CAB420 – Machine Learning

Assignment 1

Student 1: Ian Daniel – n5372828

Student 2: Jia Sheng Chong – n9901990

Note: The corresponding Matlab file is called 'SOLUTION.m'

0. Theory (10 Marks)

Working with the following logistic regression function:

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

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

$$\frac{\partial L}{\partial w_j} = -\sum_{i=1}^{N} y_i x_{ij} \left(1 - g\left(y_i(w^T x_i + b)\right)\right) + 2\lambda w_j$$

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

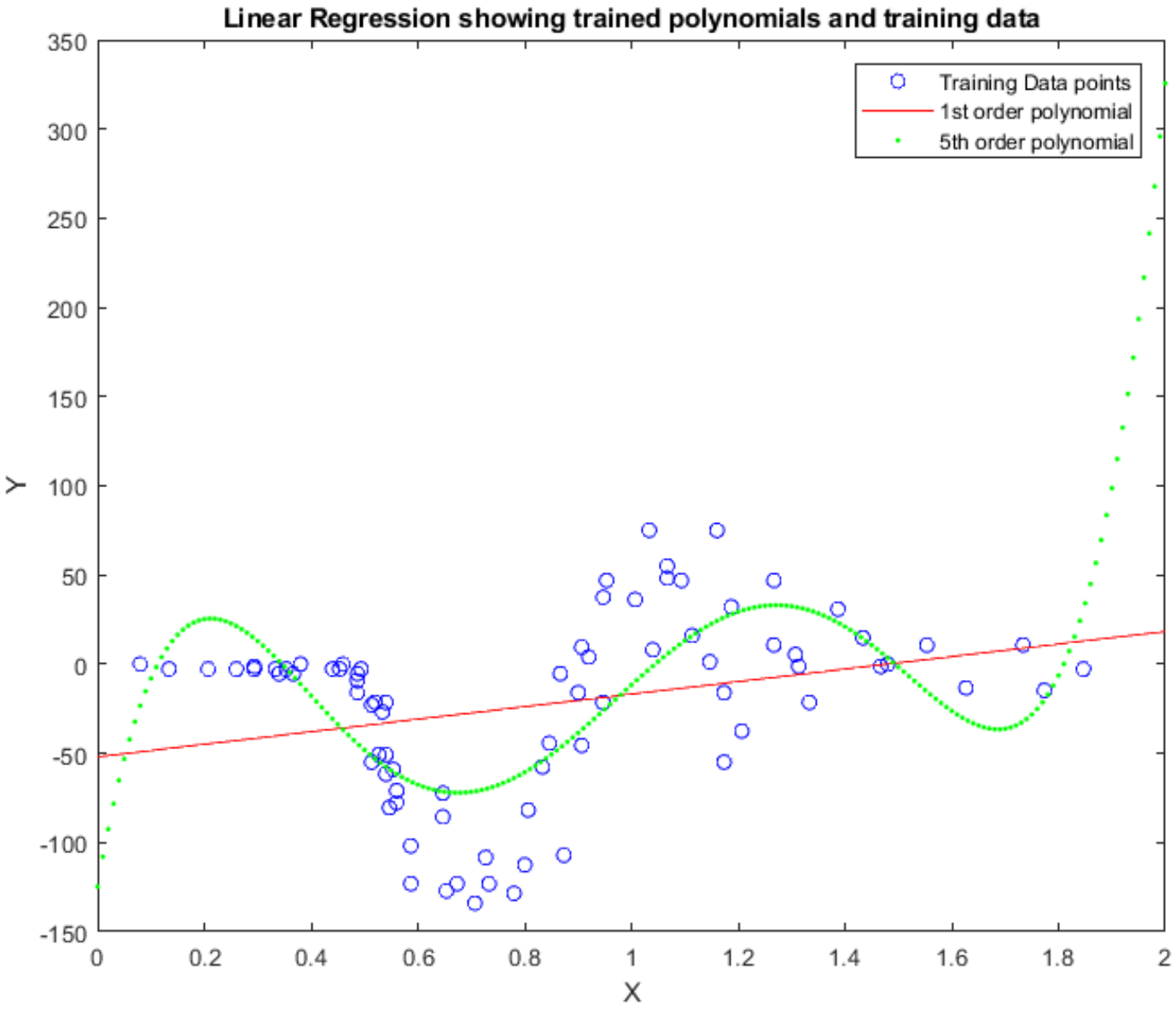
$$\frac{\partial L}{\partial w_j \partial w_k} = -\sum_{i=1}^{N} y_i^2 x_{ij} x_{ik} g\left(y_i(w^T x_i + b)\right)\left(1 - g\left(y_i(w^T x_i + b)\right)\right) + 2\lambda \varpi_{jk}$$

ϖ_{jk} = 1 if $i = j$ and 0 if $i \neq j$

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

$$a^T H a = \sum_{i,j,k} a_j a_k y_i^2 x_{ij} x_{ik} g\left(y_i(w^T x_i + b)\right)\left(1 - g\left(y_i(w^T x_i + b)\right)\right) + 2\lambda \sum_j a_j^2 \geq 0$$

