CAB420 - Machine Learning Assignment 1

Table of Contents

Name: Shaun Sewell

Student Number: N9509623

Theory

Logistic regression is a method of fitting a probabilistic classifier that gives soft linear thresh-olds. It is common to use logistic regression with an objective function consisting of the negative log probability of the data plus an L2 regularizer:

$$L(\mathbf{w}) = -\sum_{i=1}^{N} Log\left(\frac{1}{1+e^{y_i(\mathbf{w}^T\mathbf{x}_i+b)}}\right) + \lambda\|\mathbf{w}\|_2^2$$

(a) Find the partial derivatives $\frac{\partial L}{\partial w_j}$

$$u = 1 + e^{y_i(\mathbf{w}^T\mathbf{x}_i+b)}$$

$$L(\mathbf{w}) = \sum_{i=1}^{N} log(u) + \lambda\|\mathbf{w}\|_2^2$$

$$\frac{\partial L}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} \frac{\partial}{\partial u} log(u) \frac{\partial u}{\partial \mathbf{w}_j} + \frac{\partial}{\partial \mathbf{w}_j}\lambda\|\mathbf{w}\|_2^2$$

$$\frac{\partial}{\partial \mathbf{w}_j}\lambda\|\mathbf{w}\|_2^2 = 2\lambda\mathbf{w}_j$$

$$\frac{\partial}{\partial u} log(u) = \frac{1}{u}$$

$$\frac{\partial u}{\partial \mathbf{w}_j} = y_i \mathbf{x}_{ij} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}$$

$$\frac{\partial L}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} \frac{y_i \mathbf{x}_{ij} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}} + 2\lambda \mathbf{w}_j$$

(b) Find the partial second derivatives $\frac{\partial L^2}{\partial w_j \partial w_k}$

$$\frac{\partial^2 L}{\partial \mathbf{w}_k \partial \mathbf{w}_j} = \sum_{i=1}^{N} \frac{\partial}{\partial \mathbf{w}_k} \frac{y_i \mathbf{x}_{ij} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}} + \frac{\partial}{\partial \mathbf{w}_k} 2\lambda \mathbf{w}_j$$

$$\text{When } k \neq j; \frac{\partial}{\partial \mathbf{w}_k} 2\lambda \mathbf{w}_j = 0$$

$$\text{When } k = j; \frac{\partial}{\partial \mathbf{w}_k} 2\lambda \mathbf{w}_j = 2\lambda$$

$$\frac{\partial}{\partial \mathbf{w}_k} \frac{y_i \mathbf{x}_{ij} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}} = \frac{\partial}{\partial \mathbf{w}_k} \frac{g}{h}$$

$$g = y_i \mathbf{x}_{ij} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}$$

$$h = 1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}$$

$$\frac{\partial}{\partial \mathbf{w}_j} \frac{g}{h} = \frac{\frac{\partial g}{\partial \mathbf{w}_j} h - g \frac{\partial h}{\partial \mathbf{w}_j}}{h^2}$$

$$\frac{\partial}{\partial \mathbf{w}_k} \frac{g}{h} = \frac{y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)} + y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{2y_i(\mathbf{w}^T \mathbf{x}_i + b)} - y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{2y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{\left(1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}\right)^2}$$

$$\frac{\partial}{\partial \mathbf{w}_k} \frac{g}{h} = \frac{y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{\left(1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}\right)^2}$$

$$\text{When } k \neq j; \frac{\partial^2 L}{\partial \mathbf{w}_k \partial \mathbf{w}_j} = \sum_{i=1}^{N} \frac{y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{\left(1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}\right)^2}$$

$$\text{When } k = j; \frac{\partial^2 L}{\partial \mathbf{w}_k \partial \mathbf{w}_j} = \sum_{i=1}^{N} \frac{y_i^2 \mathbf{x}_{ij} \mathbf{x}_{ik} e^{y_i(\mathbf{w}^T \mathbf{x}_i + b)}}{\left(1 + e^{y_i\left(\mathbf{w}^T \mathbf{x}_i + b\right)}\right)^2} + 2\lambda$$

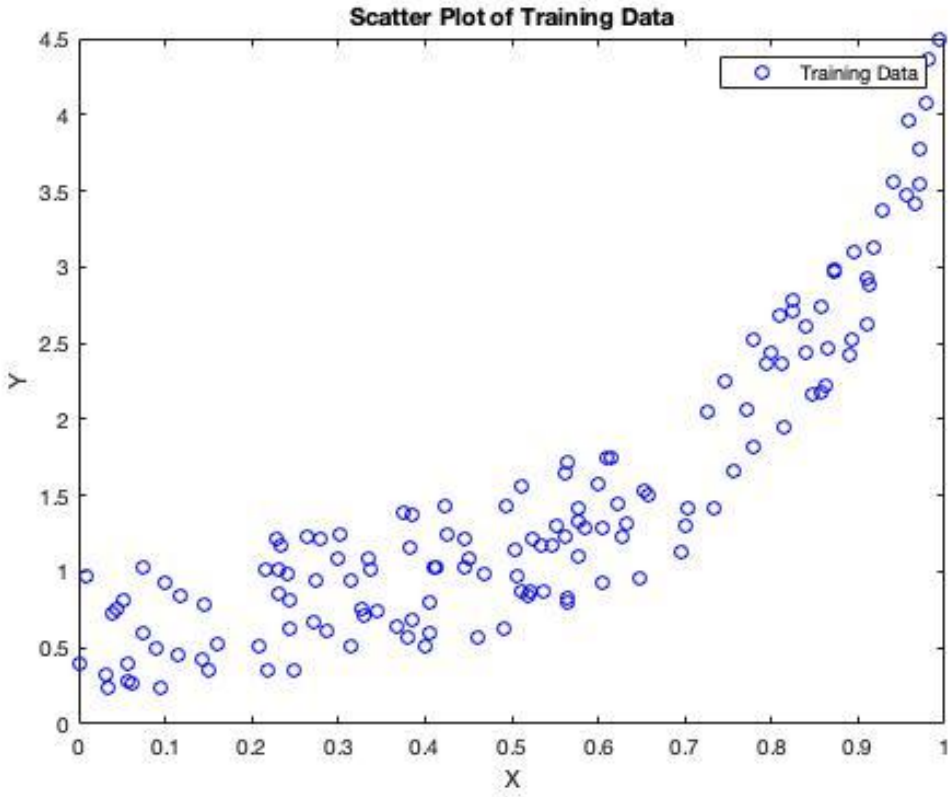
(c) From these results, show that L (w) is a convex function.

Practice

1. Feature, Classes and Linear Regression

(a) Plot the training data in a scatter plot.

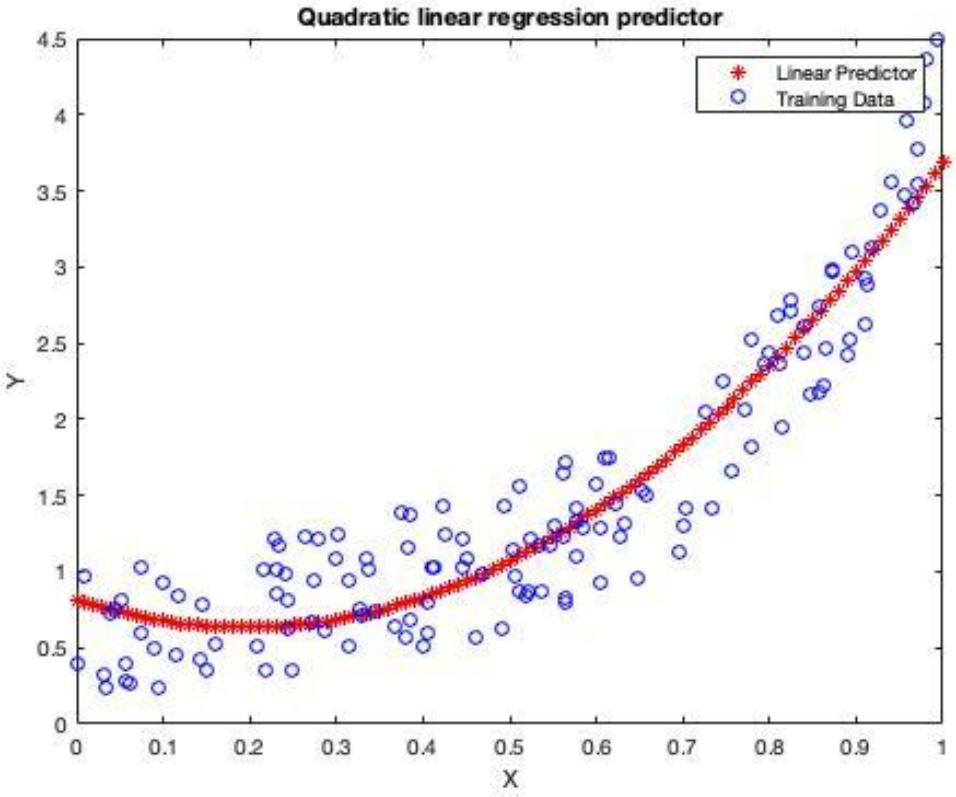
```
clear ; close all; clc
% Load training data and separate features
mTrain = load('data/mTrainData.txt');
Xtr = mTrain(: ,1); Ytr = mTrain(: ,2);

% Plot training data
figure('name', 'Scatter Plot of Training Data');
plot (Xtr, Ytr, 'bo');
xlabel('X');
ylabel('Y');
legend('Training Data');
title('Scatter Plot of Training Data');
```



(b) Create a linear regression learner using the above functions. Plot it on the same plot as the training data.

```
Xtr_2 =  polyx(Xtr,2);
% train a regression learner
regress_learner = linearReg(Xtr_2 ,Ytr);
```

