

Assignment 1, COMP4702

Roy Portas, 43560846

March 30, 2017

Question 1.2

The first column of the data is a timestamp, with each value being around 30 minutes apart. The data starts at the first of January 2015 and ends in the last day of December.

The second column appears to be a unique ID for each entry, as this number is never repeated and goes up by 1 for each record.

The third column contains numbers between 25 and 30, this is possibly a temperature in degrees. The values also change gradually which seems correct for the given time intervals.

Viewing data on the graph shows that the temperature is low at the start of the year and gradually climbs to it's max around 6 months in, then returns back to a lower temperature in December. This suggests that it is probably temperature measured in the northern hemisphere. Additionally this data has outliers, which is probably instrumentation error.

The fourth column contains numbers between 26 and around 50000. If this is plotted against the date, it can be seen that there are three distinct dips in the data. Two of the dips are at the start of the year and the last is towards the end.

The fifth column contains numbers between 7.3 and 8.3, with a mean of 7.846 and a standard StdDev of 0.142. This suggests that the value doesn't change much. Viewing the data over the date does not show any trends in the data.

When the sixth column is plotted against the fifth column, a linear relationship can be seen, as the values in the sixth column increases when the values of the fifth column increases. This indicates there is some form of relationship between two sets of data.

The last two columns don't provide much information, apart from increased outliers at the end of the year, suggesting that the sensors may be starting to fail.

It could possibly be weather data, containing temperature, humidity, etc.

Question 1.6

```
1 % in is the input array
2 % n is the group size
3 function out = q6(in, n)
4
5     out = [];
6
7     chunks = length(in)/n;
8
9     for i = 1:chunks
```

```

10     end_ind = length(in) - i * n + n;
11
12     start_ind = end_ind - n + 1;
13     start_ind = max([1, start_ind]);
14
15     temp = in(start_ind:end_ind);
16
17     out = [out, temp];
18 end
19
20 end