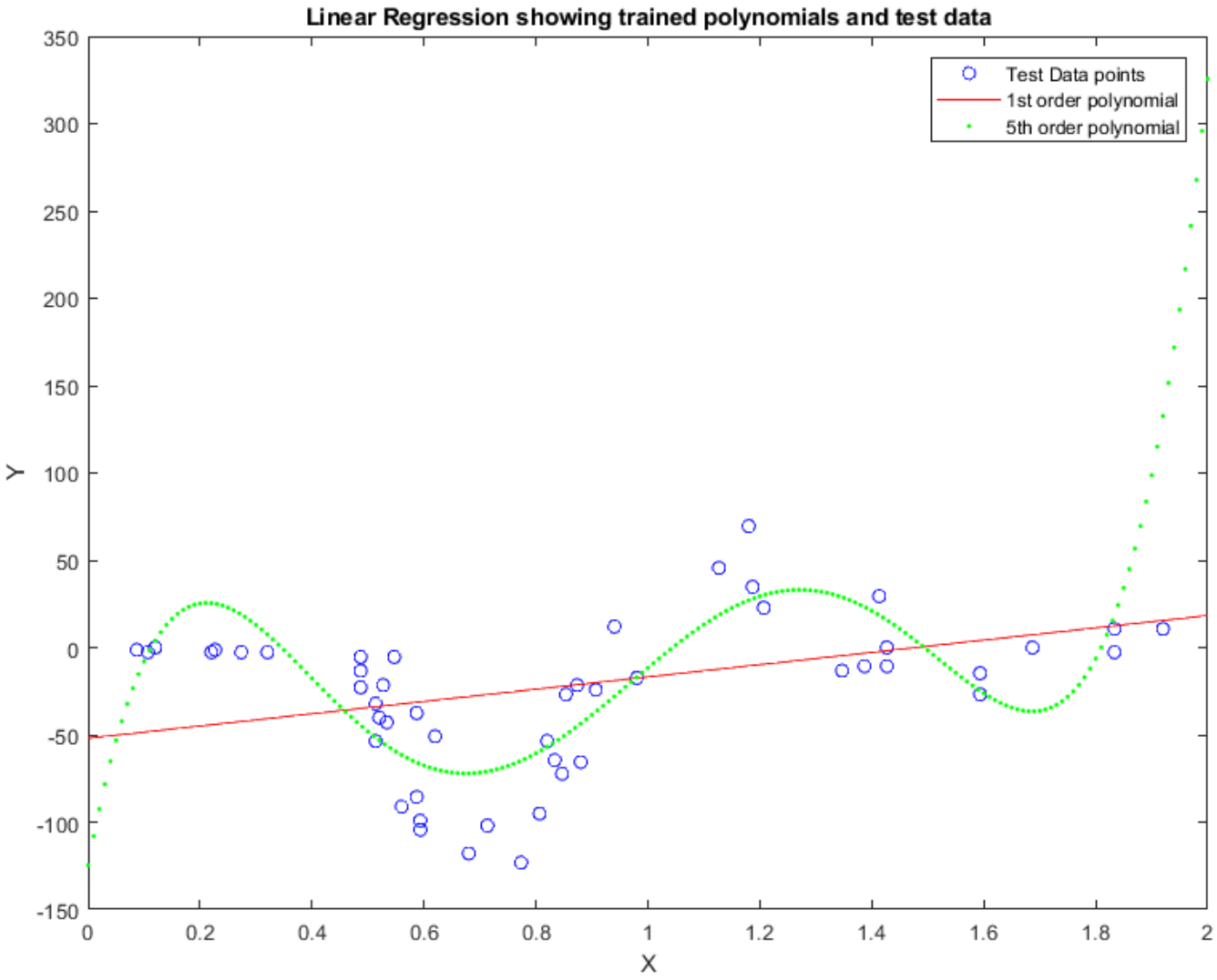
1. **Features, Classes and Linear Regression (10 Marks)**



The above plot shows the training data points (blue circles) along with the 1st and 5th order polynomial. The 1st order polynomial shows a consistent positive gradient however the error is clearly high. The 5th order polynomial fits the data much closer, however before the first and after the last data point the error is extreme.