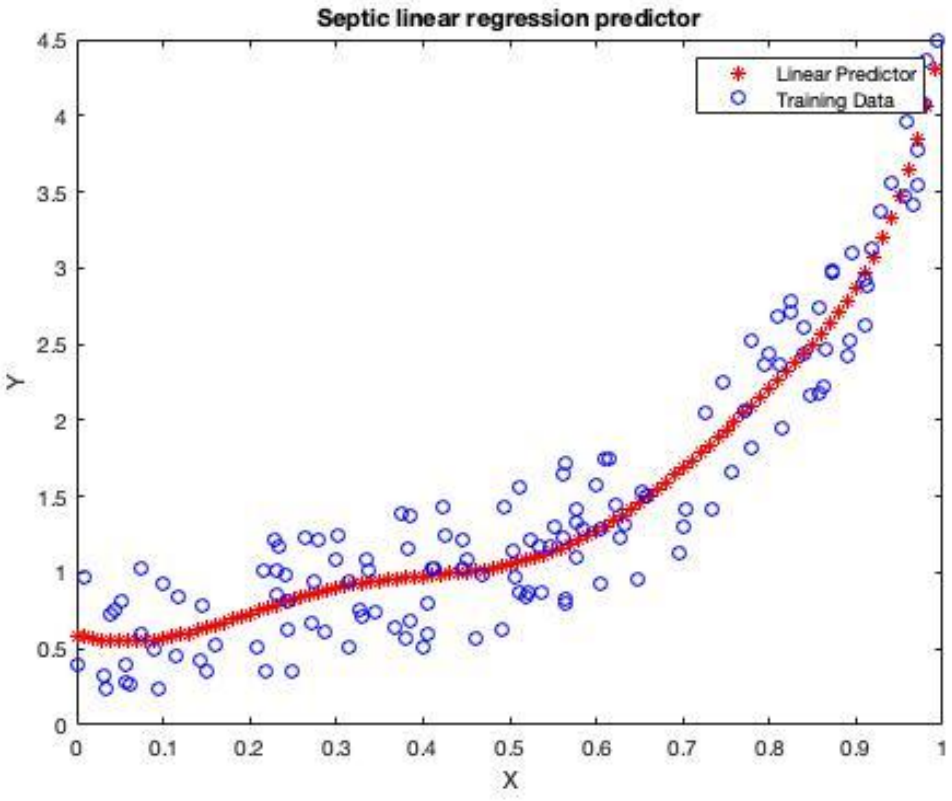
```
xline = [0:.01:2]' ; % transpose : make a column vector , like
 training x
yline = predict( regress_learner , polyx (xline ,2) );

% Plot predictions
figure('name', 'Quadratic linear regression predictor');
plot(xline, yline, 'r*');
% Plot training data
hold on
plot (Xtr, Ytr, 'bo');

% Set figure properties
xlim([0 1]);
ylim([0 4.5]);
xlabel('X');
ylabel('Y');
legend('Linear Predictor', 'Training Data');
title('Quadratic linear regression predictor');
```



(c) Create plots with the data and a higher-order polynomial (3, 5, 7, 9, 11, 13).

```
Xtr_7 =  polyx(Xtr,7);
% train the learner
septic_learner = linearReg(Xtr_7 ,Ytr);
xline = [0:.01:2]' ;                                % transpose : make
 a column vector , like training x
yline = predict( septic_learner , polyx (xline ,7) );
```

```
% Plot predictions
figure('name', "Septic linear regression predictor");
plot(xline, yline, 'r*');
hold on
% Plot training data
plot (Xtr, Ytr, 'bo');
% Set figure properties
xlim([0 1]);
ylim([0 4.5]);
xlabel('X');
ylabel('Y');
legend('Linear Predictor', 'Training Data');
title('Septic linear regression predictor');
```



(d) Calculate the mean squared error (MSE) associated with each of your learned models on the training data.

```
% Quadratic model
MSE_Quadratic_Trained = mse(regress_learner,Xtr_2, Ytr);
fprintf('The MSE for the quadratic predictor on training data was:
 %.2f\n', MSE_Quadratic_Trained);

% Septic model
MSE_Septic_Trained = mse(septic_learner,Xtr_7, Ytr);
fprintf('The MSE for the septic predictor on training data was: %.2f
\n', MSE_Septic_Trained);
```

```
The MSE for the quadratic predictor on training data was: 0.11
The MSE for the septic predictor on training data was: 0.08
```

(e) Calculate the MSE for each model on the test data (in ?mTestData.txt?).

```
mTest = load('data/mTestData.txt');
Y_Test = mTest(: ,1); X_Test = mTest(: ,2);

% Quadratic model
X_Test_2 = polyx(X_Test,2);
MSE_Quadratic_Test = mse(regress_learner,X_Test_2, Y_Test);
fprintf('The MSE for the quadratic predictor on test data was: %.2f
\n', MSE_Quadratic_Test);

% Septic model
X_Test_7 = polyx(X_Test,7);
MSE_Septic_Test = mse(septic_learner,X_Test_7, Y_Test);
fprintf('The MSE for the septic predictor on test data was: %.2f\n',
 MSE_Septic_Test);
```

```
The MSE for the quadratic predictor on test data was: 376.91
The MSE for the septic predictor on test data was: 109031782837.51
```

(f) Calculate the MAE for each model on the test data. Compare the obtained MAE values with the MSE values obtained in above (e).

```
X_Test_2 = polyx(X_Test,2);
MAE_Quadratic_Test = mae(regress_learner,X_Test_2, Y_Test);
fprintf('The MAE for the quadratic predictor on test data was: %.2f
\n', MAE_Quadratic_Test);

% Septic model
X_Test_7 = polyx(X_Test,7);
MAE_Septic_Test = mae(septic_learner,X_Test_7, Y_Test);
fprintf('The MAE for the septic predictor on test data was: %.2f\n',
 MAE_Septic_Test);
```

```
The MAE for the quadratic predictor on test data was: 12.05
The MAE for the septic predictor on test data was: 100833.43
```

The MAE of the models is significatnly smaller than the MSE. This highlights the penalty that the MSE creates for errors that are a significant distance from the mean.

2. kNN Regression

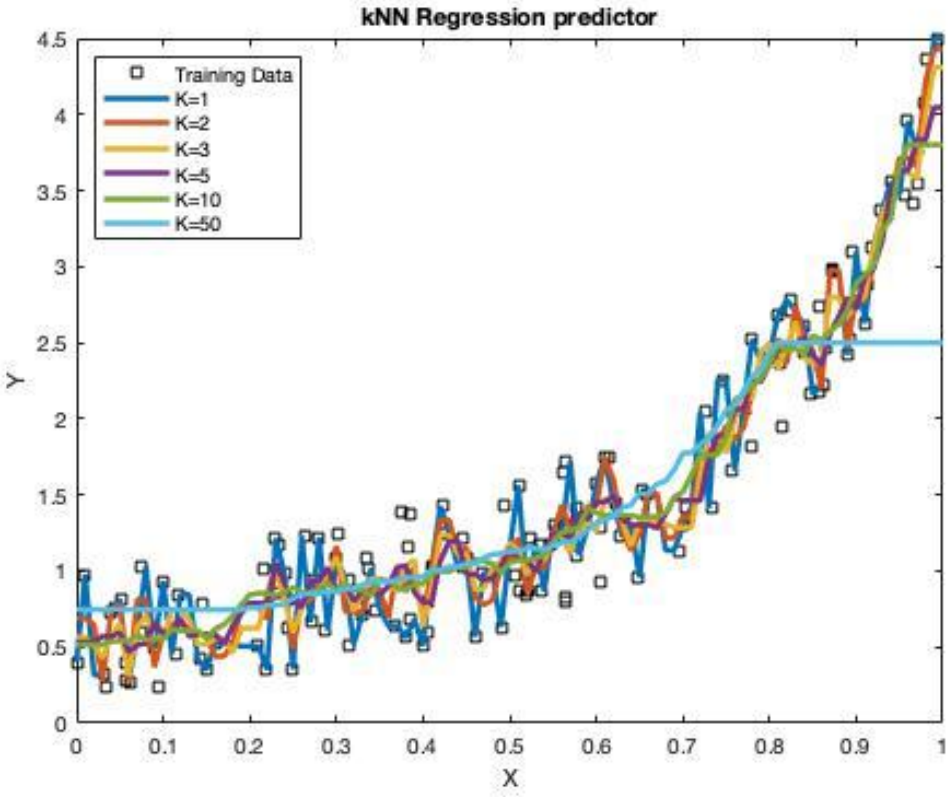
(a) Using the knnRegress class, implement (add code to) the predict function to make it functional.

```
% Test function: predict on Xtest
function Yte = predict(obj,Xte)
    % get size of training, test data
    [Ntr,Mtr] = size(obj.Xtrain);
    [Nte,Mte] = size(Xte);
```

```
    % figure out how many classes & their labels
    classes = unique(obj.Ytrain);

    % make Ytest the same data type as Ytrain
    Yte = repmat(obj.Ytrain(1), [Nte,1]);

    K = min(obj.K, Ntr);  % can't have more than Ntrain neighbors

    for i=1:Nte
        % compute sum of squared differences
        dist = sum( bsxfun( @minus, obj.Xtrain, Xte(i,:) ).^2 , 2);

        % find nearest neighbors over Xtrain (dimension 2)
        [tmp,idx] = sort(dist);

        % idx(1) is the index of the nearest point, etc.

        % predict ith test example's value from nearest neighbors
        Yte(i)=mean(obj.Ytrain(idx(1:K)));
    end
end
```

(b) Using the same technique as in Problem 1a, plot the predicted function for several values of k : 1, 2, 3, 5, 10, 50. How does the choice of k relate to the "complexity" of the regression function?

```
K_Values = [1, 2, 3, 5, 10, 50];

% Plot training data
figure('name', 'kNN Regression predictor');
plot (Xtr, Ytr, 'ks');
title('kNN Regression predictor')
legend('Training Data','location','Northwest');
xlim([0 1]);
ylim([0 4.5]);
xlabel('X');
ylabel('Y');
hold on

xline = [0:.01:2]' ; % transpose : make a column vector , like
 training x

% Create a kNN regression predictor from the data Xtr, Ytr for each K.
for i=K_Values
    learner = knnRegress(i, Xtr, Ytr);

    % Plot the current models predictions
    yline = predict(learner, xline);
    plot(xline, yline, '', 'DisplayName', strcat('K=',
 num2str(i)),'LineWidth',2.5);
end
```



As the value of K increases the complexity of the function appears to decrease. This is because as the number of neigbours used increases the effect an individual can have on the functions output is decreased.

(c) We discussed in class that the k-nearest-neighbor classifier?s decision boundary can be shown to be piecewise linear. What kind of functions can be output by a nearest neighbor regression function? Briefly justify your conclusion.