```

Question 2.1

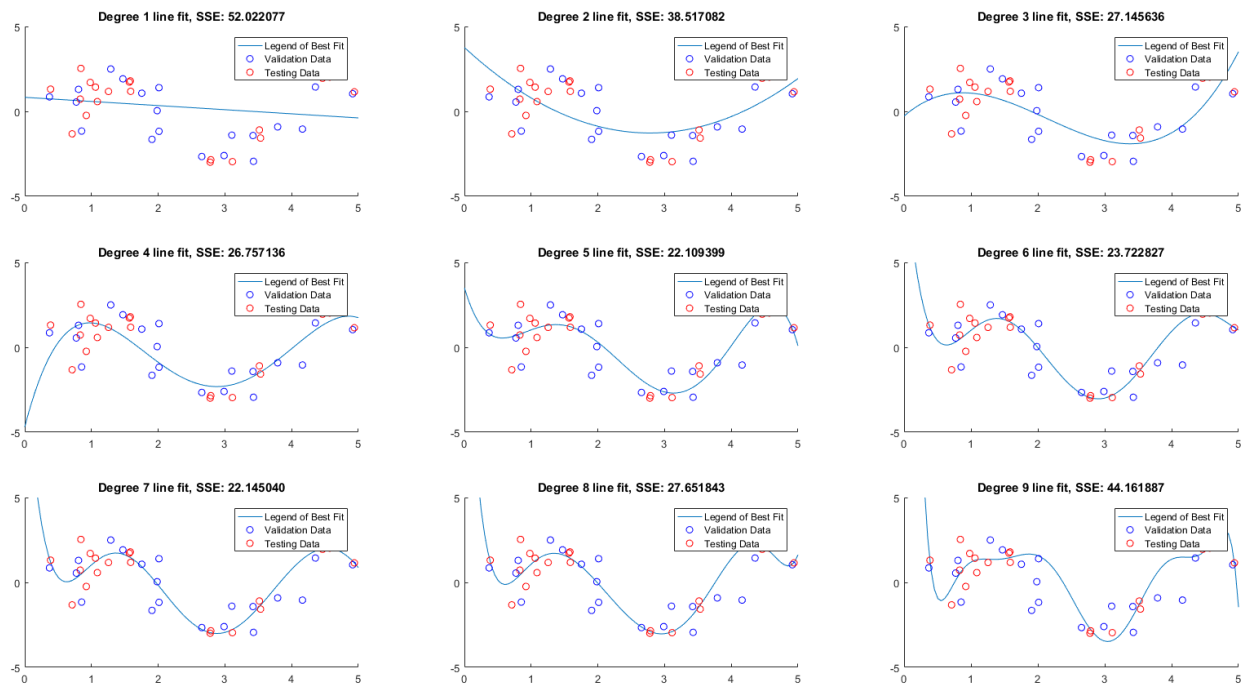


Figure 1: Lines of best fit

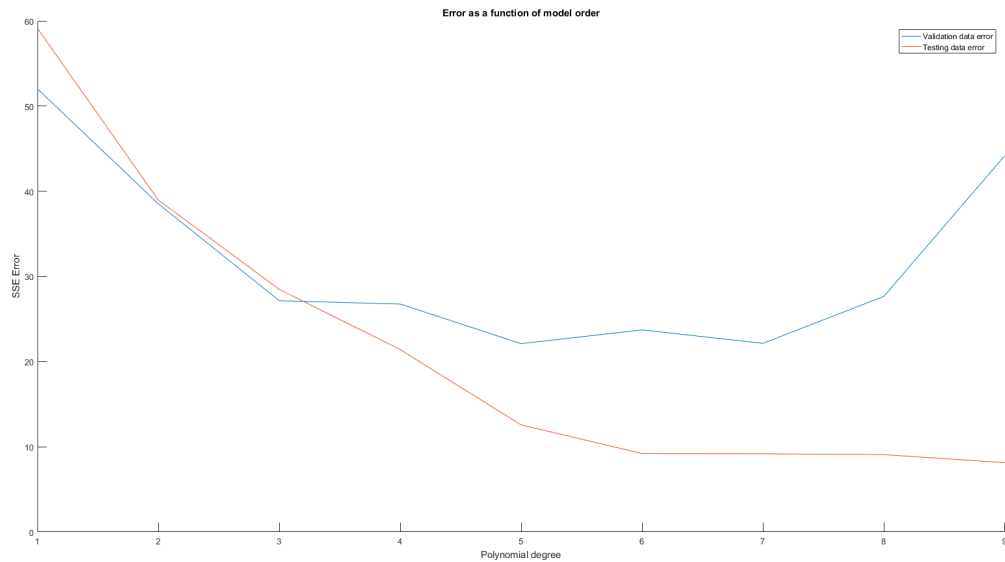


Figure 2: Error vs Polynomial Degree

The above figure shows that as more polynomial degrees are introduced the error on the testing data decreases, which is expected as this was the data the model was trained on. However the error on the validation data increases after the 7th polynomial, thus we are overfitting the data.

Question 2.4

```

1 function q4(data, class)
2     class_names = unique(class);
3     x_values = 1:length(class);
4
5     class1 = zeros(1, length(class));
6     class2 = zeros(1, length(class));
7
8     for i = x_values
9         if strcmp(class{i}, class_names{1})
10             class1(i) = 1;
11         else
12             class2(i) = 1;
13         end
14     end
15
16     % Verify the classes are correct

```

```

17 % figure;
18 % hold on;
19 % scatter(1:length(class1), class1);
20 % scatter(1:length(class2), class2);
21 % hold off;
22
23 estimate_range = 1:0.1:8;
24
25 class1_data = data;
26 class1_data(class2 == 1) = NaN;
27
28 class1_mle = mle(class1_data);
29 class1_pdf = normpdf(estimate_range, class1_mle(1), class1_mle
    (2));
30
31 class2_data = data;
32 class2_data(class1 == 1) = NaN;
33
34 class2_mle = mle(class2_data);
35 class2_pdf = normpdf(estimate_range, class2_mle(1), class2_mle
    (2));
36
37
38 % figure;
39
40 % scatter(x_values, data);
41
42 % Verify the classes are divided
43 figure;
44
45 hold on;
46
47 yyaxis left;
48 scatter(class1_data, x_values, 'r');
49 scatter(class2_data, x_values, 'b');
50
51 yyaxis right;
52 plot(estimate_range, class1_pdf, 'r');
53 plot(estimate_range, class2_pdf, 'b');
54 legend('Iris Setosa', 'Iris Versicolor');
55 hold off;
56
57 % Plot the likelihood
58 figure;
59 hold on;