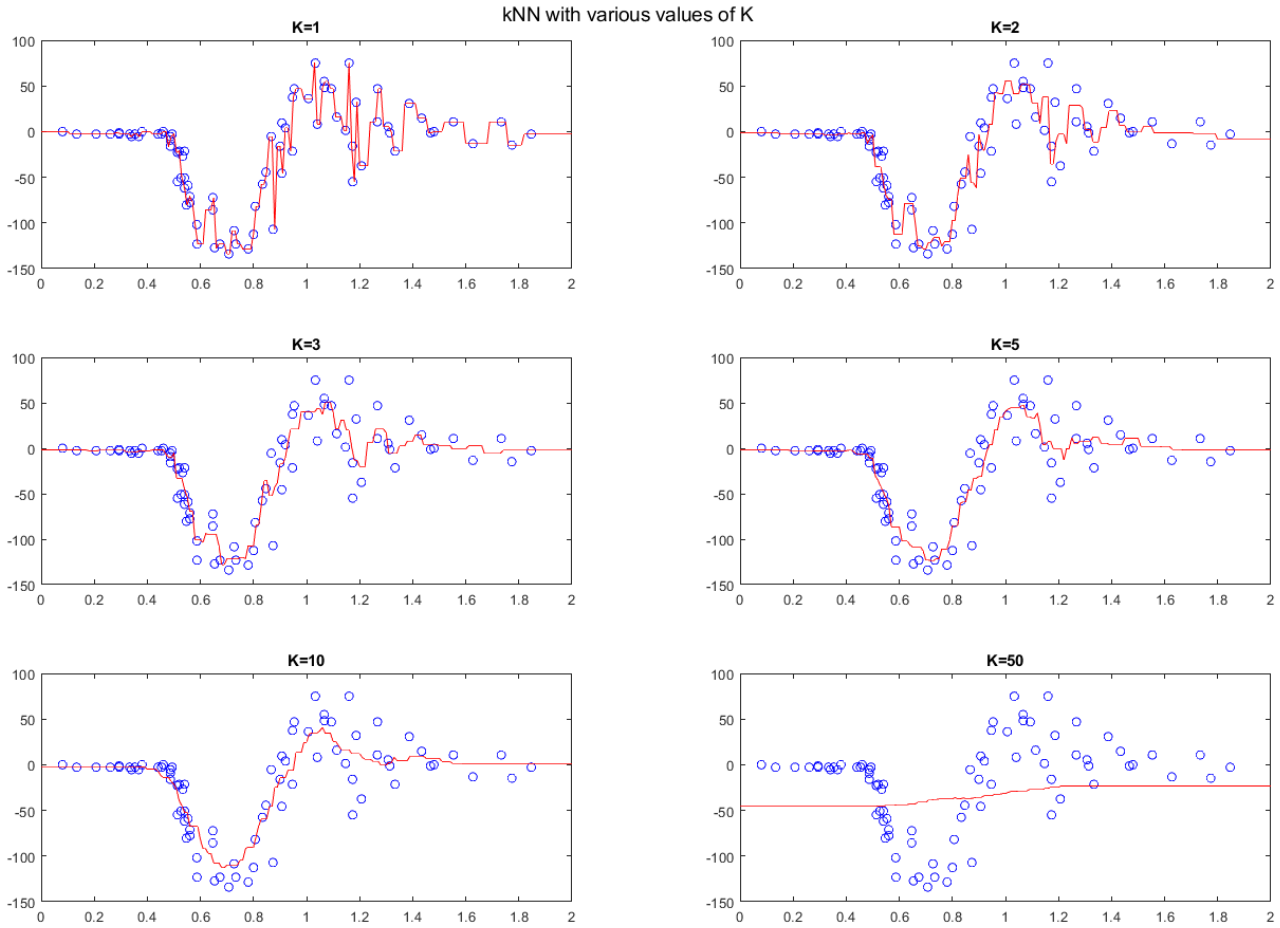
The mean squared error (MSE) for the 1st order polynomial is 2270.8 and the 5th order polynomial is 1254.6, this MSE reflects what can be seen in the plot – almost half the error. The MSE of the test data has less error, possible because there is 62.5% less data points. None the less, the learned function still fits the curve of the data points. 1st order polynomial is 1704.9 and the 5th order polynomial is 999.7, below is a plot with the test data and the two polynomials.

For the creation of the polynomial the 'x' data, which is a column vector, is passed through a simple function that returns a matrix. Each column of the matrix is the order of magnitude that the data calculated with. For example, column 0 is x^0 which will always be 1. Column 1 is x^1 which will also be the original data. Column 2 will be x^2 and so on. This data is passed onto the training class linearReg and then the predictor method predict. MSE is calculated by averaging the sum of all the squared differences between the predicted Y values and actual Y values.



2. kNN Regression (15 Marks)

The following image shows the same training data that was used in question 1, expect that this uses kNN (k nearest neighbour) instead of linear regression. The values of k are used to determine the number of data points to calculate the mean average. As seen in the below image, the red line for k=1 is simply following every data point. As the value of K increases, so does the number of mean values calculated smoothing the line. There is a point however that the line can be over-fit as k=50 is a clear example of this, it is almost a straight line.