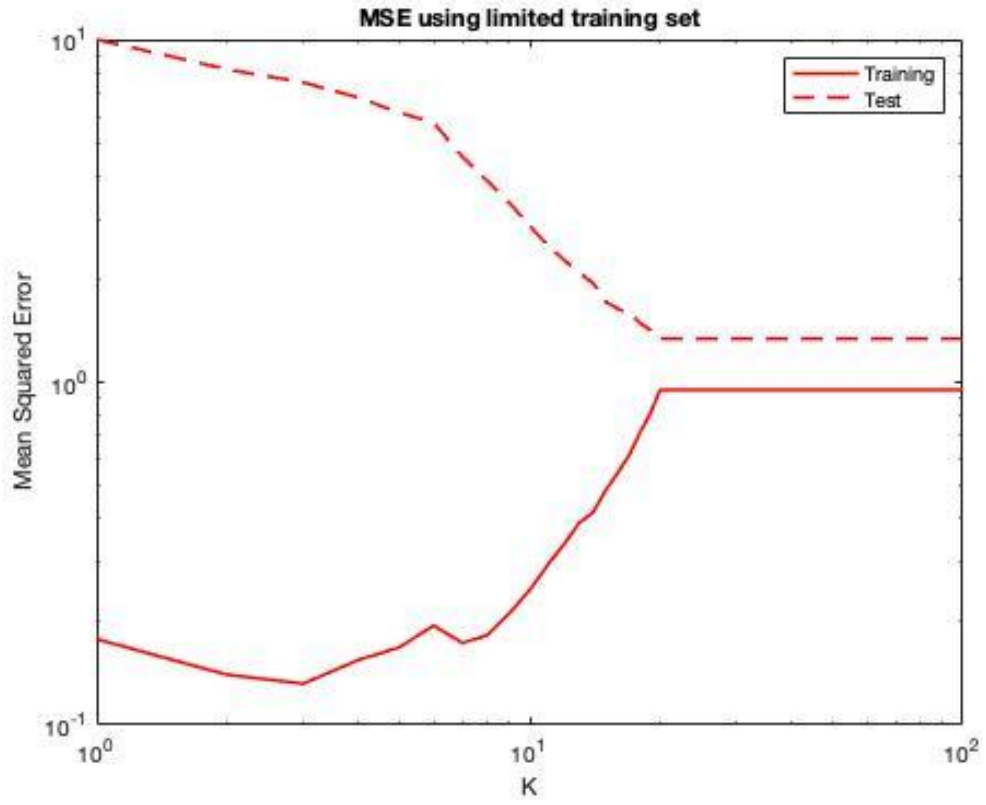
Given that the K Nearest Neighbors function produces a piecewise linear decision boundary any function that can be represented by a set of linear equations can be output.

3. Hold-out and Cross-validation

(a) Similarly to Problem 1 and 2, compute the MSE of the test data on a model trained on only the first 20 training data examples for k = 1, 2, 3, . . . , 100. Plot both train and test MSE versus k on a log-log scale (see help loglog?). Assign title to your figure (ie. 20 data) and legends to your curves (ie. test, train). Discuss what you observed from the figure.

```
X_Training_20 = Xtr(1:20, :); Y_Training_20 = Ytr(1:20, :);
MSE_Training_20 = zeros(100,1);
MSE_Test_20 = zeros(100,1);

for k=1:100
    learner_20 = knnRegress(k, X_Training_20, Y_Training_20);
    MSE_Training_20(k, 1) = mse(learner_20,Xtr, Ytr);
    % Store the MSE of the predictions
    MSE_Test_20(k, 1) = mse(learner_20,X_Test, Y_Test);
end
```

```
figure('name', 'MSE Comparison 20');
loglog(1:100, MSE_Training_20(:, 1),'r', 'LineWidth', 1.5);
hold on;
loglog(1:100, MSE_Test_20(:, 1),'--r', 'LineWidth', 1.5);
title('MSE using limited training set');
xlabel('K');
ylabel('Mean Squared Error');
legend('Training', 'Test');
hold off;
```

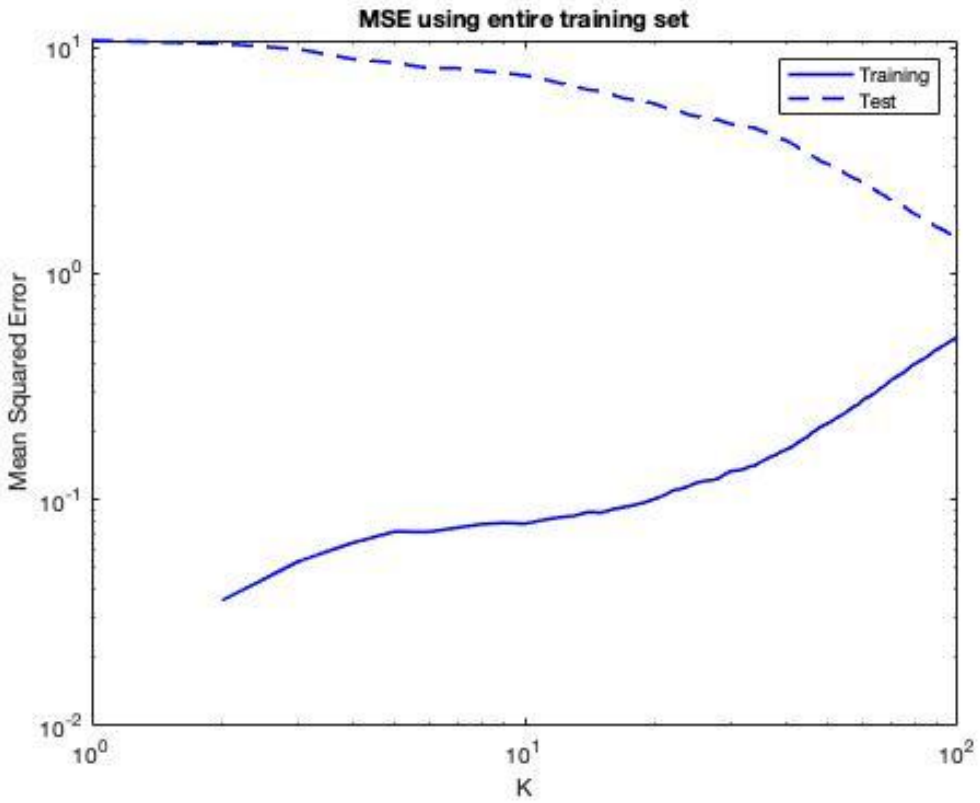


As the K value increases towards 20 the MSE of the Testing set decreases where as the Training set MSE rapidly increases. Once the K value reaches 20 the MSE of both sets plateaus because K cannot be larger than the training set.

(b) Repeat, but use all the training data. What happened? Contrast with your results from Problem 1 (hint: which direction is complexity in this picture?).

```
MSE_Training_All = zeros(100,1);
MSE_Test_All = zeros(100,1);

for k=1:100
    learner_All = knnRegress(k, Xtr, Ytr);
    MSE_Training_All(k, 1) = mse(learner_All,Xtr, Ytr);
    % Store the MSE of the predictions
    MSE_Test_All(k, 1) = mse(learner_All,X_Test, Y_Test);
```

```
end

figure('name', 'MSE Comparison');
loglog(1:100, MSE_Training_All(:, 1),'b', 'LineWidth', 1.5);
hold on;
loglog(1:100, MSE_Test_All(:, 1),'--b', 'LineWidth', 1.5);
title('MSE using entire training set');
xlabel('K');
ylabel('Mean Squared Error');
legend('Training', 'Test');
hold off;
```

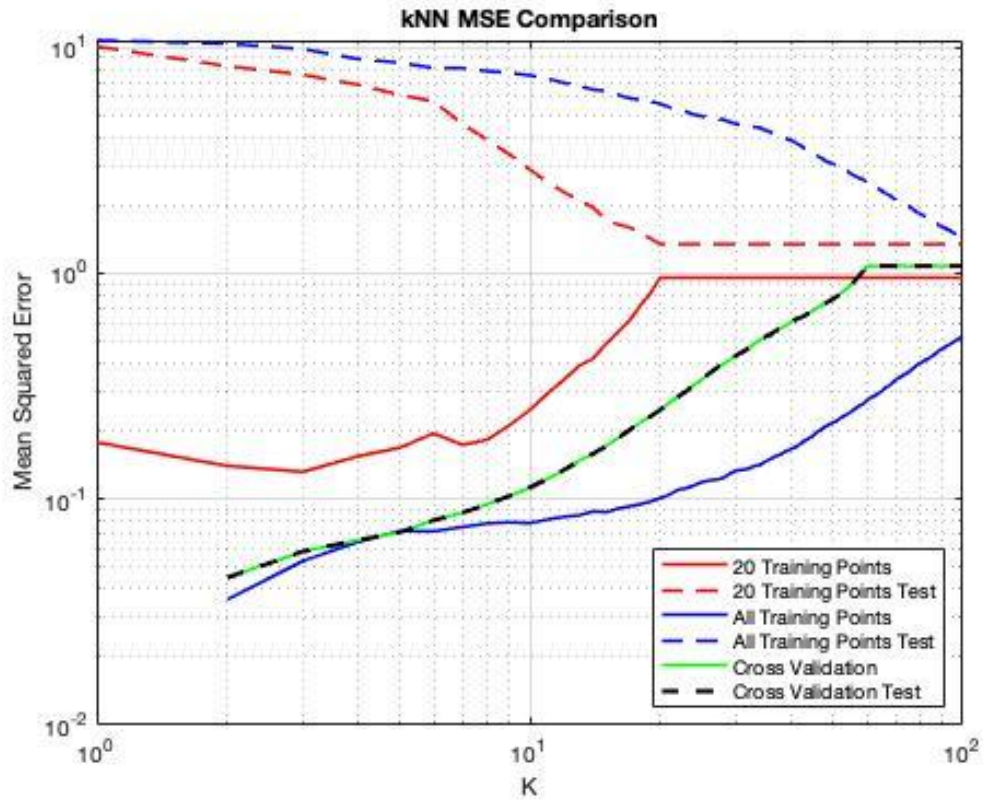


Unlike in (a) the MSE of both sets gradually trends towards 1. This is the result of not inavertently restricting the maximum k value allowed.

(c) Using ?only the training data,? estimate the curve using 4-fold cross-validation. Split the training data into two parts, indices 1:20 and 21:140; use the larger of the two as training data and the smaller as testing data, then repeat three more times with different sets of 20 and average the MSE. Plot this together with (a) and (b). Use different colors or marks to differentiate three scenarios. Discus why might we need to use this technique via comparing curves of three scenario?

```
MSE_Training_CV = zeros(100,1);
MSE_Test_CV = zeros(100, 1);

for k=1:100
    Cross_Val_Training_MSE = 0;
    Cross_Val_Testing_MSE = 0;
```

```
    for i=1:4

        % Find the index to split the training data set
        start_index = 20*(i - 1) + 1;       % 1:20,21:40,41:60,61:80
        end_index = start_index + 19;
        Test_Data_Range = start_index:end_index;

        % Split the training data
        Cross_Val_Testing_X = Xtr(Test_Data_Range);
        Cross_Val_Testing_Y = Ytr(Test_Data_Range);
        Cross_Val_Training_X = Xtr(setdiff(1:80, Test_Data_Range));
        Cross_Val_Training_Y = Ytr(setdiff(1:80, Test_Data_Range));

        % Create a learner using this data
        learner_Cross_Val = knnRegress(k, Cross_Val_Training_X,
 Cross_Val_Training_Y);

        % Calculate the mse
        training_mse = mse(learner_Cross_Val,Cross_Val_Training_X,
 Cross_Val_Training_Y);
        testing_mse = mse(learner_Cross_Val,Cross_Val_Testing_X,
 Cross_Val_Testing_Y);

        Cross_Val_Training_MSE = Cross_Val_Training_MSE +
 training_mse;
        Cross_Val_Testing_MSE = Cross_Val_Testing_MSE + training_mse;

    end

    MSE_Training_CV(k, 1) = Cross_Val_Training_MSE / 4.0;
    MSE_Test_CV(k, 1) = Cross_Val_Testing_MSE / 4.0;
end



% Plot training data
figure('name', 'kNN MSE Comparison');
loglog(1:100, MSE_Training_20(:, 1),'r', 'LineWidth', 1.5);
hold on;
loglog(1:100, MSE_Test_20(:, 1),'--r', 'LineWidth', 1.5);
loglog(1:100, MSE_Training_All(:, 1),'b', 'LineWidth', 1.5);
loglog(1:100, MSE_Test_All(:, 1),'--b', 'LineWidth', 1.5);
loglog(1:100, MSE_Training_CV(:, 1),'g', 'LineWidth', 1.5);
loglog(1:100, MSE_Test_CV(:, 1),'--k', 'LineWidth', 2);
grid on
title('kNN MSE Comparison');
xlabel('K');
ylabel('Mean Squared Error');
legend('20 Training Points', '20 Training Points Test', 'All Training
 Points', 'All Training Points Test', 'Cross Validation', 'Cross
 Validation Test', 'Location', 'southeast');
hold off;
```



