
pB_x1_x2_x3_x4 = pB ./ sum(sum(sum(sum(pB(:, :, :, :), 1), 2), 3), 4);
pC_x1_x2_x3_x4 = pC ./ sum(sum(sum(sum(pC(:, :, :, :), 1), 2), 3), 4);

% Calculate p(x1, x2) for pA, pB, pC
pA_x1_x2 = sum(sum(pA_x1_x2_x3_x4, 3), 4);
pB_x1_x2 = sum(sum(pB_x1_x2_x3_x4, 3), 4);
pC_x1_x2 = sum(sum(pC_x1_x2_x3_x4, 3), 4);

% Calculate p(x1) for pA, pB, pC
pA_x1 = sum(sum(sum(pA, 2), 3), 4);
pB_x1 = sum(sum(sum(pB, 2), 3), 4);
pC_x1 = sum(sum(sum(pC, 2), 3), 4);

% Calculate p(x2) for pA, pB, pC
pA_x2 = sum(sum(sum(pA, 1), 3), 4);
pB_x2 = sum(sum(sum(pB, 1), 3), 4);
pC_x2 = sum(sum(sum(pC, 1), 3), 4);

% Calculate p(x1)p(x2) for pA, pB, pC
pA_x1_x2_Calculated = pA_x1 .* pA_x2;
pB_x1_x2_Calculated = pB_x1 .* pB_x2;
pC_x1_x2_Calculated = pC_x1 .* pC_x2;

% Now determine marginal independence of each probability distribution pA, pB, pC
fprintf("\nPart 2.3 - x1 is marginally independent of x2 (which is p(x1, x2) = p(x1)p(x2))\n")

if(abs(pA_x1_x2 - pA_x1_x2_Calculated) > tolerance)
    fprintf("p(x1, x2) != p(x1)p(x2) for pA distribution, so not marginally independent\n")
else
    fprintf("p(x1, x2) == p(x1)p(x2) for pA distribution, so is marginally independent\n")
end

if(abs(pB_x1_x2 - pB_x1_x2_Calculated) > tolerance)
    fprintf("p(x1, x2) != p(x1)p(x2) for pB distribution, so not marginally independent\n")
else
    fprintf("p(x1, x2) == p(x1)p(x2) for pB distribution, so is marginally independent\n")
end

if(abs(pC_x1_x2 - pC_x1_x2_Calculated) > tolerance)
    fprintf("p(x1, x2) != p(x1)p(x2) for pC distribution, so not marginally independent\n")
else
    fprintf("p(x1, x2) == p(x1)p(x2) for pC distribution, so is marginally independent\n")
end
```

Figure 12 - Code for Question 1, part 2.3

4. x_3 *is conditionally independent of* x_4 *given* x_1 *and* x_2 for distribution B, but not for distributions A or C. The output of the code for this question can be found below in Figure 13 while the code itself can be found below in Figure 14.

```
Part 2.4 - x3 is conditionally independent of x4 given x1 and x2 (which is p(x3, x4|x1, x2) = p(x3|x1,x2)p(x4|x1,x2))
p(x3, x4|x1, x2) ~= p(x3|x1,x2)p(x4|x1,x2) for pA distribution, so not conditionally independent
p(x3, x4|x1, x2) == p(x3|x1,x2)p(x4|x1,x2) for pB distribution, so they are conditionally independent
p(x3, x4|x1, x2) ~= p(x3|x1,x2)p(x4|x1,x2) for pC distribution, so not conditionally independent
```

Figure 13 - Output of code for Question 1, part 2.4

```
%%% 2.4 - x3 is conditionally independent of x4 given x1 and x2 (which is p(x3, x4|x1, x2) = p(x3|x1,x2)p(x4|x1,x2)

% Calculate p(x3, x4|x1, x2) for pA, pB, pC
pA_x3_x4_given_x1_x2 = pA ./ sum(sum(pA(:, :, :, :), 3), 4);
pB_x3_x4_given_x1_x2 = pB ./ sum(sum(pB(:, :, :, :), 3), 4);
pC_x3_x4_given_x1_x2 = pC ./ sum(sum(pC(:, :, :, :), 3), 4);

% Calculate p(x3|x1,x2) for pA, pB, pC
pA_x3_given_x1_x2 = sum(pA_x3_x4_given_x1_x2, 4);
pB_x3_given_x1_x2 = sum(pB_x3_x4_given_x1_x2, 4);
pC_x3_given_x1_x2 = sum(pC_x3_x4_given_x1_x2, 4);