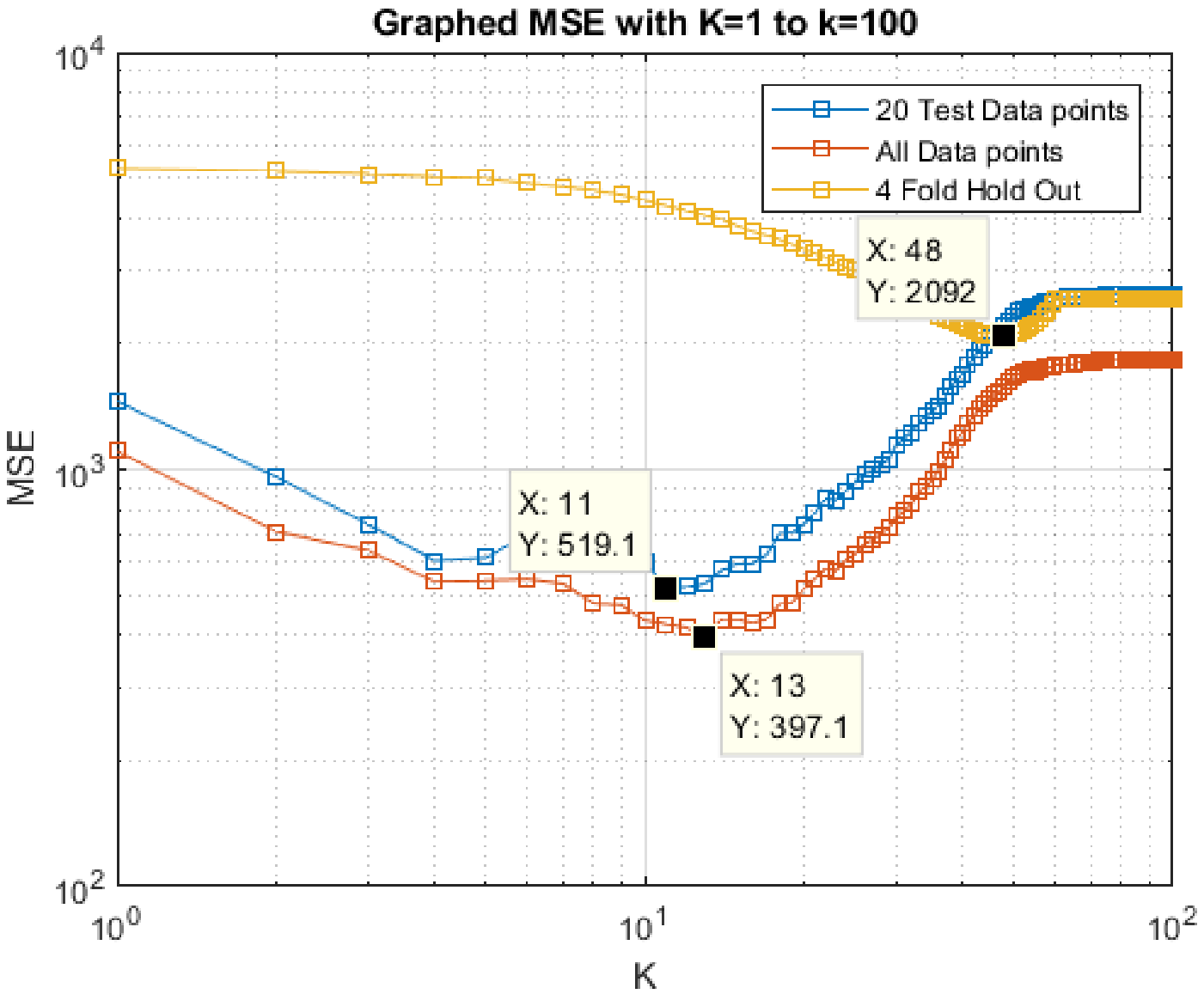


Simple based of the above 6 figures, k=10 appears to be the best fit. This is because it is the smooths line available while still following most of the data points though the curve. With an exception being around x=0.6 to x=0.8. K=5 follows those points better however the line has more noise. In the next question, the MSE will show the best k value would be between 11 and 13. This supports the above figure for k=10 as the best for this group.

3. Hold-out and Cross-validation (15 Marks)

Below shows the MSE used the test data. The blue line is using only 20 out of the 50 points available for the learner while the red is using them all, the yellow line is using a 4 fold hold out. The lowest MSE error when using only 20 test data points is 519.1, this is when k=11. Using more test data does improve the MSE error, it lowers to 387.1 when k=13. Regardless the error increases significantly after both of these points and levels out just past k=50. When looking at the previous figures, k=50 is almost a horizontal line.

The yellow line (4 Fold Hold Out) uses a portion of the training data for its test data. In this case we used a 60/20 ratio which is training and test data points respectively. As you can see by the below figure the error is much higher compared to using all 80 data points and the other 50 for test validation. Also the k value is almost 4 times higher at k=48. If you look at the above figure for k=50 – this is almost a straight line. Typically this training method is only used when there isn't much data to learn from.