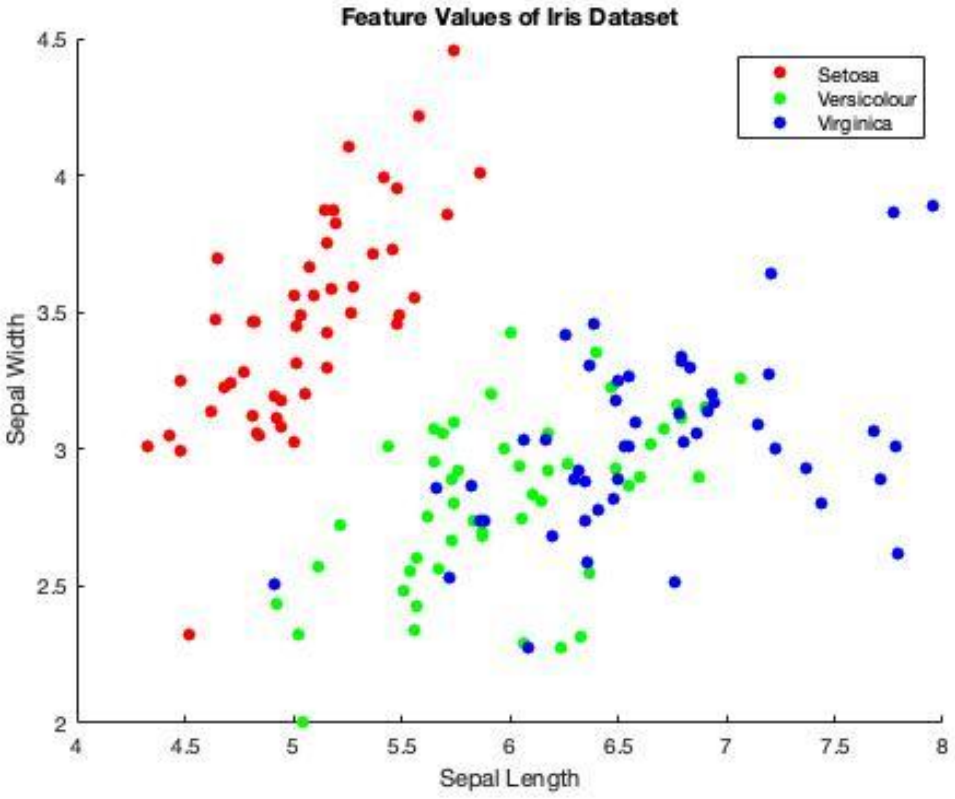
By using 4-fold cross validation the MSE variance from training data to testing data is minimised. This technique is needed given the inverted nature of the results from (a) and (b).

4. Nearest Neighbor Classifiers

(a) Plot the data by their feature values, using the class value to select the color.

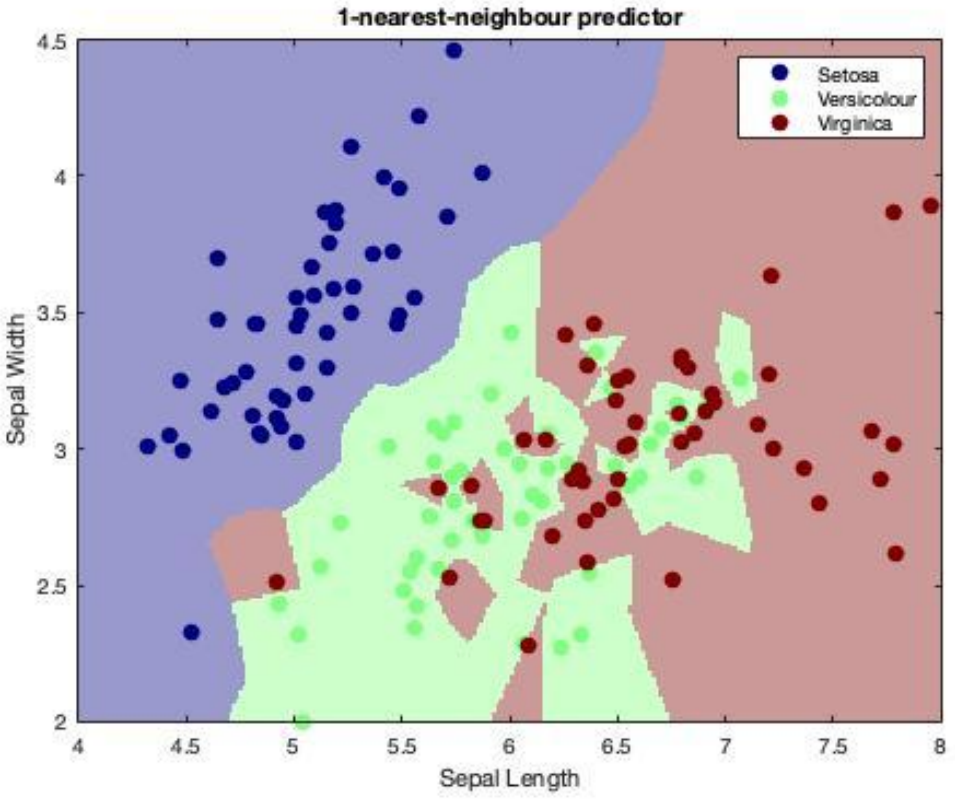
```
iris = load('data/iris.txt');
pi = randperm(size(iris, 1));
Y = iris(pi, 5); X = iris(pi, 1:2);

% Plot the feature values
figure('Name','Feature Values of Iris Dataset');
hold on;
colours = unique(Y);
for colour = 1:length(colours)

    RGB_Vector = [0, 0, 0];
    RGB_Vector(colour) = 1;
    feature_indicies = find(Y==colours(colour));
    feature_points = X(feature_indicies, :);
    feature_point_x = feature_points(1:end, 1);
    feature_point_y = feature_points(1:end, 2);
    scatter(feature_point_x, feature_point_y, [],
 RGB_Vector, 'filled');
end
```

```
title('Feature Values of Iris Dataset');
xlabel('Sepal Length');
ylabel('Sepal Width');
legend('Setosa', 'Versicolour', 'Virginica');
hold off;
```



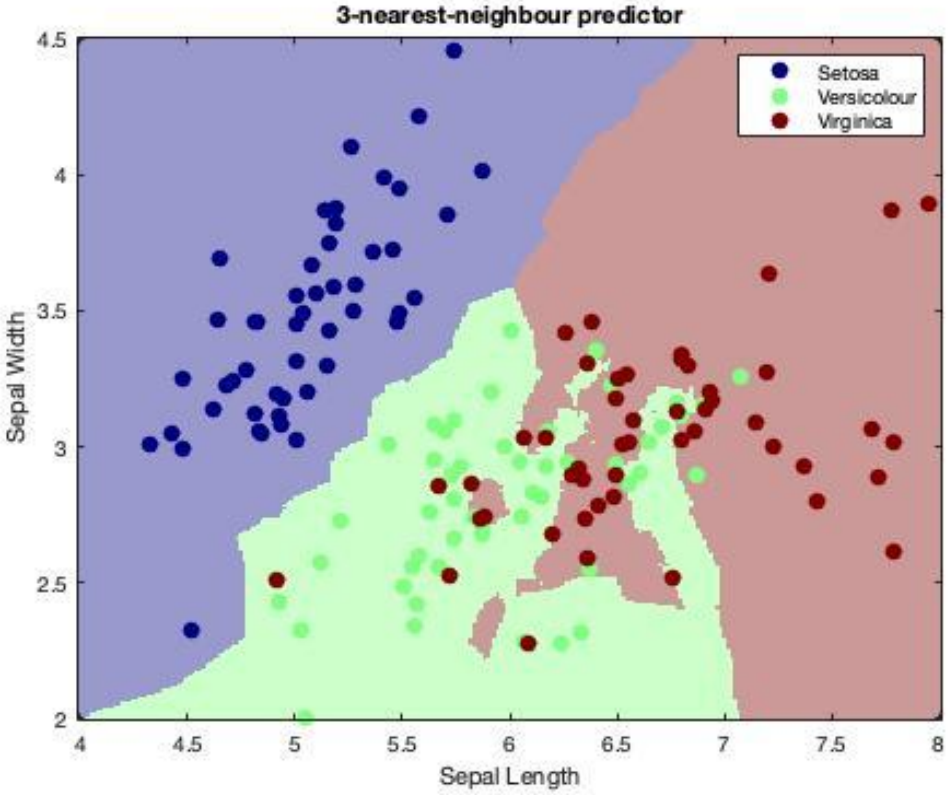
(b) Use the provided ?knnClassify class to learn a 1-nearest-neighbor predictor. Use the function class2DPlot(learner,X,Y)? to plot the decision regions and training data together.

