Assignment 1, COMP4702

Roy Portas

March 29, 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

```
% in is the input array
% n is the group size
function out = q6(in, n)

    out = [];

    chunks = length(in)/n;

    for i = 1:chunks
```

```
        end_ind = length(in) - i * n + n;

        start_ind = end_ind - n + 1;
        start_ind = max([1, start_ind]);

        temp = in(start_ind:end_ind);

        out = [out, temp];
    end

end
```

Question 2.1

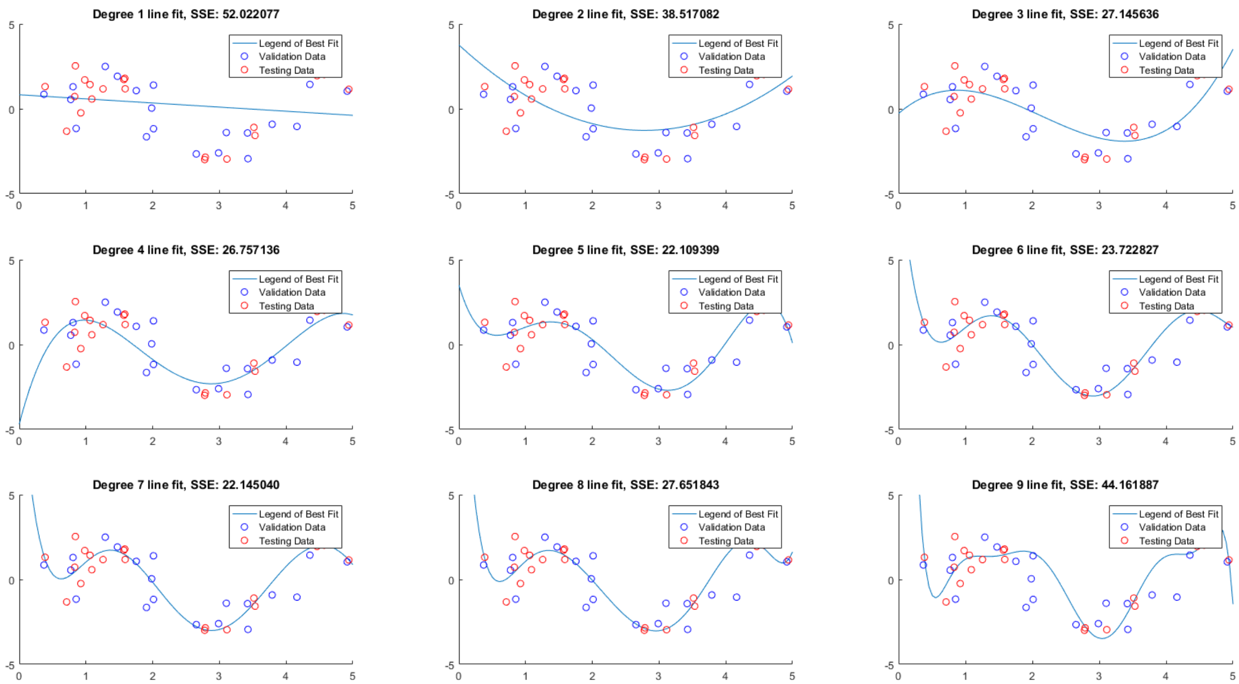

Figure 1: Lines of best fit

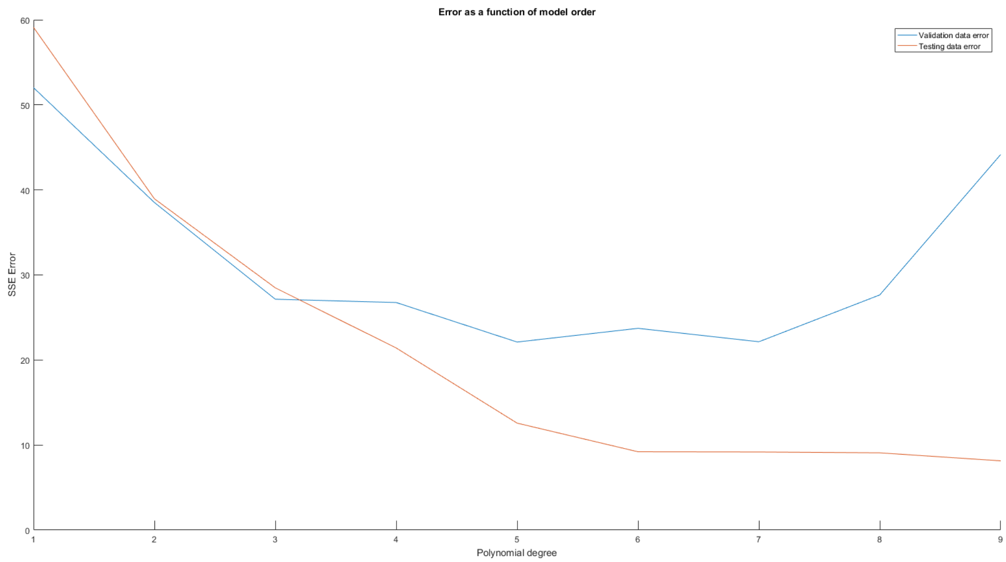

Figure 2: Error vs Polynomial Degree

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

```
function q4(data, class)
    class_names = unique(class);
    x_values = 1:length(class);

    class1 = zeros(1, length(class));
    class2 = zeros(1, length(class));

    for i = x_values
        if strcmp(class{i}, class_names{1})
            class1(i) = 1;
        else
            class2(i) = 1;
        end
    end

    % Verify the classes are correct
```

```
    % figure;
    % hold on;
    % scatter(1:length(class1), class1);
    % scatter(1:length(class2), class2);
    % hold off;

    estimate_range = 1:0.1:8;

    class1_data = data;