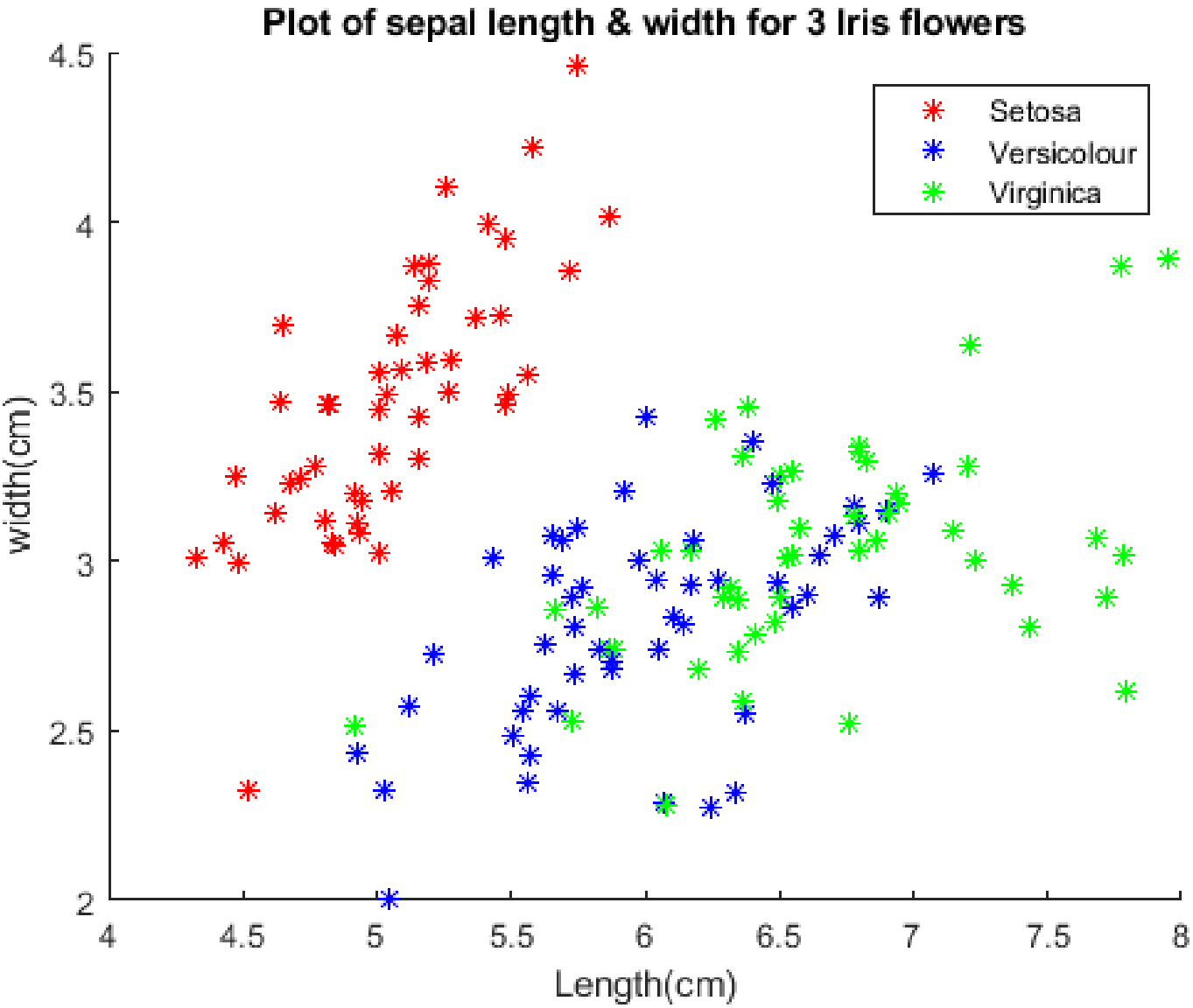


By contrast the kNN training method is more complex compared to linear regression. However the predictions follows the data as the MSE show. Linear regression's 5th order polynomial has an error of 999.7 while kNN with k=13 has an error of only 397.1 – that is a 2.5 times improvement. While the error is considerably high compared to either Linear regression (5th order) or kNN (k=13) for the 4 fold hold out. If there isn't much data to train from it is better than nothing for automatic training, this could be used until there is more data available then the train models could be recalculated.

(4). Nearest Neighbour Classifiers (15 Marks)

With this section I like to start explaining the data. This is from the widely used data set called Iris. It has 50 data points for each class and there are 3 classes. However in the data set that I will be using there is 2 data points missing, one from class 0 and the other from class 2. It is unknown why they are not there. Each row of data contain the length and width of both the sepal and petal of a flower. The 3 classes/flowers are 0 = Iris Setosa, 1 = Iris Versicolour and 2 = Iris Virginica.

The first step will be simply to show the length and width of the sepal (in cm) for the 3 flowers.