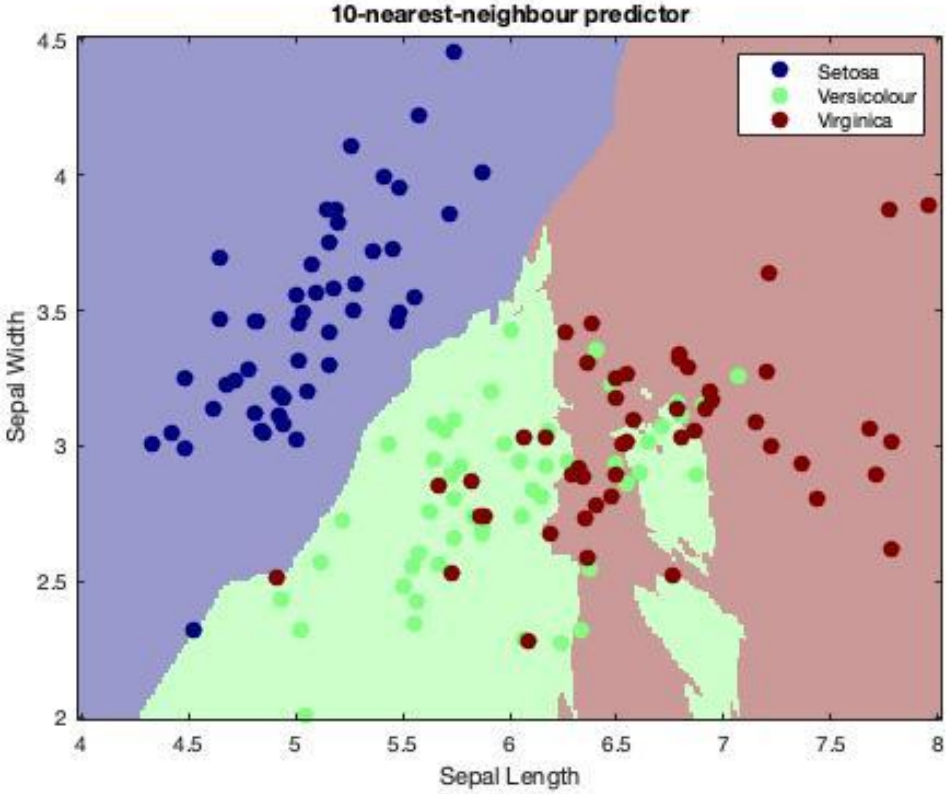
```
iris_knn_learner_1 = knnClassify(1, X, Y);
class2DPlot(iris_knn_learner_1,X,Y);
title('1-nearest-neighbour predictor');
xlabel('Sepal Length')
ylabel('Sepal Width')
legend('Setosa', 'Versicolour', 'Virginica');
```

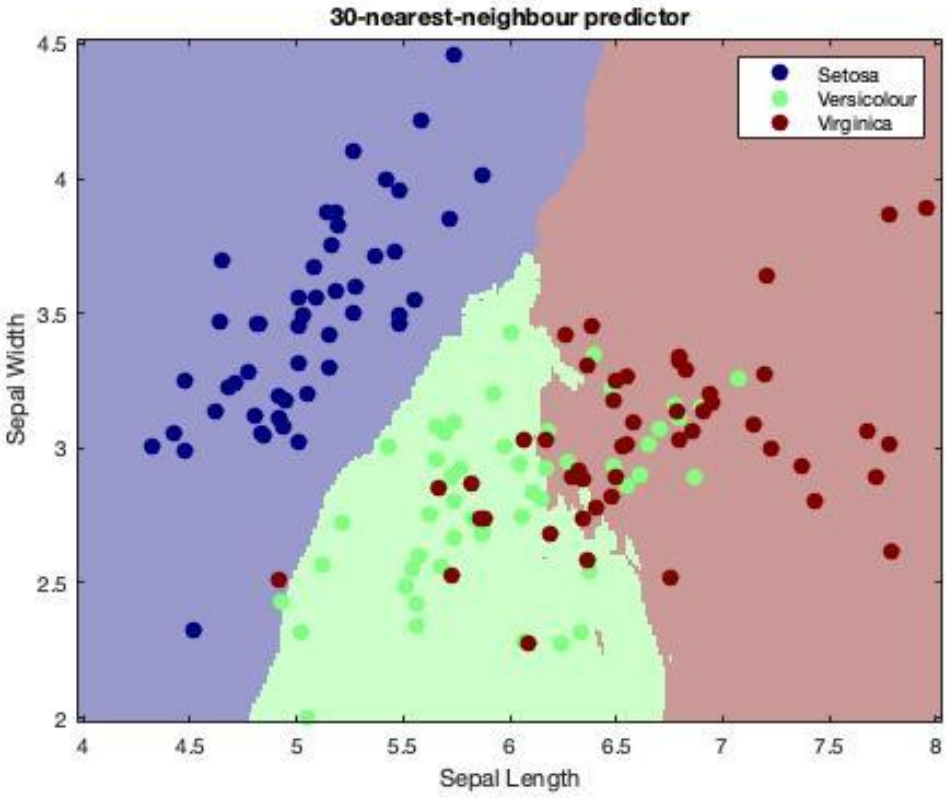


(c) Do the same thing for several values of k (say, [3, 10, 30]) and comment on their appearance.

```
iris_k_values_small = [3, 10, 30];

for k = iris_k_values_small
    iris_knn_learner = knnClassify(k, X, Y);
    class2DPlot(iris_knn_learner,X,Y);
    title(strcat(int2str(k), '-nearest-neighbour predictor'));
    xlabel('Sepal Length')
    ylabel('Sepal Width')
    legend('Setosa', 'Versicolour', 'Virginica');
end
```


3-nearest-neighbour predictor
Setosa
Versicolour
Virginica
Sepal Width
Sepal Length
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8


10-nearest-neighbour predictor
Setosa
Versicolour
Virginica
Sepal Width
Sepal Length
4.5
4
3.5
3
2.5
2
4
4.5
5
5.5
6
6.5
7
7.5
8

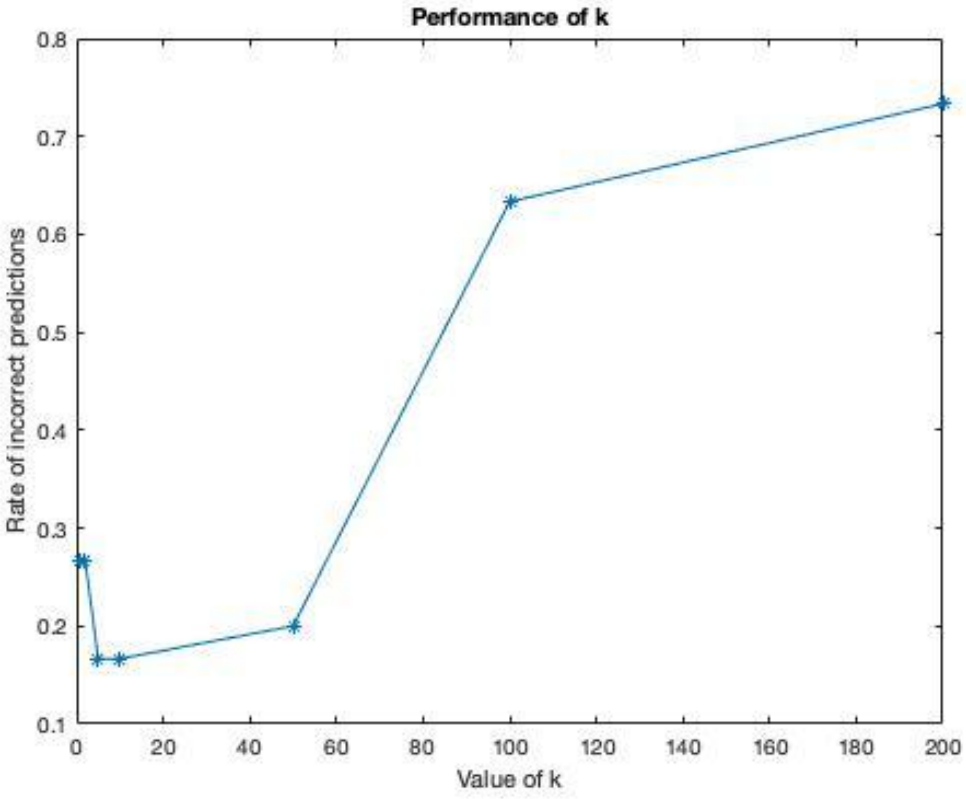


As the value of k increases the complexity of the decision boundary decreases and as such the number of correctly labeled points decreases. This shows an increasing level of underfitting on the data set. Conversely the lower K values show a high level of overfitting as can be seen in the K = 1 graph in which every value has been correctly labeled.

(d) Now split the data into an 80/20 training/validation split. For k = [1, 2, 5, 10, 50, 100, 200], learn a model on the 80% and calculate its performance (# of data classified incorrectly) on the validation data. What value of k appears to generalize best given your training data? Comment on the performance at the two endpoints, in terms of over- or under-fitting.

```
iris_training_x = X(1:118, 1:end);  % 118 is approx 80%
iris_training_y = Y(1:118);

iris_val_x = X(119:end, 1:end);
iris_val_y = Y(119:end);

% Define the K values to test
iris_k_values_large = [1, 2, 5, 10, 50, 100, 200];
errors = [];

for k = iris_k_values_large
    % Train a model
    iris_knn_learner = knnClassify(k, iris_training_x,
 iris_training_y);
    % Make some predictions
    prediction = predict(iris_knn_learner, iris_val_x);
    % Measure the errors in the predictions
```

```
    errors = [errors, numel(find(prediction~=iris_val_y))];

end

figure('Name','Performace of k');
plot(iris_k_values_large, errors./size(iris_val_x,1),'-*')
title('Performance of k');
xlabel('Value of k')
ylabel('Rate of incorrect predictions')
```



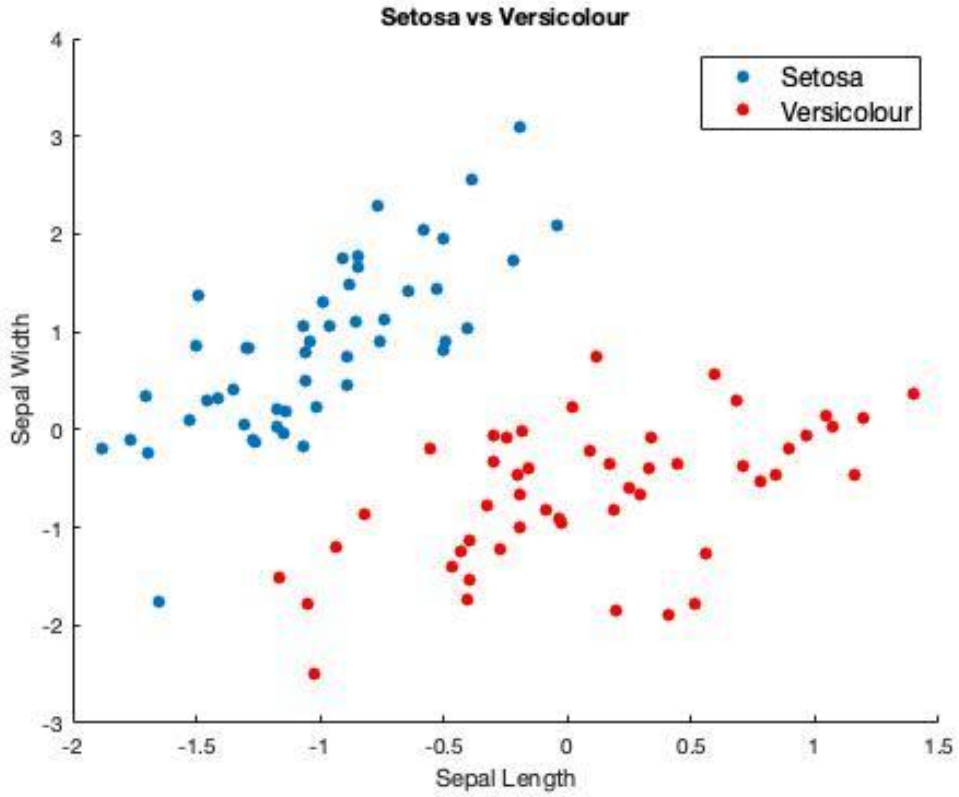
The above graph highlights the points made in part (c). The lower values of K overfit the data and the larger values underfit.