% Calculate p(x4|x1,x2) for pA, pB, pC
pA_x4_given_x1_x2 = sum(pA_x3_x4_given_x1_x2, 3);
pB_x4_given_x1_x2 = sum(pB_x3_x4_given_x1_x2, 3);
pC_x4_given_x1_x2 = sum(pC_x3_x4_given_x1_x2, 3);

% Calculate p(x3|x1,x2)p(x4|x1,x2) for pA, pB, pC
pA_x3_x4_given_x1_x2_Calculated = pA_x3_given_x1_x2 .* pA_x4_given_x1_x2;
pB_x3_x4_given_x1_x2_Calculated = pB_x3_given_x1_x2 .* pB_x4_given_x1_x2;
pC_x3_x4_given_x1_x2_Calculated = pC_x3_given_x1_x2 .* pC_x4_given_x1_x2;

% Now determine conditional independence of each probability distribution pA, pB, pC

pA_condIndependent = true;
pB_condIndependent = true;
pC_condIndependent = true;

for x1 = 1:5
    for x2 = 1:5
        if(abs(pA_x3_x4_given_x1_x2(x1, x2, :, :) - pA_x3_x4_given_x1_x2_Calculated(x1, x2, :, :)) > tolerance)
            pA_condIndependent = false;
        end

        if(abs(pB_x3_x4_given_x1_x2(x1, x2, :, :) - pB_x3_x4_given_x1_x2_Calculated(x1, x2, :, :)) > tolerance)
            pB_condIndependent = false;
        end

        if(abs(pC_x3_x4_given_x1_x2(x1, x2, :, :) - pC_x3_x4_given_x1_x2_Calculated(x1, x2, :, :)) > tolerance)
            pC_condIndependent = false;
        end
    end
end

fprintf("\nPart 2.4 - x3 is conditionally independent of x4 given x1 and x2 (which is p(x3, x4|x1, x2) = p(x3|x1,x2)p(x4|x1,x2))\n")

if(~pA_condIndependent)
    fprintf("p(x3, x4|x1, x2) ~= p(x3|x1,x2)p(x4|x1,x2) for pA distribution, so not conditionally independent\n")
else
    fprintf("p(x3, x4|x1, x2) == p(x3|x1,x2)p(x4|x1,x2) for pA distribution, so they are conditionally independent\n")
end

if(~pB_condIndependent)
    fprintf("p(x3, x4|x1, x2) ~= p(x3|x1,x2)p(x4|x1,x2) for pB distribution, so not conditionally independent\n")
else
    fprintf("p(x3, x4|x1, x2) == p(x3|x1,x2)p(x4|x1,x2) for pB distribution, so they are conditionally independent\n")
end

