CAB420 - MACHINE LEARNING

Assignment 1

Phuoc Nguyen n9951733

Long Thai n10006257

Theory

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

a) Finding the partial derivative

Simplifying $L(w)$:

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

Finding $\frac{\partial L}{\partial w_j}$:

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

$$\frac{\partial}{\partial w_j}(\lambda \|w\|_2^2) = 2\lambda w_j$$

$$\begin{aligned} \frac{\partial}{\partial w_j}\left(\log\left(1 + e^{y_i(w^T x_i + b)}\right)\right) &= \frac{\partial}{\partial w_j}\left(1 + e^{y_i(w^T x_i + b)}\right) \cdot \frac{1}{1 + e^{y_i(w^T x_i + b)}} \\ &= x_i y_i e^{y_i(w^T x_i + b)} \cdot \frac{1}{1 + e^{y_i(w^T x_i + b)}} \\ &= \frac{x_i y_i e^{y_i(w^T x_i + b)}}{1 + e^{y_i(w^T x_i + b)}} \end{aligned}$$

$$\therefore \frac{\partial L}{\partial w_j} = \sum_{i=1}^{N} \frac{x_i y_i e^{y_i(w^T x_i + b)}}{1 + e^{y_i(w^T x_i + b)}} + 2\lambda w_j$$

b) Finding the second partial derivative

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

Let $u = x_i y_i e^{y_i(w^T x_i + b)}$ and $v = 1 + e^{y_i(w^T x_i + b)}$

$$\frac{\partial}{\partial w_k}\left(\frac{x_i y_i e^{y_i(w^T x_i + b)}}{1 + e^{y_i(w^T x_i + b)}}\right) = \frac{\frac{\partial}{\partial w_k}(u) \cdot v - \frac{\partial}{\partial w_k}(v) \cdot u}{v^2}$$

$$\frac{\partial}{\partial w_k}(u) \cdot v = \frac{\partial}{\partial w_k}\left(x_i y_i e^{y_i(w^T x_i + b)}\right) \cdot \left(1 + e^{y_i(w^T x_i + b)}\right)$$
$$= x_i^2 y_i^2 e^{y_i(w^T x_i + b)} \cdot \left(1 + e^{y_i(w^T x_i + b)}\right)$$
$$= x_i^2 y_i^2 e^{y_i(w^T x_i + b)} + x_i^2 y_i^2 e^{2y_i(w^T x_i + b)}$$

$$\frac{\partial}{\partial w_k}(v) \cdot u = \frac{\partial}{\partial w_k}\left(1 + e^{y_i(w^T x_i + b)}\right) \cdot x_i y_i e^{y_i(w^T x_i + b)}$$
$$= x_i y_i e^{y_i(w^T x_i + b)} \cdot x_i y_i e^{y_i(w^T x_i + b)}$$
$$= x_i^2 y_i^2 e^{2y_i(w^T x_i + b)}$$

$$\frac{\partial}{\partial w_k}(u) \cdot v - \frac{\partial}{\partial w_k}(v) \cdot u = x_i^2 y_i^2 e^{y_i(w^T x_i + b)} + x_i^2 y_i^2 e^{2y_i(w^T x_i + b)} - x_i^2 y_i^2 e^{2y_i(w^T x_i + b)}$$
$$= x_i^2 y_i^2 e^{y_i(w^T x_i + b)}$$

$$\frac{\partial}{\partial w_k}\left(\frac{x_i y_i e^{y_i(w^T x_i + b)}}{1 + e^{y_i(w^T x_i + b)}}\right) = \frac{x_i^2 y_i^2 e^{y_i(w^T x_i + b)}}{\left(1 + e^{y_i(w^T x_i + b)}\right)^2}$$

$$\therefore \frac{\partial^2 L}{\partial w_j w_k} = \sum_{i=1}^{N} \frac{x_i^2 y_i^2 e^{y_i(w^T x_i + b)}}{\left(1 + e^{y_i(w^T x_i + b)}\right)^2}$$

c) Show L(w) is a convex function

$$a^T H a \equiv \sum_{j,k} a_j a_k H_{j,k} \geq 0$$

$\sum_{j,k} a_j a_k H_{j,k} \geq 0$ is true as all elements in H is greater than or equal to 0.