    class1_data(class2 == 1) = NaN;

    class1_mle = mle(class1_data);
    class1_pdf = normpdf(estimate_range, class1_mle(1), class1_mle
        (2));

    class2_data = data;
    class2_data(class1 == 1) = NaN;

    class2_mle = mle(class2_data);
    class2_pdf = normpdf(estimate_range, class2_mle(1), class2_mle
        (2));


    % figure;

    % scatter(x_values, data);

    % Verify the classes are divided
    figure;

    hold on;

    yyaxis left;
    scatter(class1_data, x_values, 'r');
    scatter(class2_data, x_values, 'b');

    yyaxis right;
    plot(estimate_range, class1_pdf, 'r');
    plot(estimate_range, class2_pdf, 'b');
    legend('Iris Setosa', 'Iris Versicolor');
    hold off;

    % Plot the likelihood
    figure;
    hold on;
```

```
    plot(estimate_range, class1_pdf);
    plot(estimate_range, class2_pdf);
    xlim([1, 10]);

    title('Likelihoods');
    xlabel('x');
    ylabel('P(x|C_i)');

    p_class1 = class1_pdf ./ (class1_pdf + class2_pdf);
    p_class2 = class2_pdf ./ (class1_pdf + class2_pdf);

    figure;
    hold on

    plot(estimate_range, p_class1);
    plot(estimate_range, p_class2);

    title('Posteriors');
    xlabel('x');
    ylabel('P(x|C_i)');

    hold off;

end
```