if(~pC_condIndependent)
    fprintf("p(x3, x4|x1, x2) ~= p(x3|x1,x2)p(x4|x1,x2) for pC distribution, so not conditionally independent\n")
else
    fprintf("p(x3, x4|x1, x2) == p(x3|x1,x2)p(x4|x1,x2) for pC distribution, so they are conditionally independent\n")
end
```

Figure 14 - Code for Question 1, part 2.4

2 – Training Conditional Gaussian Classifiers

The plotted output of the trained Conditional Gaussian Classifier can be found below in Figure 15. As shown on the plot, the pixel noise standard deviation is roughly 0.2517. The function that trains the Conditional Gaussian Classifier (*trainConditionalGaussian())* can be found below in Figure 16, while the script that plots the output can be found below in Figure 17. The $\hat{u}_{ki}$ parameters are stored in *mean_featureI_classK* in both the function and plotting script code.

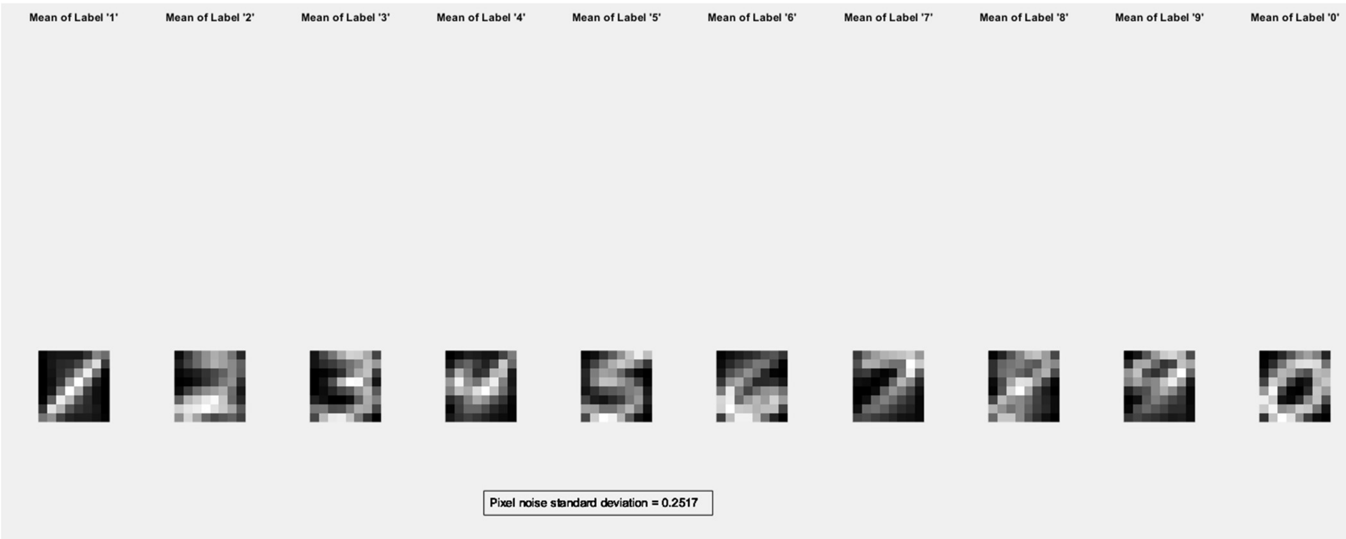

Figure 15 - Plot of handwritten digits from Conditional Gaussian Classifier

```matlab
function [variance, mean_featureI_classK] = trainConditionalGaussian()
%% Training Conditional Gaussian Classifiers

clear all;
close all;
clc;

% 64 (8x8 image of digit in raster scan order) x 700 train cases x 10
% digit labels (1-0) where label 10 is 0
load('./data/a1digits.mat');

%% Training

% Class label k in [1,2...K]
% Real value vector of D features x = (x1, ... xD)

% number of training data points in class k
mk = 700;

% uki mean of feature i condiitoned on class k. 2D array where k (labels
% 0-9) and i (features 1-64)
mean_featureI_classK = zeros(10,64);

% Loop through each class k (1:10), and for each feature i (1:64), sum its
% value in each of the k training datapoints and divide by total number of
% training datapoints mk
for class_k = 1:10
    for feature_i = 1:64
        mean_featureI_classK(class_k, feature_i) = mean(digits_train(feature_i, :, class_k));
    end
end

% Calculating variance
variance = 0;

% Loop through each class k, training point j, and feature i, to calculate
% variance
for class_k = 1:10
    for trainingPoint_j = 1:700
        for feature_i = 1:64
            variance = variance + ((digits_train(feature_i, trainingPoint_j, class_k) - mean_featureI_classK(class_k, feature_i))^2);
        end
    end
end

% Divide by DM (64 features * (700 training samples * 10 classes))
variance = variance / (64 * (700 * 10));

end
```

Figure 16 - Training function code for Q2 Conditional Gaussian Classifier

```
%% Answer to Question 2 - Training Conditional Gaussian Classifiers

clear all;
close all;
clc;

%% Training Gaussian

% Call Gaussian training function and retrieve variance and u_ki
[variance, mean_featureI_classK] = trainConditionalGaussian();

% Create subplot of 1x10 that plots each class with correct label
for class_k = 1:10
    subplot(1,10,class_k);
    imagesc(reshape(mean_featureI_classK(class_k,:),8,8)'); axis equal; axis off; colormap gray;
    title(strcat("Mean of Label '", num2str(mod(class_k, 10)), "'"))
    annotation('textbox', [.4 .1 .3 .3], 'String', strcat("Pixel noise standard deviation = ", num2str(sqrt(variance))), 'FitBoxToText','on');
end