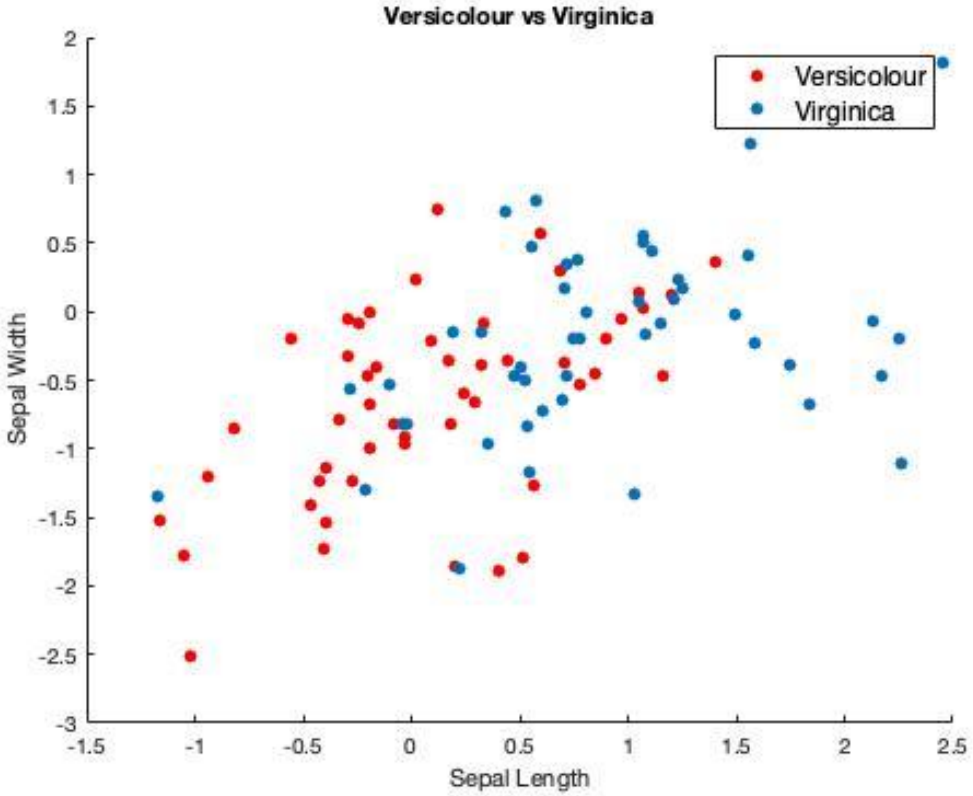
5. Perceptrons and Logistic Regression

(a) Show the two classes in a scatter plot and verify that one is linearly separable while the other is not.

```
iris = load('data/iris.txt');
X = iris(:,1:2); Y=iris(:,end);
[X, Y] = shuffleData(X,Y); % Randomise
X = rescale(X);
%Class 0 vs 1
XA = X(Y<2,:); YA=Y(Y<2);
%Class 1 vs 2
XB = X(Y>0,:); YB=Y(Y>0);
```

```
figure('Name','Setosa vs Versicolour');
hold on;
% find members of Class 0 and plot them
index_class_zero = find(YA==0);
x_points_class_zero = XA(index_class_zero, 1:end);
scatter(x_points_class_zero(:, 1), x_points_class_zero(:,
 2), 'filled');

% Find memebers of Class 1 and plot them
index_class_one = find(YA==1);
x_points_class_one = XA(index_class_one, 1:end);
scatter(x_points_class_one(:, 1), x_points_class_one(:,
 2), 'r', 'filled');
title('Setosa vs Versicolour');
xlabel('Sepal Length')
ylabel('Sepal Width')
legend('Setosa', 'Versicolour', 'FontSize', 12);
hold off;

figure('Name','Versicolour vs Virginica');
hold on;

% Already have class one so just need to plot it
scatter(x_points_class_one(:, 1), x_points_class_one(:,
 2), 'r', 'filled');

% Find memebers of Class 2 and plot them
index_class_two = find(YB==2);
x_points_class_two = XB(index_class_two, 1:end);
scatter(x_points_class_two(:, 1), x_points_class_two(:, 2), 'filled');
title('Versicolour vs Virginica');
xlabel('Sepal Length')
ylabel('Sepal Width')
legend('Versicolour', 'Virginica', 'FontSize', 12);
hold off;
```


Setosa vs Versicolour
Setosa
Versicolour
Sepal Width
Sepal Length
4
3
2
1
0
-1
-2
-3
-2
-1.5
-1
-0.5
0
0.5
1
1.5

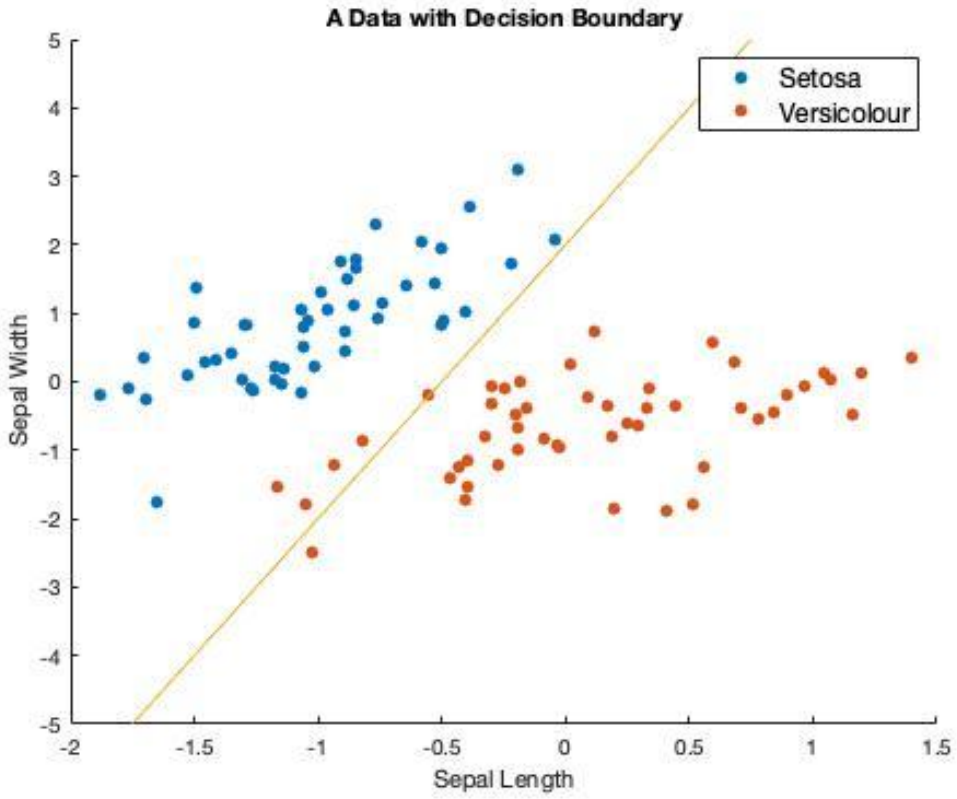

Versicolour vs Virginica
Versicolour
Virginica
Sepal Width
Sepal Length
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5

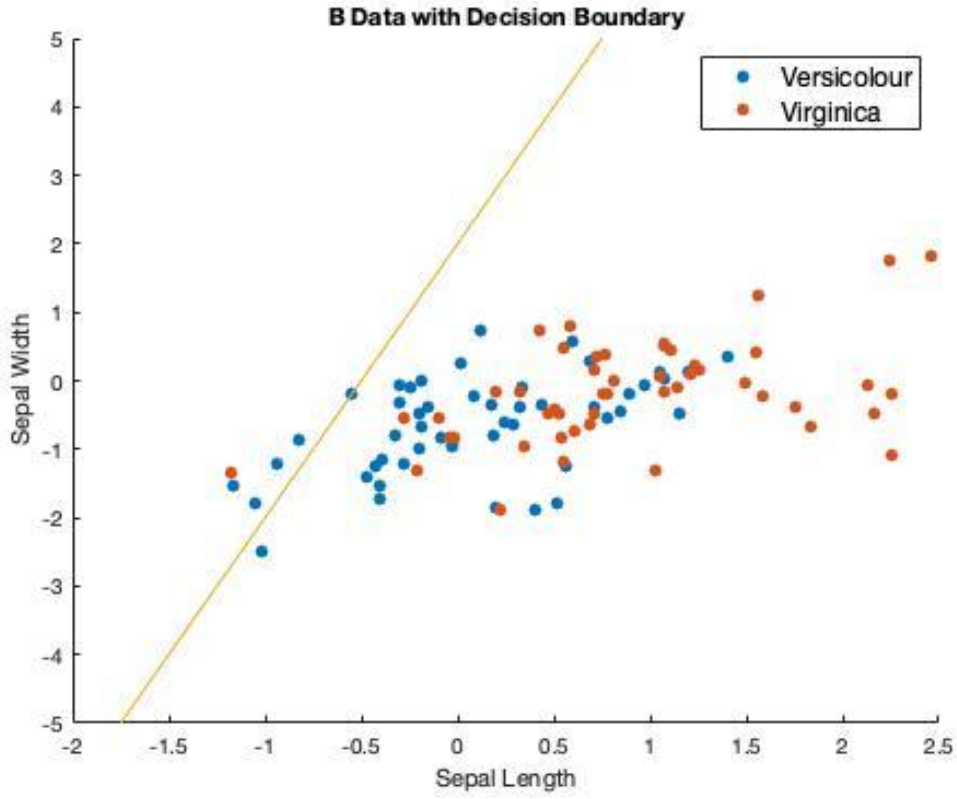
(b) Write (fill in) the function @logisticClassify2/plot2DLinear.m so that it plots the two classes of data in different colors, along with the decision boundary (a line). Include the listing of your code in your report. To demo your function plot the decision boundary corresponding to the classifier:

$$sign(.5 + 1x_1 - .25x_2)$$

```
function plot2DLinear(obj, X, Y)
% plot2DLinear(obj, X,Y)
%   plot a linear classifier (data and decision boundary) when
 features X are 2-dim
%   wts are 1x3,  wts(1)+wts(2)*X(1)+wts(3)*X(2)
%
  [n,d] = size(X);

  if (d~=2)
      error('Sorry -- plot2DLogistic only works on 2D data...');
  end

% Create the figure to plot the data on
figure('Name','2D Linear Classifier Plot');
hold on;

classes_in_Y = unique(Y)';
for c = classes_in_Y
    indicies_of_class_data = find(Y==c);
    x_values = X(indicies_of_class_data, 1:end);
    scatter(x_values(:, 1), x_values(:, 2), 'filled');
end

% Plot decision boundary.
wts = getWeights(obj);
f = @(x1, x2) wts(1) + wts(2)*x1 + wts(3)*x2;
fimplicit(f)
legend(strcat("Class ", num2str(classes_in_Y(1))), strcat("Class ",
 num2str(classes_in_Y(2))), 'Decision Boundary', 'FontSize',12);
hold off;


learner=logisticClassify2(); % create "blank" learner
learner=setClasses(learner, unique(YA)); % define class labels using
 YA or YB
wts = [0.5 1 -0.25];
learner=setWeights(learner, wts); % set the learner's parameters

plot2DLinear(learner, XA, YA);
title('A Data with Decision Boundary');
xlabel('Sepal Length')
ylabel('Sepal Width')
legend('Setosa', 'Versicolour', 'FontSize', 12);

plot2DLinear(learner, XB, YB);
title('B Data with Decision Boundary');
xlabel('Sepal Length')
```

```
ylabel('Sepal Width')
legend('Versicolour', 'Virginica', 'FontSize', 12);
```