```

```

60 plot(estimate_range , class1_pdf);
61 plot(estimate_range , class2_pdf);
62 xlim([1 , 10]);
63
64 title('Likelihoods');
65 xlabel('x');
66 ylabel('P(x|C_i)');
67
68 p_class1 = class1_pdf ./ (class1_pdf + class2_pdf);
69 p_class2 = class2_pdf ./ (class1_pdf + class2_pdf);
70
71 figure;
72 hold on
73
74 plot(estimate_range , p_class1);
75 plot(estimate_range , p_class2);
76
77 title('Posteriors');
78 xlabel('x');
79 ylabel('P(x|C_i)');
80
81 hold off;
82
83 end

```

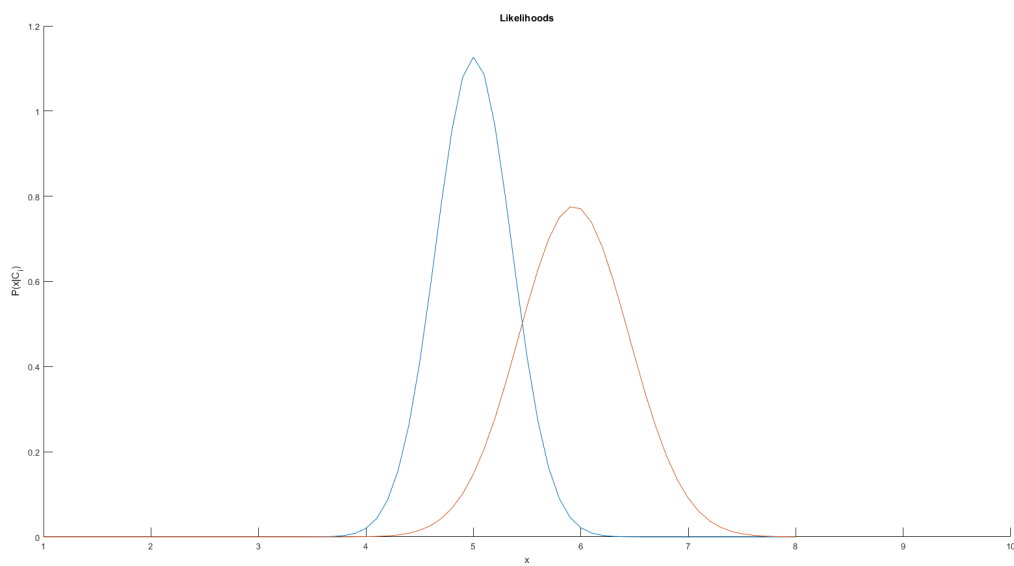


Figure 3: Likelihoods

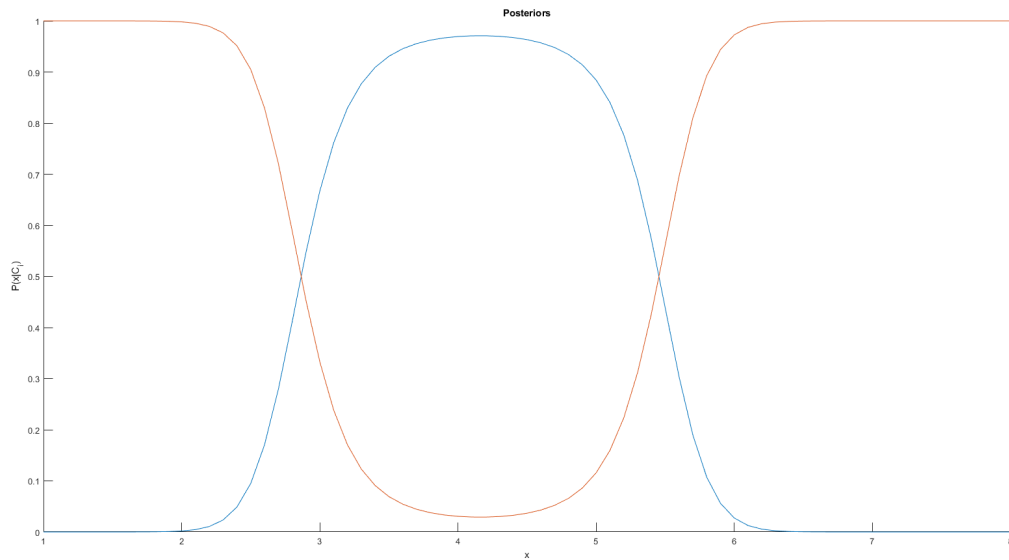


Figure 4: Posteriors

Question 3.1