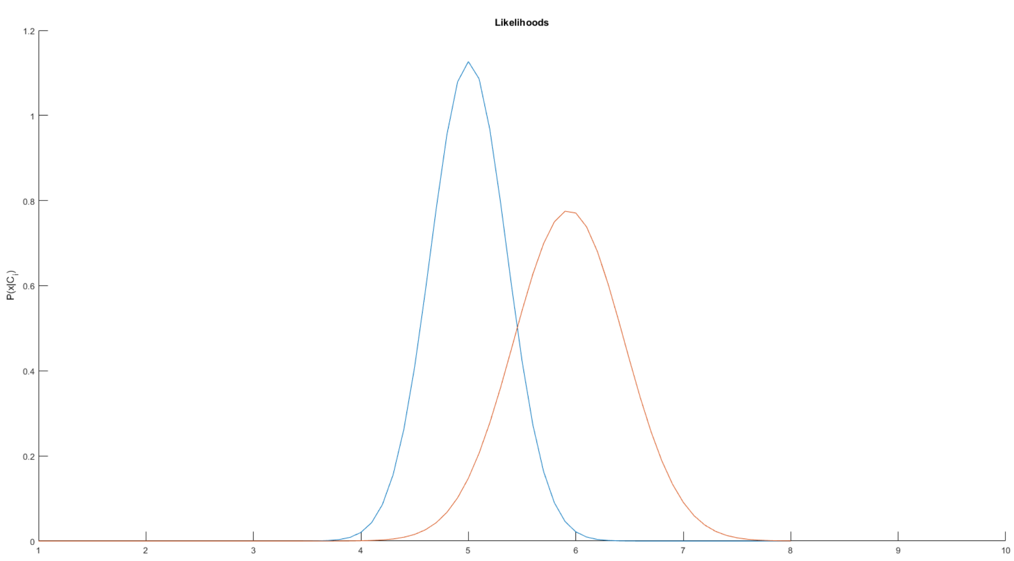

Figure 3: Likelihoods

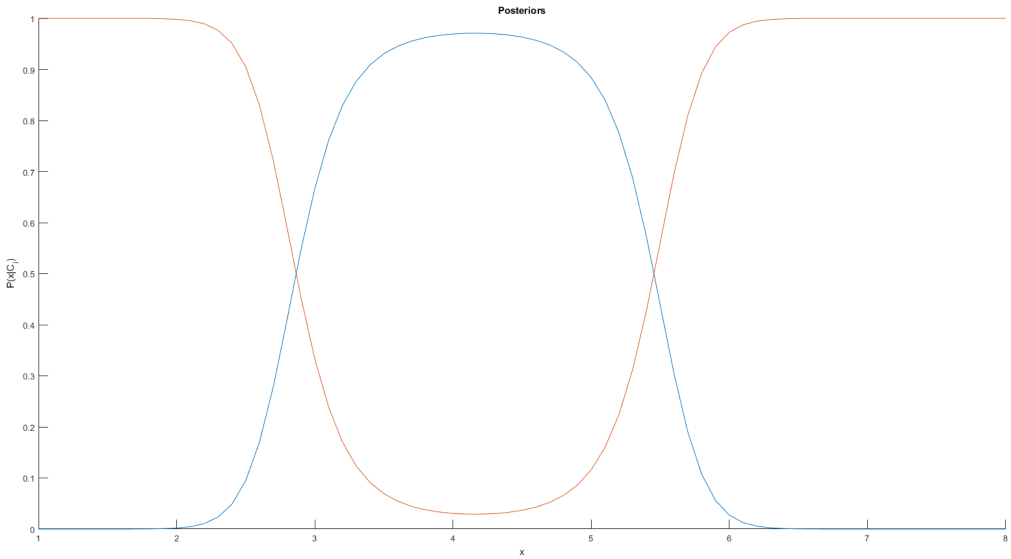

Figure 4: Posteriors

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.