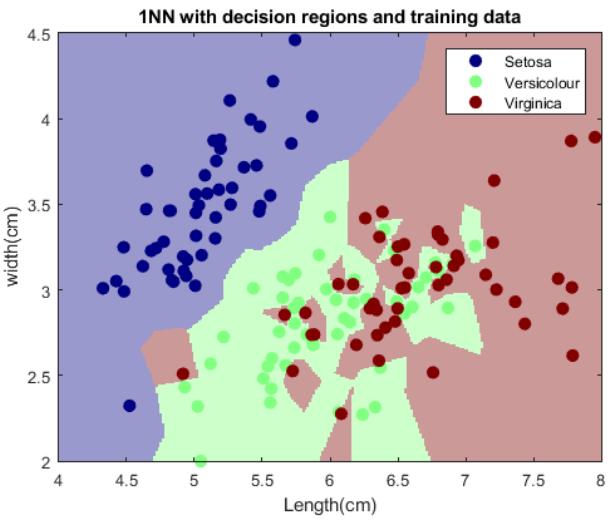
One class can be linearly separable (Setosa vs Versicolour/Virginica) and the other two cannot (Versicolour & Virginica).

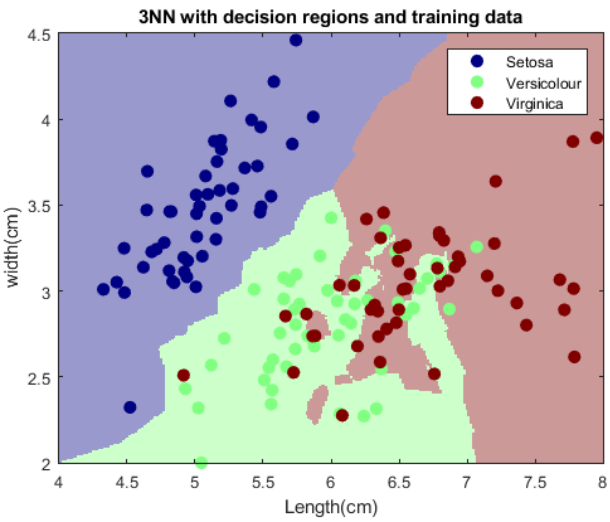
The method for plotting the 3 classes is to first determine that unique values used to separate the 3 in the dataset. Then simply using a switch statement in Matlab to select the colour of the *.

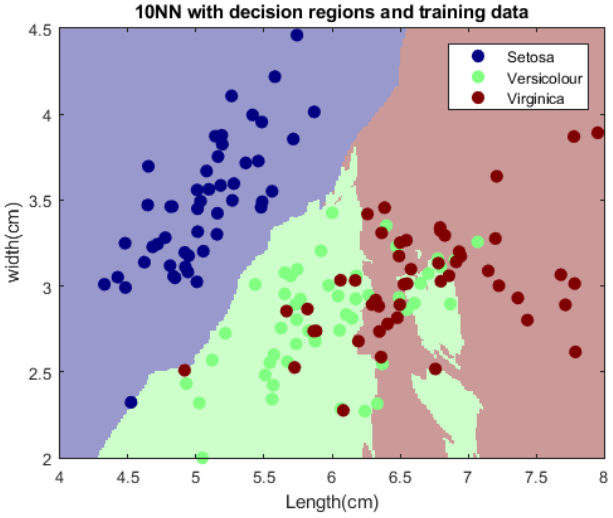
The follow 4 figures show kNN Classifier with different values of k: 1, 3, 10 and 30. As you can see 1NN doesn't deal well with outliers. The Virginica (red) data point bear (5, 2.5) has a noticeable area extending into Setosa's (blue) area. There is another Virginica data point near (6.1, 2.3) that should be part of Versicolour's (green) area.

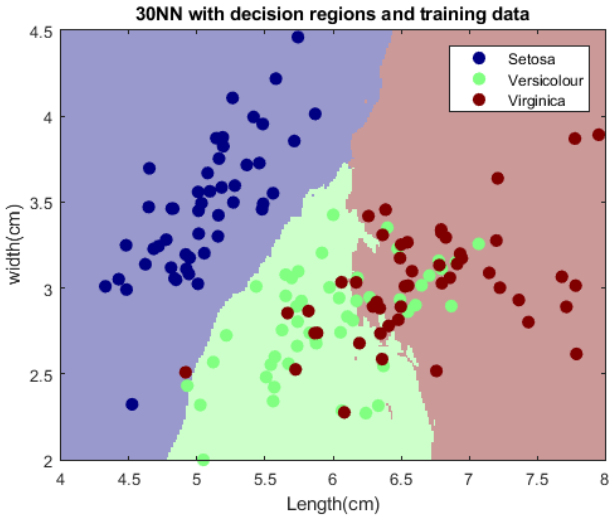
3NN mostly solves the 2 previously mentioned outliers by claiming that space or Versicolour. The Virginica boundary starts to push out the Versicolour outliers. Boundary lines continue to smooth out as the value of k increases.

By 10NN the boundary between blue and red/green has become quite clear and it is not reacting to the blue outlier near (4.5, 2.3). There is however an incorrect boundary associated to Versicolour around the co-ordinates (6.8, 2.25). As this region was there when k=3 that region was connected to the main green boundary and made sense.