With the given dataset, we want to find a way to classify the two classes, this can be given by the posterior $P(C_i|x)$. To do so we first need to estimate the prior $P(C_i)$ and $p(x|C)$. This will allow us to estimate the proportion of data points for a given class i . $p(x|c)$ can easily be found by using the matlab function `mvnpdf`.

Finding the posteriors for each class yields the below graph.

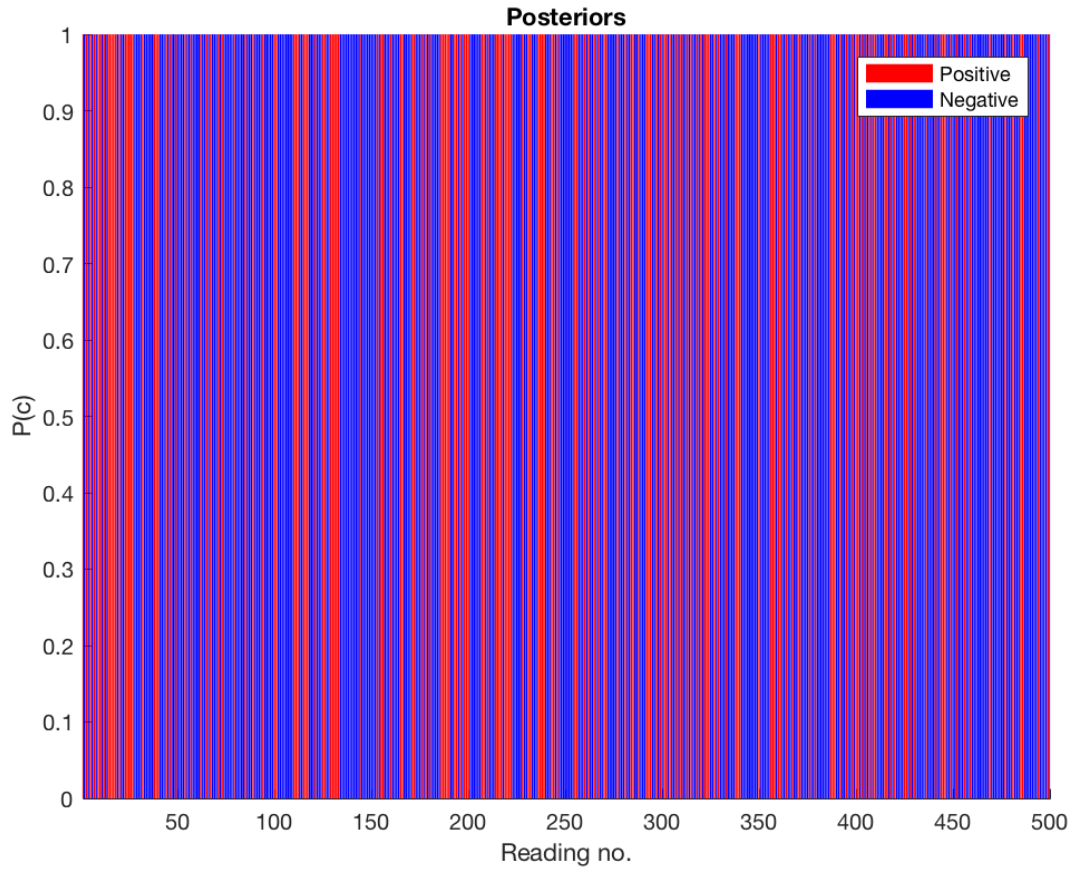


Figure 5: Posteriors

This shows the classification of the algorithm, visually we can see that there is more blue than red, meaning more people test negative to diabetes than positive, which makes sense when you look at the raw data.