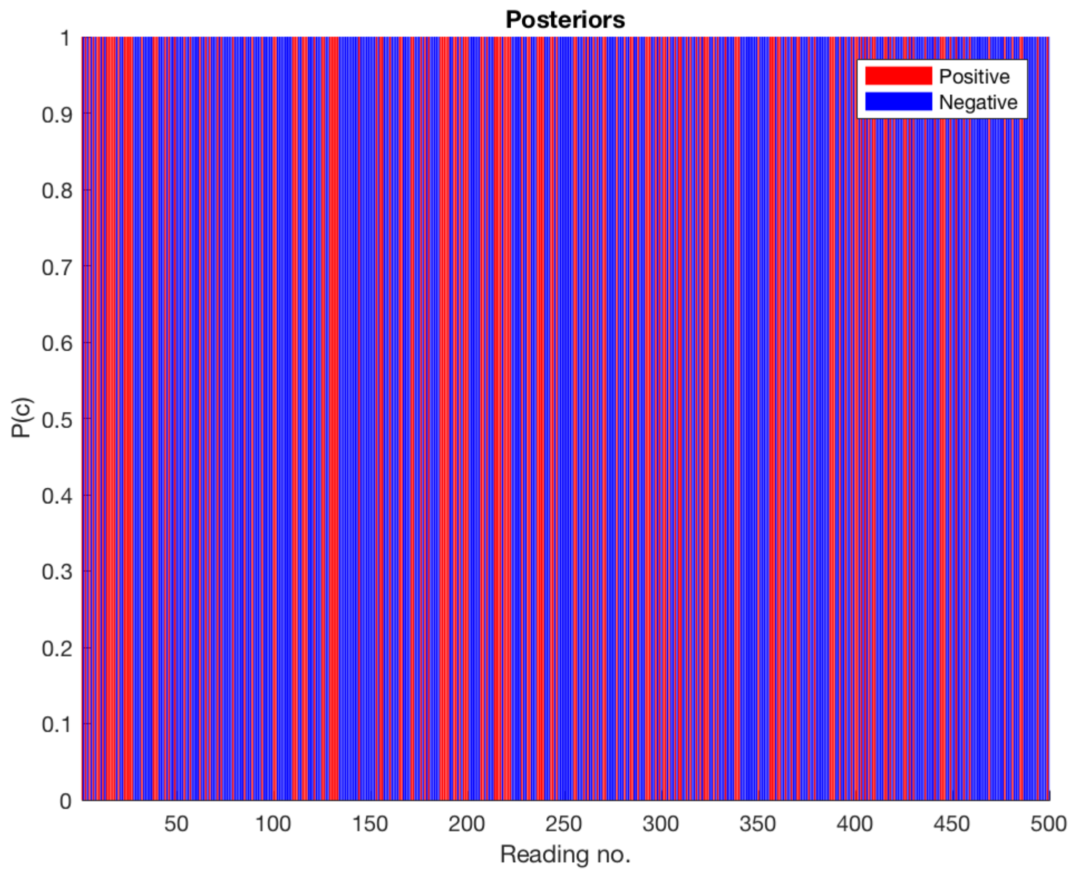

Figure 5: Posteriors

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:

```
const =

    14.5107


linear =

    -0.0230
```

```
    0.0030
    0.0465
    0.0089
   -0.0099
   -0.3704
    1.7751
   -0.2770


quadratic =

  Columns 1 through 6

   -0.0388   -0.0004    0.0002   -0.0014    0.0002    0.0012
   -0.0004   -0.0003    0.0001   -0.0001    0.0001    0.0000
    0.0002    0.0001   -0.0008   -0.0003    0.0000    0.0015
   -0.0014   -0.0001   -0.0003   -0.0004   -0.0000    0.0011
    0.0002    0.0001    0.0000   -0.0000   -0.0000    0.0002
    0.0012    0.0000    0.0015    0.0011    0.0002   -0.0019
   -0.0717   -0.0056    0.0040   -0.0262    0.0064   -0.0130
    0.0043    0.0005   -0.0004    0.0008   -0.0003    0.0016

  Columns 7 through 8

   -0.0717    0.0043
   -0.0056    0.0005
    0.0040   -0.0004
   -0.0262    0.0008
    0.0064   -0.0003
   -0.0130    0.0016
   -2.5106    0.0301
    0.0301    0.0005
```

Question 3.2

LDA can be implemented similiar to the QDA done above.

```
%
% Q2
% Linear Discriminant Analysis
%

% General Steps
```

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

%% Step 0: Load the data
raw = readtable('pima_indians_diabetes.csv');

raw_data = raw(:, 1:8);

% The classfied classes, e.g. positive or negative
actual = table2array(raw(:, 9));
training_actual = actual(1:500);
testing_actual = actual(500:end);