(c) Complete the predict.m function to make predictions for your linear classifier. Note that, in my code, (c) the two classes are stored in the variable obj.classes?, with the first entry being the "negative" class (or class 0), and the second entry being the "positive" class. Again, verify that your function works by computing & reporting the error rate of the classifier in the previous part on both data sets A and B. (The error rate on data set A should be = 0.0505.)

```
function Yte = predict(obj,Xte)
% Yhat = predict(obj, X)  : make predictions on test data X

% (1) make predictions based on the sign of wts(1) + wts(2)*x(:,1)
 + ...
% (2) convert predictions to saved classes: Yte = obj.classes( [1 or
 2] );

wts = getWeights(obj);

f = @(x1, x2) wts(1) + wts(2)*x1 + wts(3)*x2;

function_data = sign(f(Xte(:,1), Xte(:,2)));

Yte = obj.classes(ceil((function_data+3)/2));
end


Y_Predictions_A = predict(learner,XA);
```

```
classification_error_A = numel(find(Y_Predictions_A~=YA));
final_classifiaction_error_A = classification_error_A/size(YA,1); % =
 0.0505
disp(strcat({'The error rate for Setosa vs Versicolour is:'},...
    {' '},{num2str(final_classifiaction_error_A,' %.4f')}));

Y_Predictions_B = predict(learner,XB);

classification_error_B = numel(find(Y_Predictions_B~=YB));
final_classifiaction_error_B = classification_error_B/size(YA,1);
disp(strcat({'The error rate for Versicolour vs Virginica is:'},...
    {' '},{num2str(final_classifiaction_error_B,' %.4f')}));
```

```
    'The error rate for Setosa vs Versicolour is: 0.0505'

    'The error rate for Versicolour vs Virginica is: 0.5455'
```

(d) In my provided code, I first transform the classes in the data Y into "class 0" (negative) and "class 1" (positive). In our notation, let $z = \theta x^{(i)}$ is the linear response of the perceptron, and σ is the standard logistic function

$$\sigma(z) = (1 + exp(-z))^{-1}$$

The (regularized) logistic negative log likelihood loss for a single data point j is then:

$$J_j(\theta) = -y^j \log \sigma\left(\theta x^{(j)T}\right) - (1 - y^{(j)}) \log\left(1 - \sigma\left(\theta x^{(j)T}\right)\right) + \alpha \sum_i \theta_i^2$$

Derive the gradient of the regularized negative log likelihood

First take the derivative of σ

$$\frac{\partial}{\partial z}\sigma(z) = \sigma\left(\Theta x^{(j)^T}\right)\left[1 - \sigma\left(\Theta x^{(j)^T}\right)\right]$$

Next derive the gradient

$$\frac{\partial J_j}{\partial \Theta_i} = \frac{\partial}{\partial \Theta_i} - y^{(j)} \log\left(\sigma\left(\Theta x^{(j)^T}\right)\right) - \frac{\partial}{\partial \Theta_i}(1 - y^j) \log\left(1 - \sigma\left(\Theta x^{(j)^T}\right)\right) + \frac{\partial}{\partial \Theta_i}\alpha \sum_i \theta_i^2$$

$$= \left[\frac{y^j}{\sigma\left(\Theta x^{(j)^T}\right)} - \frac{1 - y^j}{1 - \sigma\left(\Theta x^{(j)^T}\right)}\right] \frac{\partial}{\partial \Theta_i}^{-} \sigma\left(\Theta x^{(j)^T}\right) + 2\alpha\theta_i$$

Sub in derivative of σ

$$= \left[\frac{y^j - \sigma\left(\Theta x^{(j)^T}\right)}{\sigma\left(\Theta x^{(j)^T}\right)\left[1 - \sigma\left(\Theta x^{(j)^T}\right)\right]}\right]^{-} \sigma\left(\Theta x^{(j)^T}\right)\left[1 - \sigma\left(\Theta x^{(j)^T}\right)\right] x^i + 2\alpha\theta_i$$

$$= \left[y^j - \sigma\left(\Theta x^{(j)^T}\right)\right] *^{-} x^j + 2\alpha\theta_i$$

(e) Complete your train.m function to perform stochastic gradient descent on the logistic loss function.

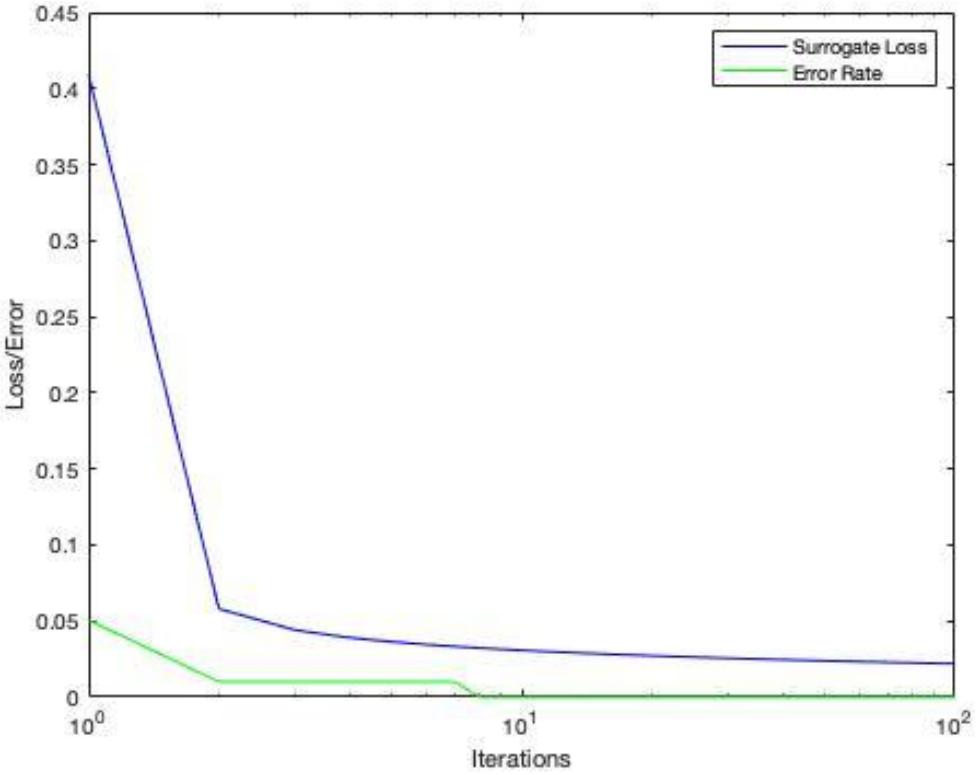
```
function obj = train(obj, X, Y, varargin)
% obj = train(obj, Xtrain, Ytrain [, option,val, ...])  : train
 logistic classifier
%     Xtrain = [n x d] training data features (constant feature not
 included)
%     Ytrain = [n x 1] training data classes
%     'stepsize', val  => step size for gradient descent [default 1]
%     'stopTol',  val  => tolerance for stopping criterion [0.0]
%     'stopIter', val  => maximum number of iterations through data
 before stopping [1000]
%     'reg', val       => L2 regularization value [0.0]
%     'init', method   => 0: init to all zeros;  1: init to random
 weights;
% Output:
%   obj.wts = [1 x d+1] vector of weights; wts(1) + wts(2)*X(:,1) +
 wts(3)*X(:,2) + ...


  [n,d] = size(X);            % d = dimension of data; n = number of
 training data

  % default options:
  plotFlag = false;
  init     = [];
  stopIter = 100; % Was 1000, reduced due to the whines of my laptop.
  stopTol  = -1;
  reg      = 0.0;
  stepsize = 1;

  i=1;                                        % parse through various
 options
  while (i<=length(varargin)),
    switch(lower(varargin{i}))
    case 'plot',      plotFlag = varargin{i+1}; i=i+1;   % plots on
 (true/false)
    case 'init',      init     = varargin{i+1}; i=i+1;   % init method
    case 'stopiter',  stopIter = varargin{i+1}; i=i+1;   % max # of
 iterations
    case 'stoptol',   stopTol  = varargin{i+1}; i=i+1;   % stopping
 tolerance on surrogate loss
    case 'reg',       reg      = varargin{i+1}; i=i+1;   % L2
 regularization
    case 'stepsize',  stepsize = varargin{i+1}; i=i+1;   % initial
 stepsize
    end
    i=i+1;
  end
```