% Set figure to fullscreen
set(gcf,'units','normalized','outerposition',[0 0 1 1])
```

Figure 17 - Plotting code for Q2 Conditional Gaussian Classifier

3 – Training Naïve Bayes Classifiers

The plotted output of the trained Naïve Bayes Classifier can be found below in Figure 18. The function that trains the Naïve Bayes Classifier (*trainNaiveBayes()*) can be found below in Figure 19, while the script that plots the output can be found below in Figure 20. The n_{ki} parameters are stored in *mean_featureI_classK* in the function and plotting script code.

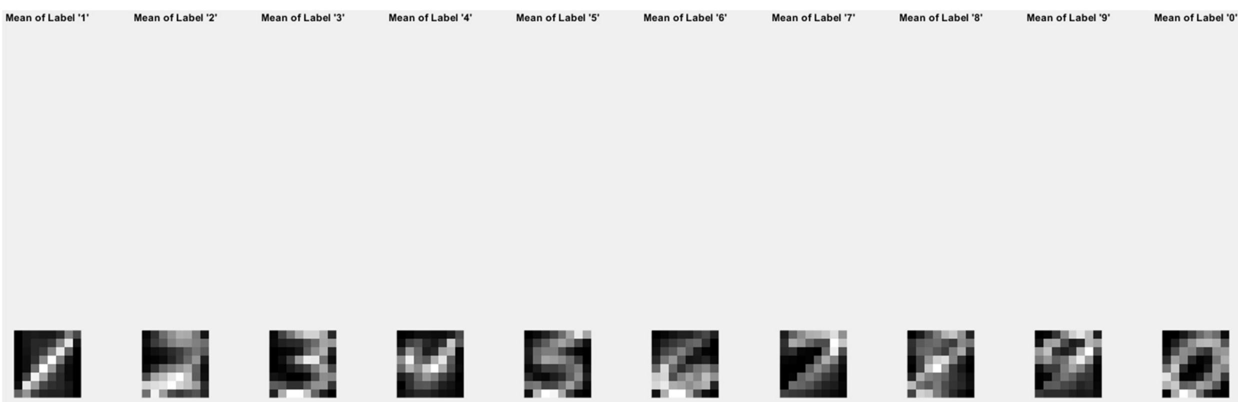

Figure 18 - Plot of handwritten digits from trained Naive Bayes Classifier

```
function [mean_featureI_classK] = trainNaiveBayes()

% 64 (8x8 image of digit in raster scan order) x 700 train cases x 10
% digit labels (1-0) where label 10 is 0
load('./data/a1digits.mat');

%% Training

% Convert real-valued features x into binary features b by thresholding: bi = 1 if xi > 0.5 otherwise bi = 0
digits_train_thresholded = zeros(64, 700, 10);

for class_k = 1:10
    for trainingPoint_j = 1:700
        digits_train_thresholded(:, trainingPoint_j, class_k) = digits_train(:, trainingPoint_j, class_k) > 0.5;
    end
end

% nki is 2D array where k (labels 0-9) and i (features 1-64)
mean_featureI_classK = ones(10,64);

% For each class k and feature i, calculate n_ki by calculating p(b_i=1 | Ck)
for class_k = 1:10
    for feature_i = 1:64
        mean_featureI_classK(class_k, feature_i) = sum(digits_train_thresholded(feature_i, :, class_k)) / 700;
    end
end
```

Figure 19 - Training function code for Q3 Naive Bayes Classifier

```
%% Answer to Question 3 - Training Naive Bayes Classifiers

clear all;
close all;
clc;

%% Training Naive Bayes

% Call Naive Bayes training function and retrieve n_ki
mean_featureI_classK = trainNaiveBayes();

% Create subplot of 1x10 that plots each class with correct label
for class_k = 1:10
    subplot(1,10,class_k);
    imagesc(reshape(mean_featureI_classK(class_k,:),8,8)'); axis equal; axis off; colormap gray;
    title(strcat("Mean of Label '", num2str(mod(class_k, 10)), "'"))
end

% Set figure to fullscreen
set(gcf,'units','normalized','outerposition',[0 0 1 1])
```

Figure 20 - Plotting code for Q3 Naive Bayes Classifier

4 – Test Performance

The output of the test performance of both the Conditional Gaussian and Naïve Bayes Classifiers can be found below in Figure 21. From the output, we can see that the Conditional Gaussian Classifier achieves an error rate of ~18.02% while the Naïve Bayes Classifier achieves and error rate of ~23.47%. The codes that tests the Conditional Gaussian Classifier can be found below in Figure 22, while the code that tests the Naïve Bayes Classifier can be found below in Figure 23.