% Convert the table into an array
data = table2array(raw_data);

% Split into training and testing
training = data(1:500, :);
testing = data(500:end, :);

% Divide training data into positive and negative classes

% An array of data
training_positive = NaN(500, 8);
training_negative = NaN(500, 8);

% Binary array of samples
positive = zeros(1, 500);
negative = zeros(1, 500);
for i = 1:500
    if strcmp(training_actual(i), 'pos')
        positive(i) = 1;
        training_positive(i, :) = training(i, :);
    else
        negative(i) = 1;
        training_negative(i, :) = training(i, :);
    end
end

```

```
pos_testing =  zeros(1, length(testing_actual));
neg_testing =  zeros(1, length(testing_actual));

% Parse the testing data
for i = 1:length(testing_actual)
    if strcmp(testing_actual(i), 'pos')
        pos_testing(i) = 1;
    else
        neg_testing(i) = 1;
    end

end

%% Step 1: Calculate the mean vectors

% Class 1 mean vector
pos_mean_vector = nanmean(training_positive);

% Class 2 mean vector
neg_mean_vector = nanmean(training_negative);

%% Step 2: Compute scatter matrices
% These will be 8x8 matrices for each class

positive_scatter_matrix = zeros(8, 8);
for i = 1:length(training_positive)
    mv = transpose(pos_mean_vector);
    row = transpose(training_positive(i, :));

    if not(any(isnan(row)))
        positive_scatter_matrix = positive_scatter_matrix + (row - mv) * (transpose(row - mv));
    end
end

within_class_scatter_matrix = positive_scatter_matrix;

negative_scatter_matrix = zeros(8, 8);
for i = 1:length(training_negative)
    mv = transpose(neg_mean_vector);
    row = transpose(training_negative(i, :));

    if not(any(isnan(row)))
```

```
            negative_scatter_matrix = negative_scatter_matrix + (row - mv) * (transpose(row - mv));
        end
end

within_class_scatter_matrix = within_class_scatter_matrix + negative_scatter_matrix;

% Calculate the between class scatter matrix
between_class_scatter_matrix = zeros(8, 8);

overall_mean = transpose(mean(training));

% Do it for positive first
mv = transpose(pos_mean_vector);

n = length(find(all(~isnan(training_positive), 2)));

temp = n * (mv - overall_mean) * transpose(mv - overall_mean);

between_class_scatter_matrix = between_class_scatter_matrix + temp;

% Do it for negative
mv = transpose(neg_mean_vector);

n = length(find(all(~isnan(training_negative), 2)));

temp = n * (mv - overall_mean) * transpose(mv - overall_mean);

between_class_scatter_matrix = between_class_scatter_matrix + temp;

%% Step 3: Calculate the eigenvectors and eigenvalues

e = eig(inv(within_class_scatter_matrix) * between_class_scatter_matrix);
[V, D] = eig(inv(within_class_scatter_matrix) * between_class_scatter_matrix);

%% Step 4: Sort the eigenvectors and eigenvalues by decreasing order
[a, b] = sort(e, 'descend');

best_eig_value = a(1);
```

```
best_eig_vector = V(b(1), :);

second_best_eig_value = a(2);
second_best_eig_vector = V(b(2), :);

W = zeros(8, 2);

W(:, 1) = transpose(best_eig_vector);
W(:, 2) = transpose(second_best_eig_vector);

% The W matrix

real(W)