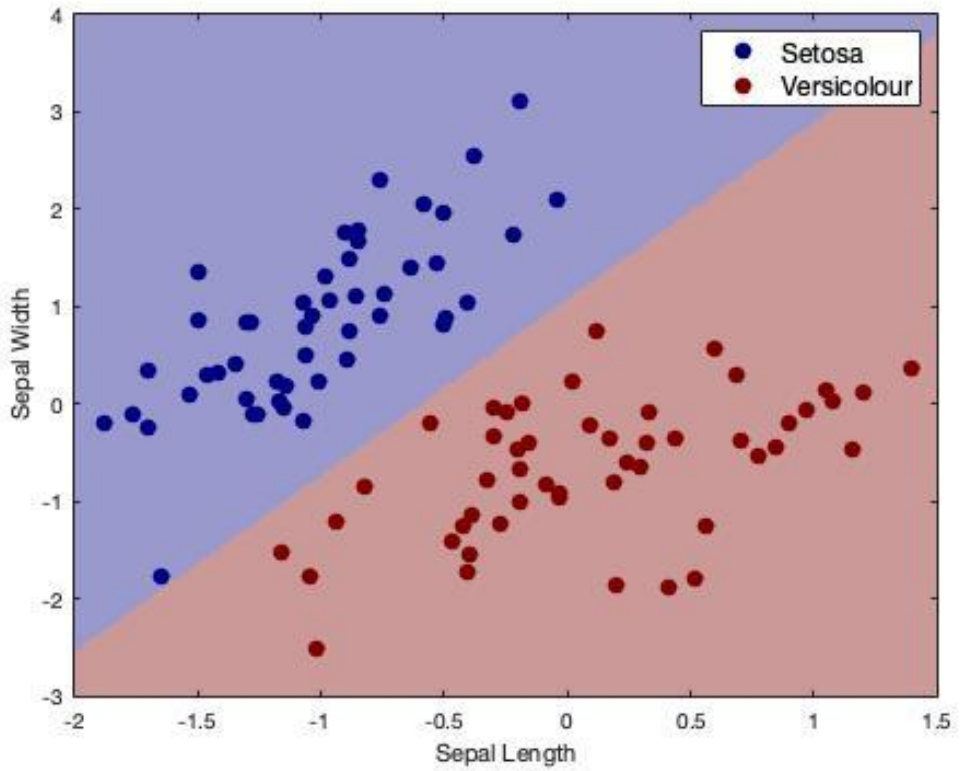
```
 X1    = [ones(n,1), X];     % make a version of training data with
the constant feature

 Yin = Y;                                % save original Y in case
needed later
 obj.classes = unique(Yin);
 if (length(obj.classes) ~= 2) error('This logistic classifier
requires a binary classification problem.'); end;
 Y(Yin==obj.classes(1)) = 0;
 Y(Yin==obj.classes(2)) = 1;           % convert to classic binary
labels (0/1)

 if (~isempty(init) || isempty(obj.wts))   % initialize weights and
check for correct size
   obj.wts = randn(1,d+1);
 end;
 if (any( size(obj.wts) ~= [1 d+1]) ) error('Weights are not sized
correctly for these data'); end;
 wtsold = 0*obj.wts+inf;

% Training loop (SGD):
iter=1; Jsur=zeros(1,stopIter); J01=zeros(1,stopIter); done=0;
while (~done)
  step = stepsize/iter;              % update step-size and evaluate
 current loss values

 %%% TODO: compute surrogate (neg log likelihood) loss
 wts_Square = (obj.wts.^2);
 Jsur(iter) = mean((-Y .* log(logistic(obj, X))) - ((1 - Y) .* log(1
- logistic(obj, X))) + reg * sum(wts_Square));
 J01(iter) = err(obj,X,Yin);

 if (plotFlag), switch d,            % Plots to help with
visualization
   case 1, fig(2); plot1DLinear(obj,X,Yin);  %  for 1D data we can
display the data and the function
   case 2, fig(2); plot2DLinear(obj,X,Yin);  %  for 2D data, just the
data and decision boundary
   otherwise, % no plot for higher dimensions... %  higher dimensions
visualization is hard
 end; end;
 fig(100);
 semilogx(1:iter, Jsur(1:iter),'b-',1:iter,J01(1:iter),'g-');
 legend('Surrogate Loss', 'Error Rate');
 xlabel('Iterations');
 ylabel('Loss/Error');
 drawnow;


 for j=1:n,
   % Compute linear responses and activation for data point j
   %%% TODO ^^^
   sigma = logistic(obj,X(j,:));
```

```
    % Compute gradient:
    %%% TODO ^^^
    grad = (Y(j)-sigma) * -X1(j,:); + 2 * reg * obj.wts;

    obj.wts = obj.wts - step * grad;      % take a step down the
 gradient .
  end

  done = false;
  %%% TODO: Check for stopping conditions

  J_Change = mean(-Y .* log(logistic(obj, X)) - (1 - Y) .* log(1 -
 logistic(obj, X)) + reg * obj.wts * obj.wts');
  if (iter == stopIter || abs(J_Change - Jsur(iter)) < stopTol)
    done = true;
  end
  wtsold = obj.wts;
  iter = iter + 1;
end
```

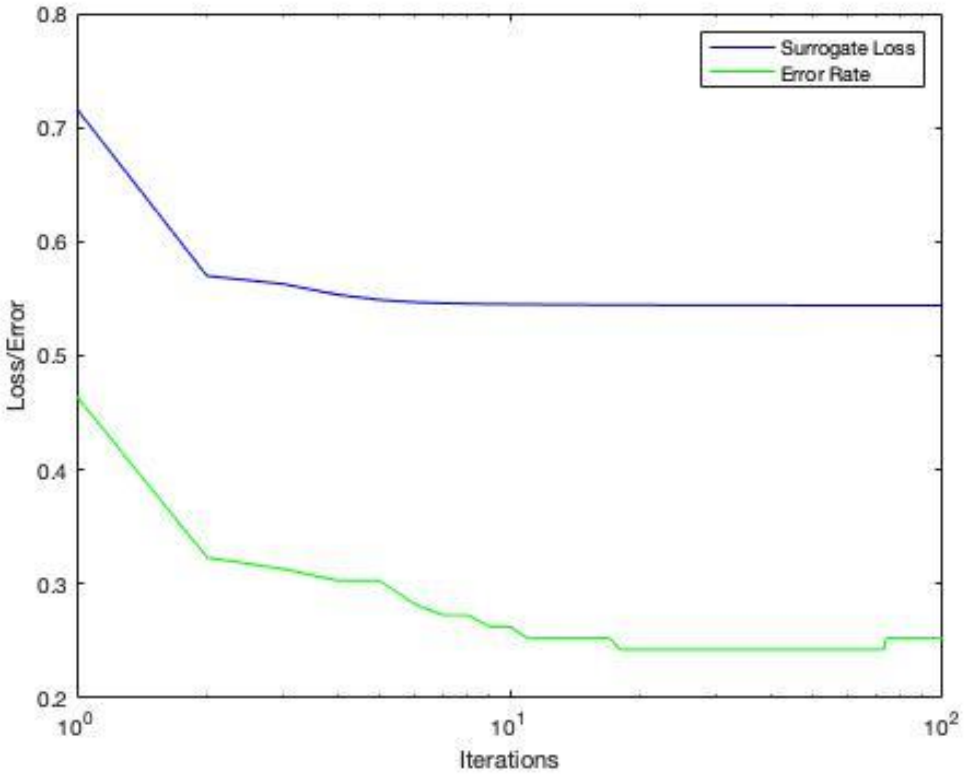
(f) Run your logistic regression classifier on both data sets (A and B); for this problem, use no regularization ($\alpha = 0$). Describe your parameter choices (stepsize, etc.) and show a plot of both the convergence of the surrogate loss and error rate, and a plot of the final converged classifier with the data.

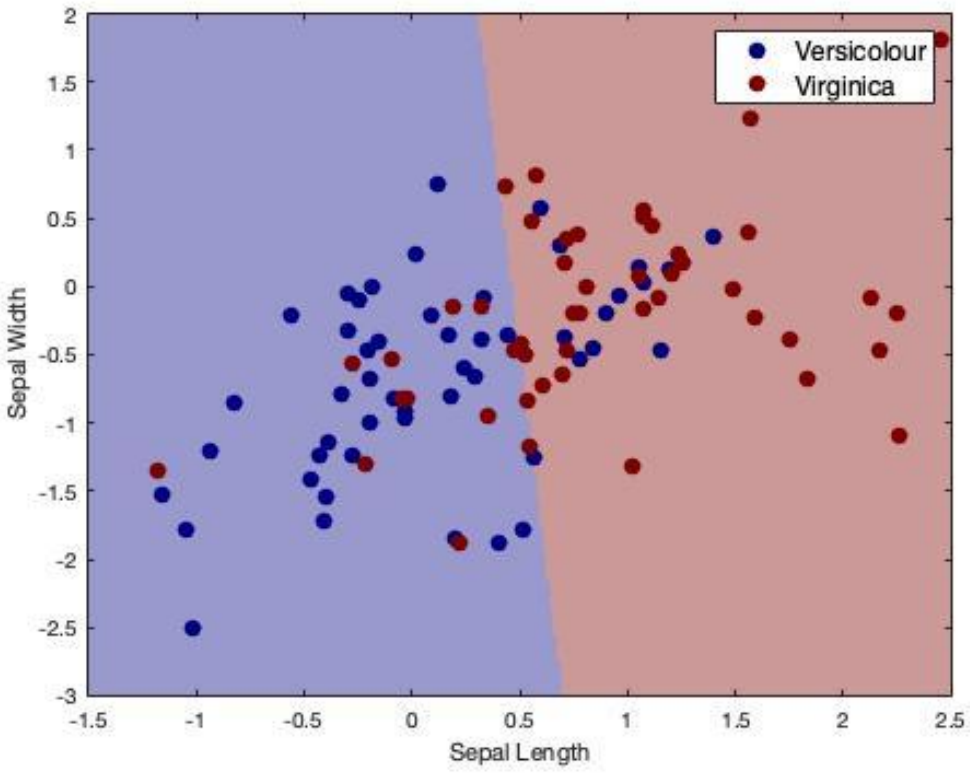
```
learner_A=logisticClassify2(); % create "blank" learner
learner_A=setClasses(learner_A, unique(YA)); % define class labels
 using YA or YB
wts = [0.5 1 -0.25];
learner_A=setWeights(learner_A, wts); % set the learner's parameters

learner_A = train(learner_A, XA, YA);

figure();
plotClassify2D(learner_A, XA, YA);
legend('Setosa', 'Versicolour', 'FontSize', 12);
xlabel('Sepal Length')
ylabel('Sepal Width')
```


Surrogate Loss
Error Rate
Loss/Error
Iterations
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10^0
10^1
10^2


Setosa
Versicolour
Sepal Width
Sepal Length
4
3
2
1
0
-1
-2
-3
-2
-1.5
-1
-0.5
0
0.5
1
1.5

```
new_YB = YB - 1;
learner_B=logisticClassify2(); % create "blank" learner
learner_B=setClasses(learner_B, unique(YA)); % define class labels
 using YA or YB
wts = [0.5 1 -0.25];
learner_B=setWeights(learner_B, wts); % set the learner's parameters
learner_B = train(learner_B, XB, new_YB);

% Plot final converged classifier decision boundaries.
figure();
plotClassify2D(learner_B, XB, new_YB);
legend('Versicolour', 'Virginica', 'FontSize', 12);
xlabel('Sepal Length')
ylabel('Sepal Width')
```





The default values in the train.m file where left in place except for the number of iterations. The iterations where reduced from 1000 to 100 as there was minimal inprovement in the error rate beyond 100.

(g) Implement the mini batch gradient descent on the logistic function

```
function mini_batches = create_mini_batches(obj, X,y, batch_size)

pi = randperm(size(X,1));
data_values = [X(pi,:), y(pi,:)];    %shuffle your data

n_mini_batches = size(X,1) / batch_size; %based on your data and the
 batch size compute the number of batches
mini_batches = zeros(batch_size,3,n_mini_batches);

for i = 1:n_mini_batches
   %extract the minibatch values
   start = batch_size * (i - 1) + 1;
   end_i = start + batch_size - 1;
   mini_batches(1:batch_size, :, i) = data_values(start:end_i,:);
end

end
```

Published with MATLAB® R2018b