The error on the training set was found to be 330.42, while the error on the testing set was found to be 181.18. This is possibly caused by the training set containing many more data rows than the testing set.

The model parameters were found to be the following:

```

1 const =
2
3     14.5107
4
5
6 linear =
7
8     -0.0230

```



```

9      0.0030
10     0.0465
11     0.0089
12    -0.0099
13    -0.3704
14     1.7751
15    -0.2770
16
17
18 quadratic =
19
20 Columns 1 through 6
21
22    -0.0388    -0.0004     0.0002    -0.0014     0.0002     0.0012
23    -0.0004    -0.0003     0.0001    -0.0001     0.0001     0.0000
24     0.0002     0.0001    -0.0008    -0.0003     0.0000     0.0015
25    -0.0014    -0.0001    -0.0003    -0.0004    -0.0000     0.0011
26     0.0002     0.0001     0.0000    -0.0000    -0.0000     0.0002
27     0.0012     0.0000     0.0015     0.0011     0.0002    -0.0019
28    -0.0717    -0.0056     0.0040    -0.0262     0.0064    -0.0130
29     0.0043     0.0005    -0.0004     0.0008    -0.0003     0.0016
30
31 Columns 7 through 8
32
33    -0.0717     0.0043
34    -0.0056     0.0005
35     0.0040    -0.0004
36    -0.0262     0.0008
37     0.0064    -0.0003
38    -0.0130     0.0016
39    -2.5106     0.0301
40     0.0301     0.0005

```

Question 3.2

LDA can be implemented similiar to the QDA done above.

```

1 %
2 % Q2
3 % Linear Discriminant Analysis
4 %
5
6 % General Steps

```

```

7 % 1. Compute d-dimensional means vectors for each class
8 % 2. Compute 'Scatter' matrices in between class and within class
9 % 3. Compute eigenvectors and corresponding eigenvalues for the
    scatter matrices
10 % 4. Sort the eigenvectors by decreasing eigenvalues and choose
    the largest to form a matrix (W)
11 % 5. Use the new matrix to transform the samples onto the new
    subspace

12
13 %% Step 0: Load the data
14 raw = readtable('pima_indians_diabetes.csv');
15
16 raw_data = raw(:, 1:8);
17
18 % The classified classes, e.g. positive or negative
19 actual = table2array(raw(:, 9));
20 training_actual = actual(1:500);
21 testing_actual = actual(500:end);
22
23 % Convert the table into an array
24 data = table2array(raw_data);
25
26 % Split into training and testing
27 training = data(1:500, :);
28 testing = data(500:end, :);
29
30 % Divide training data into positive and negative classes
31
32 % An array of data
33 training_positive = NaN(500, 8);
34 training_negative = NaN(500, 8);
35
36 % Binary array of samples
37 positive = zeros(1, 500);
38 negative = zeros(1, 500);
39 for i = 1:500
40     if strcmp(training_actual(i), 'pos')
41         positive(i) = 1;
42         training_positive(i, :) = training(i, :);
43     else
44         negative(i) = 1;
45         training_negative(i, :) = training(i, :);
46     end
47 end
48