```
Errors for Gaussian and Naive Bayes Classification
    - First row is Gaussian, second row is Naive Bayes
    - Columns 1-10 are # errors for each class 1-10
    - Column 11 is error rate for each classifier

   69.0000   81.0000   63.0000   61.0000   68.0000   44.0000   63.0000  109.0000  110.0000   53.0000    0.1802
   87.0000  104.0000   91.0000   85.0000  111.0000   60.0000   89.0000  121.0000  133.0000   58.0000    0.2347
```

Figure 21 - Error rate for Conditional Gaussian and Naive Bayes Classifiers

```matlab
%% Answer to Question 4 - Test Performance

clear all;
close all;
clc;

load('./data/a1digits.mat');

% Create 2 x 11 array where row1 is Gaussian, row2 is Naive Bayes, column represents label 1:10, last column has overall error rate in %
error_rate = zeros(2, 11);

% ak = 1/10 since all classes have same number of observations
prior_class_prob = 1/10;

%% Testing Conditional Gaussian

% Get the variance and mean of Gaussian training function
[variance, mean_featureI_classK] = trainConditionalGaussian();

% Loop through all testpoints for each class
for testPoint_j = 1:400
    for class_k = 1:10
        % Create 1x10 array to hold the calculated probability that a given test point belongs to each of the 10 classes, and loop through
        % each class to calculate its probability
        class_conditional_dist = zeros(1, 10);
        for class_guess = 1:10
            sum_part = 0;
            % Loop through each feature and calculate the relvant part of p(x|Ck) for each class
            for feature_i = 1:64
                sum_part = sum_part + (digits_test(feature_i, testPoint_j, class_k) - mean_featureI_classK(class_guess, feature_i))^2;
            end
            class_conditional_dist(class_guess) = exp((-1/(2 * variance)) * sum_part);
        end
        % Select the highest probability class and guess this
        [val, predicted_class] = max(class_conditional_dist);
        % If prediction is incorrect, add error count to what class it should have been
        if(predicted_class ~= class_k)
            error_rate(1, class_k) = error_rate(1, class_k) + 1;
        end
    end
end
```

Figure 22 - Code to test Conditional Gaussian Classifier in Q4

```matlab
%% Testing Naive Bayes

% Convert real-valued features x into binary features b by thresholding: bi = 1 if xi > 0.5 otherwise bi = 0
digits_test_thresholded = zeros(64, 400, 10);

% Convert real-valued features x into binary features b by thresholding: bi = 1 if xi > 0.5 otherwise bi = 0
for class_k = 1:10
    for testPoint_j = 1:400
        digits_test_thresholded(:, testPoint_j, class_k) = digits_test(:, testPoint_j, class_k) > 0.5;
    end
end

% Get nki values for classes 1-10 and features 1-64
mean_featureI_classK = trainNaiveBayes();

% Loop through all test points and classes
for testPoint_j = 1:400
    for class_k = 1:10
        % Create 1x10 array to hold the calculated probability that a given test point belongs to each of the 10 classes, and loop through
        % each class to calculate its probability
        prob_each_class = ones(1, 10);
        for class_guess = 1:10
            for feature_i = 1:64
                % If current feature is 1, multiply current probability by nki
                if(digits_test_thresholded(feature_i, testPoint_j, class_k))
                    prob_each_class(1, class_guess) = prob_each_class(1, class_guess) * mean_featureI_classK(class_guess, feature_i);
                % If current feature is 0, multiply current probability by (1-nki)
                else
                    prob_each_class(1, class_guess) = prob_each_class(1, class_guess) * (1 - mean_featureI_classK(class_guess, feature_i));
                end
            end
        end
        % Select the highest probability class and guess this
        [val, predicted_class] = max(prob_each_class);
        % If prediction is incorrect, add error count to what class it should have been
        if(predicted_class ~= class_k)
            error_rate(2, class_k) = error_rate(2, class_k) + 1;
        end
    end
end

%% Calculate error rate for both Gaussian and Naive Bayes
error_rate(1, 11) = sum(error_rate(1, :)) / (400 * 10);
error_rate(2, 11) = sum(error_rate(2, :)) / (400 * 10);
```

Figure 23 - Code to test Naive Bayes Classifier in Q4