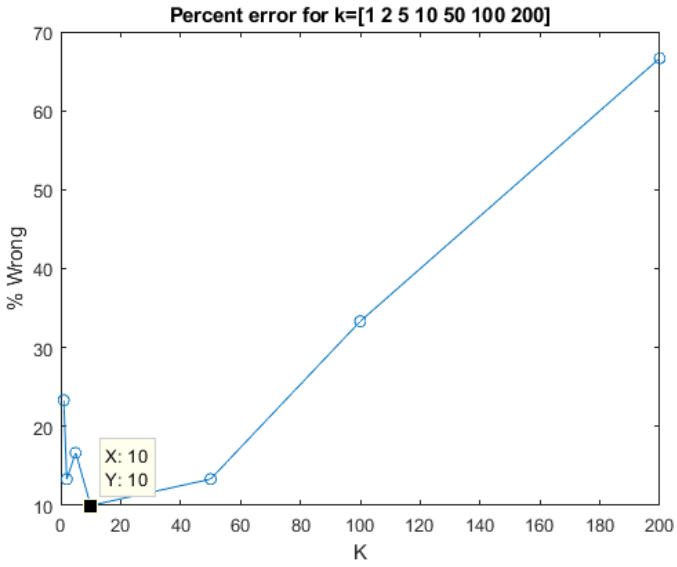






By 30NN the Versicolour boundary around (6.8, 2.25) has shifted to the left with the main boundary and now makes sense when looking at the figure. The Setosa outlier near (4.5, 2.25) is now affecting the boundary line and there is a notable number of Versicolour data points in the Virginica boundary area around the point (6.6, 3).

The figure to the right shows 7 data points which is the percentage of incorrect predictions. Running this test multiple times will provide slightly different answers because the split between training and test is different every time. That said, any values of k=100 and higher has consistently high error rates. Typically k values between 10 and 30 work well.

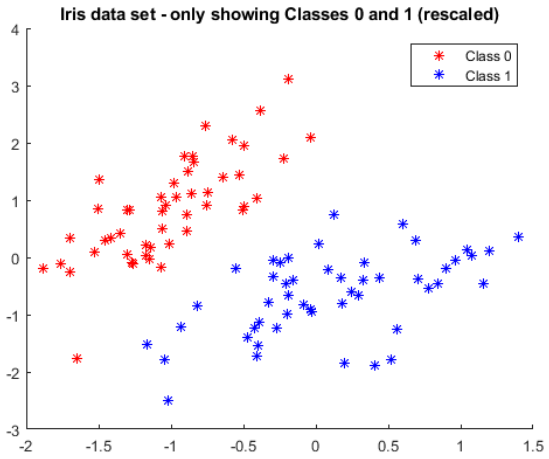


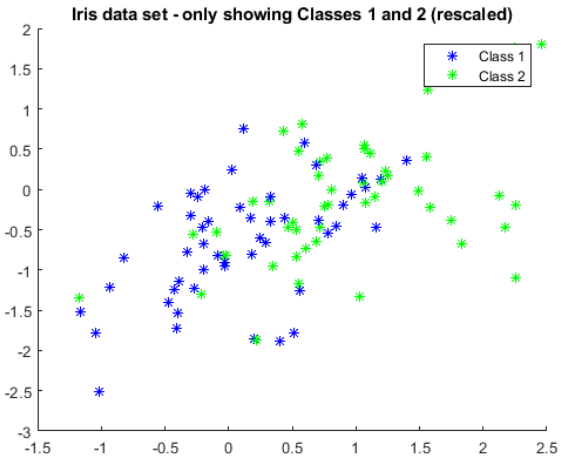
Looking at the two end-points of the test data, k=1 easily under fits and has when there is two classes occupying the same space. At the other end k=200 over fits the training data. A visual figure can't not be provided for this as Matlab throws an error saying

“Subscript indices must either be real positive integers or logicals”. However what I imagine this would look like is data points that are not outliers are treated as such. So the boundaries may not be covering the areas that would be useful.