```

```

49
50 pos_testing = zeros(1, length(testing_actual));
51 neg_testing = zeros(1, length(testing_actual));
52
53 % Parse the testing data
54 for i = 1:length(testing_actual)
55     if strcmp(testing_actual(i), 'pos')
56         pos_testing(i) = 1;
57     else
58         neg_testing(i) = 1;
59     end
60
61 end
62
63 %% Step 1: Calculate the mean vectors
64
65 % Class 1 mean vector
66 pos_mean_vector = nanmean(training_positive);
67
68 % Class 2 mean vector
69 neg_mean_vector = nanmean(training_negative);
70
71 %% Step 2: Compute scatter matrices
72 % These will be 8x8 matrices for each class
73
74 positive_scatter_matrix = zeros(8, 8);
75 for i = 1:length(training_positive)
76     mv = transpose(pos_mean_vector);
77     row = transpose(training_positive(i, :));
78
79     if not(any(isnan(row)))
80         positive_scatter_matrix = positive_scatter_matrix + (row -
            mv) * (transpose(row - mv));
81     end
82 end
83
84 within_class_scatter_matrix = positive_scatter_matrix;
85
86 negative_scatter_matrix = zeros(8, 8);
87 for i = 1:length(training_negative)
88     mv = transpose(neg_mean_vector);
89     row = transpose(training_negative(i, :));
90
91     if not(any(isnan(row)))

```

```

92         negative_scatter_matrix = negative_scatter_matrix + (row -
          mv) * (transpose(row - mv));
93     end
94 end
95
96 within_class_scatter_matrix = within_class_scatter_matrix +
    negative_scatter_matrix;
97
98 % Calculate the between class scatter matrix
99 between_class_scatter_matrix = zeros(8, 8);
100
101 overall_mean = transpose(mean(training));
102
103 % Do it for positive first
104 mv = transpose(pos_mean_vector);
105
106 n = length(find(all(~isnan(training_positive), 2)));
107
108 temp = n * (mv - overall_mean) * transpose(mv - overall_mean);
109
110 between_class_scatter_matrix = between_class_scatter_matrix + temp
    ;
111
112 % Do it for negative
113 mv = transpose(neg_mean_vector);
114
115 n = length(find(all(~isnan(training_negative), 2)));
116
117 temp = n * (mv - overall_mean) * transpose(mv - overall_mean);
118
119 between_class_scatter_matrix = between_class_scatter_matrix + temp
    ;
120
121 %% Step 3: Calculate the eigenvectors and eigenvalues
122
123 e = eig(inv(within_class_scatter_matrix) *
    between_class_scatter_matrix);
124 [V, D] = eig(inv(within_class_scatter_matrix) *
    between_class_scatter_matrix);
125
126 %% Step 4: Sort the eigenvectors and eigenvalues by decreasing
    order
127 [a, b] = sort(e, 'descend');
128
129 best_eig_value = a(1);

```

```

130 best_eig_vector = V(b(1), :);
131
132 second_best_eig_value = a(2);
133 second_best_eig_vector = V(b(2), :);
134
135 W = zeros(8, 2);
136
137 W(:, 1) = transpose(best_eig_vector);
138 W(:, 2) = transpose(second_best_eig_vector);
139
140 % The W matrix
141
142 real(W)
143
144 % Multiply the training data with W to classify
145 training_lda = training * W;

```

The error for the training set was found to be 319.41 and the error for the testing set was found to be 178.59.

The model parameters are listed below

```

1  const =
2
3      7.4066
4
5
6  linear =
7
8      -0.1251
9      -0.0319
10     0.0114
11     0.0041
12     0.0008
13    -0.0835
14    -0.8727
15    -0.0033

```

Question 3.5

The computed KL values are listed in the table below

M and H1	2.6898
M and K1	0.2741
M and K2	0.28186

There was an issue encountered while finding these variables, it was caused by the 0 values in some of the bins of the histogram estimator, this caused the division to produce Infinity values. This was fixed by replacing the infinity values with 0.