% Multiply the training data with W to classify
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

```
const =

    7.4066


linear =

   -0.1251
   -0.0319
    0.0114
    0.0041
    0.0008
   -0.0835
   -0.8727
   -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.

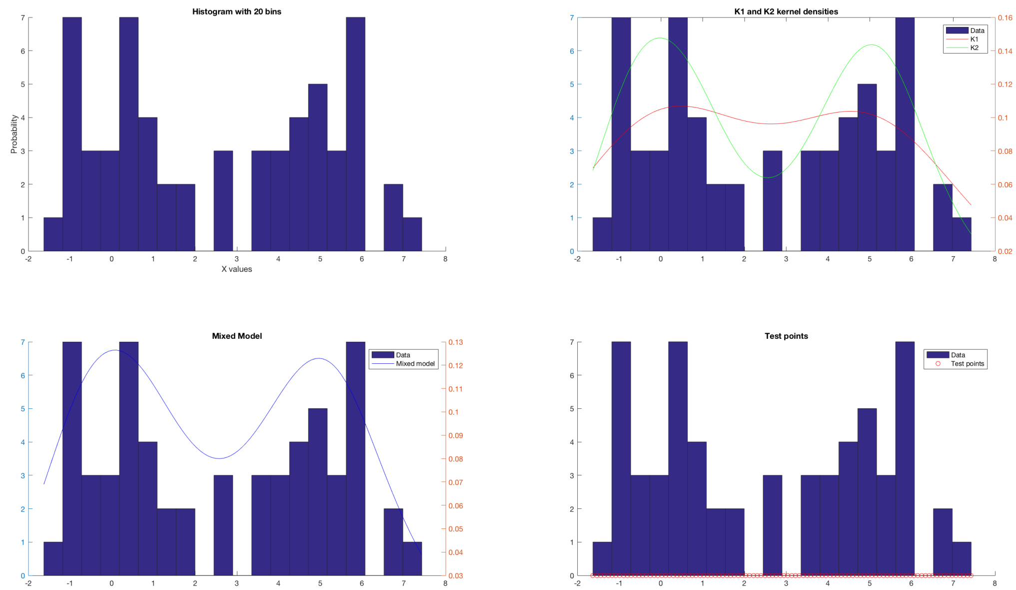

Figure 6: Q5

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

```
% Q5

x = [randn(30, 1); 5 + randn(30, 1)];

subplot(2, 2, 1);
hold on;
hist(x, 20);
title('Histogram with 20 bins');
xlabel('X values');
ylabel('Probability');
hold off;

% Demo the results
```

```
subplot(2, 2, 2);
yyaxis left;
hist(x, 20);
yyaxis right;

% Generate 100 random datapoints between that covers the range of
    data in x
x_points = min(x):((max(x)-min(x))/99):max(x);

% Generate the histogram bin counts
[dist, edges] = histcounts(x, 20);
N = sum(dist);
x0 = edges(1);
h = edges(2) - edges(1);

data = zeros(1, 100);

% Histogram estimator
for i = 1:100
    bin = ceil((x_points(i) - x0) / h);
    x_in_bin = dist(bin);

    data(i) = x_in_bin / (N * h);
end


% Create the kernel density estimators

% Default kernel width
[K1, k1x, bw1] = ksdensity(x, x_points);

% Half of the default kernel width
[K2, k2x, bw2] = ksdensity(x, x_points, 'width', bw1/2);

hold on;
plot(k1x, K1, 'r');
plot(k2x, K2, 'g');

title('K1 and K2 kernel densities');
legend('Data', 'K1', 'K2');
hold off;

% Calculate the Guassian mixed model
mixed = K1/2 + K2/2;

```

```
subplot(2, 2, 3);
hold on;
yyaxis left;
hist(x, 20);
yyaxis right;
plot (k1x, mixed, 'b');

title('Mixed Model');
legend('Data', 'Mixed model');
hold off;

subplot(2, 2, 4);
hold on;
hist(x, 20);
scatter(x_points, zeros(1, 100), 'r');

title('Test points');
legend('Data', 'Test points');
hold off;

% Do the calculations
KL(mixed, data)
KL(mixed, K1)
KL(mixed, K2)

%
% Calculate the KL Divergence
%
function result = KL(p, q)
    result = zeros(size(p));

    valid = p > 0 & q > 0;

    result1 = sum(p(valid) .* log(p(valid) ./ q(valid)));
    result2 = sum(q(valid) .* log(q(valid) ./ p(valid)));

    result(valid) = result1 + result2;

    result = mode(result);
end
```