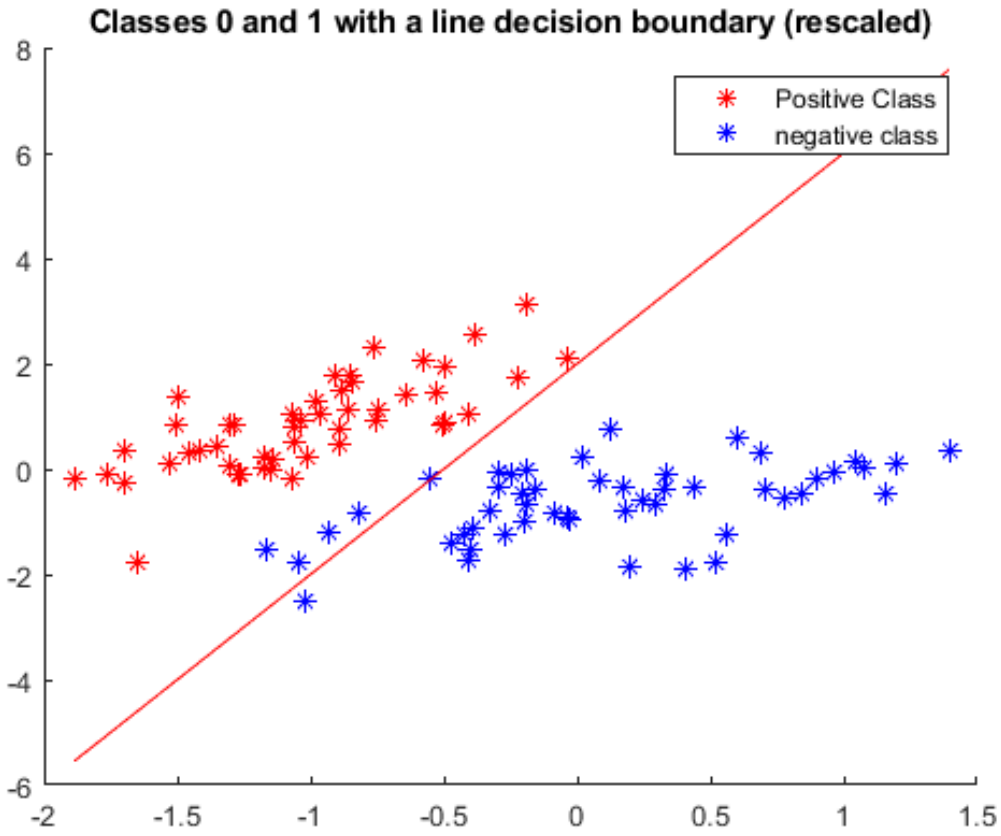
(5) Perceptrons and Logistic Regression (25 Marks)

Starting once again with the Iris data set. First displaying classes 0 and 1 in a scatter plot that are linearly separable. The second scatter plot has classes 1 and 2 that is clearly not separable.

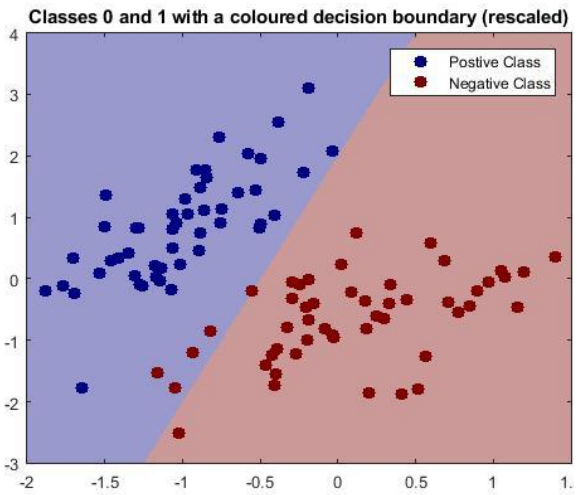


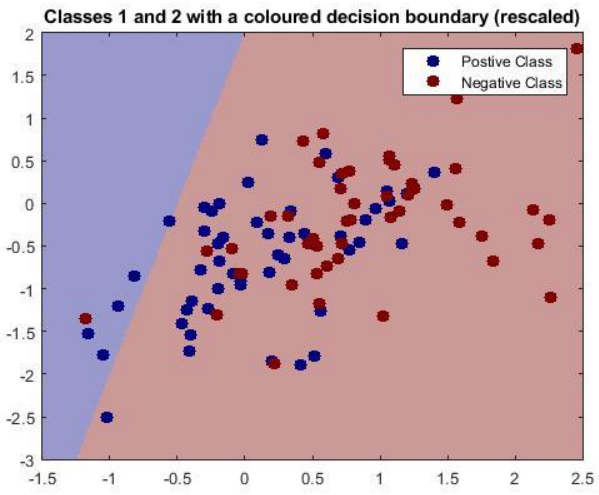


Using the weights 0.5, 1 and -0.25 a line decision boundary with the classes 0 and 1.



Completing the predict function for the linear classifier displays the following figure. The negative class in the blue and positive red. The log learner error is 0.0505.





The above graphs shows the coloured decision boundary of data sets (A and B). The regularization has been set to 0, and our parameter choices are all at the default.

Below is the train.m code that we written:

```
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 = true;
  init     = [];
  stopIter = 1000;
  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
  % Jsur(iter) = inf;   %%% TODO: compute surrogate (neg log likelihood) loss
  Jsur(iter) = mean((log(logistic(obj, X)) .* -Y) - (log(1 - logistic(obj, X)) .* (1 - Y)) +
reg * sum((obj.wts * obj.wts')'));
  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(1); semilogx(1:iter, Jsur(1:iter),'b-',1:iter,J01(1:iter),'g-'); drawnow;

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

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

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


  %%% TODO: Check for stopping conditions
  JD = mean(-Y .* log(logistic(obj, X)) - (1-Y) .* log(1 - logistic(obj, X)) + reg * obj.wts *
obj.wts');
  if abs(JD - Jsur(iter)) || iter == stopIter
      done = true;
  end

  wtsold = obj.wts;
  iter = iter + 1;
end;
```