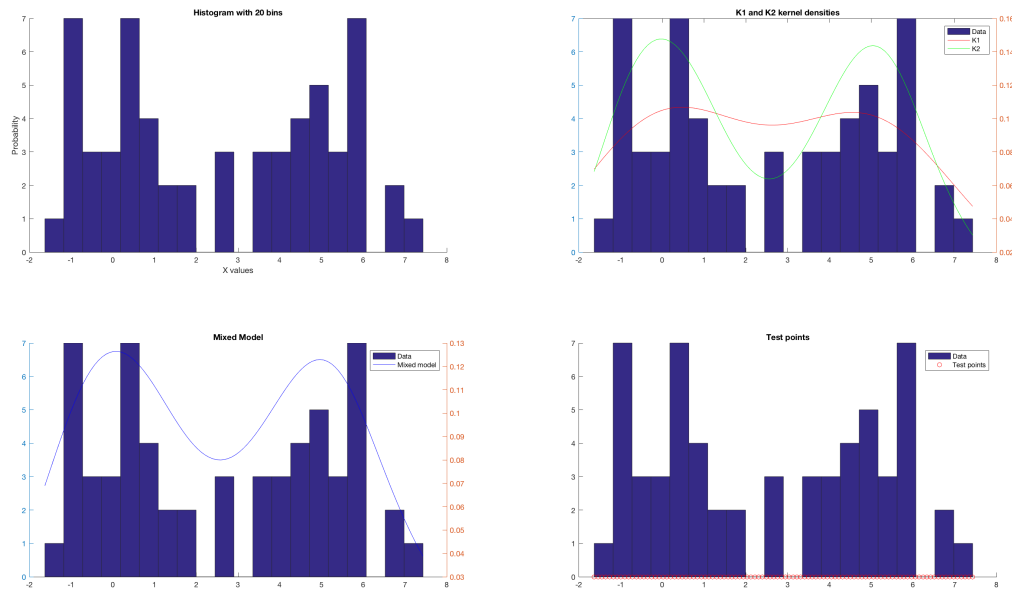


Figure 6: Q5

Below is the code used to calculate the KL function, see the function called KL at the bottom of the script.

```

1 % Q5
2
3 x = [randn(30, 1); 5 + randn(30, 1)];
4
5 subplot(2, 2, 1);
6 hold on;
7 hist(x, 20);
8 title('Histogram with 20 bins');
9 xlabel('X values');
10 ylabel('Probability');
11 hold off;
12
13 % Demo the results

```

```

14 subplot(2, 2, 2);
15 yyaxis left;
16 hist(x, 20);
17 yyaxis right;
18
19 % Generate 100 random datapoints between that covers the range of
    data in x
20 x_points = min(x):((max(x)-min(x))/99):max(x);
21
22 % Generate the histogram bin counts
23 [dist, edges] = histcounts(x, 20);
24 N = sum(dist);
25 x0 = edges(1);
26 h = edges(2) - edges(1);
27
28 data = zeros(1, 100);
29
30 % Histogram estimator
31 for i = 1:100
32     bin = ceil((x_points(i) - x0) / h);
33     x_in_bin = dist(bin);
34
35     data(i) = x_in_bin / (N * h);
36 end
37
38
39 % Create the kernel density estimators
40
41 % Default kernel width
42 [K1, k1x, bw1] = ksdensity(x, x_points);
43
44 % Half of the default kernel width
45 [K2, k2x, bw2] = ksdensity(x, x_points, 'width', bw1/2);
46
47 hold on;
48 plot(k1x, K1, 'r');
49 plot(k2x, K2, 'g');
50
51 title('K1 and K2 kernel densities');
52 legend('Data', 'K1', 'K2');
53 hold off;
54
55 % Calculate the Guassian mixed model
56 mixed = K1/2 + K2/2;
57

```

```

58 subplot(2, 2, 3);
59 hold on;
60 yyaxis left;
61 hist(x, 20);
62 yyaxis right;
63 plot (k1x, mixed, 'b');
64
65 title('Mixed Model');
66 legend('Data', 'Mixed model');
67 hold off;
68
69 subplot(2, 2, 4);
70 hold on;
71 hist(x, 20);
72 scatter(x_points, zeros(1, 100), 'r');
73
74 title('Test points');
75 legend('Data', 'Test points');
76 hold off;
77
78 % Do the calculations
79 KL(mixed, data)
80 KL(mixed, K1)
81 KL(mixed, K2)
82
83 %
84 % Calculate the KL Divergence
85 %
86 function result = KL(p, q)
87     result = zeros(size(p));
88
89     valid = p > 0 & q > 0;
90
91     result1 = sum(p(valid) .* log(p(valid) ./ q(valid)));
92     result2 = sum(q(valid) .* log(q(valid) ./ p(valid)));
93
94     result(valid) = result1 + result2;
95
96     result = mode(result);
97 end

```