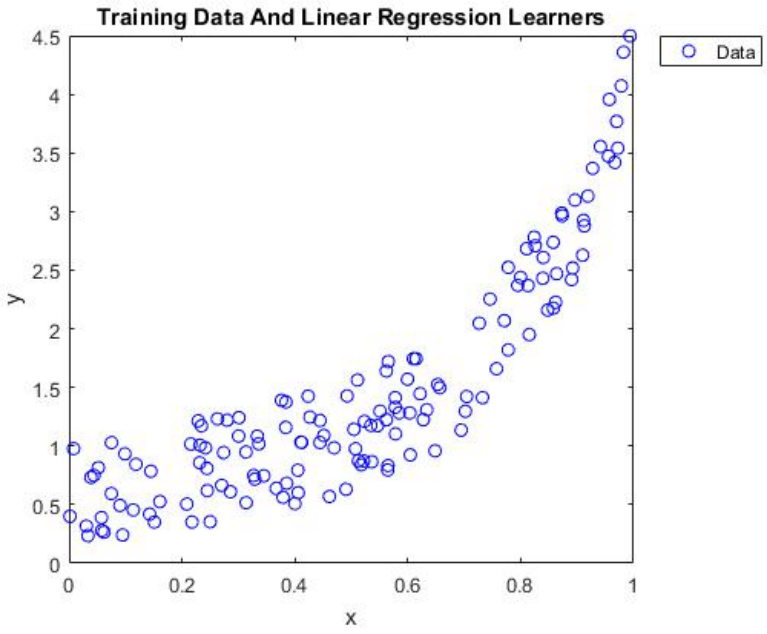
$\therefore L(w)$ is convex as its Hessian, H_L, is positive semi-definite.

Practice

1. Features, Classes, and Linear Regression

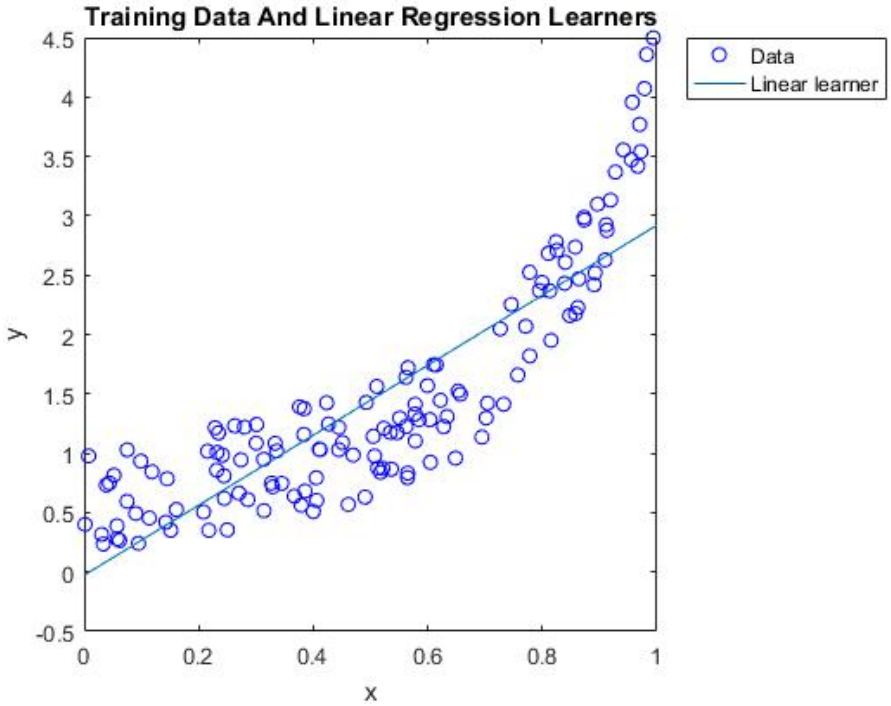
(Run setup code in Appendix 1)

a) Plot the training data in a scatter plot



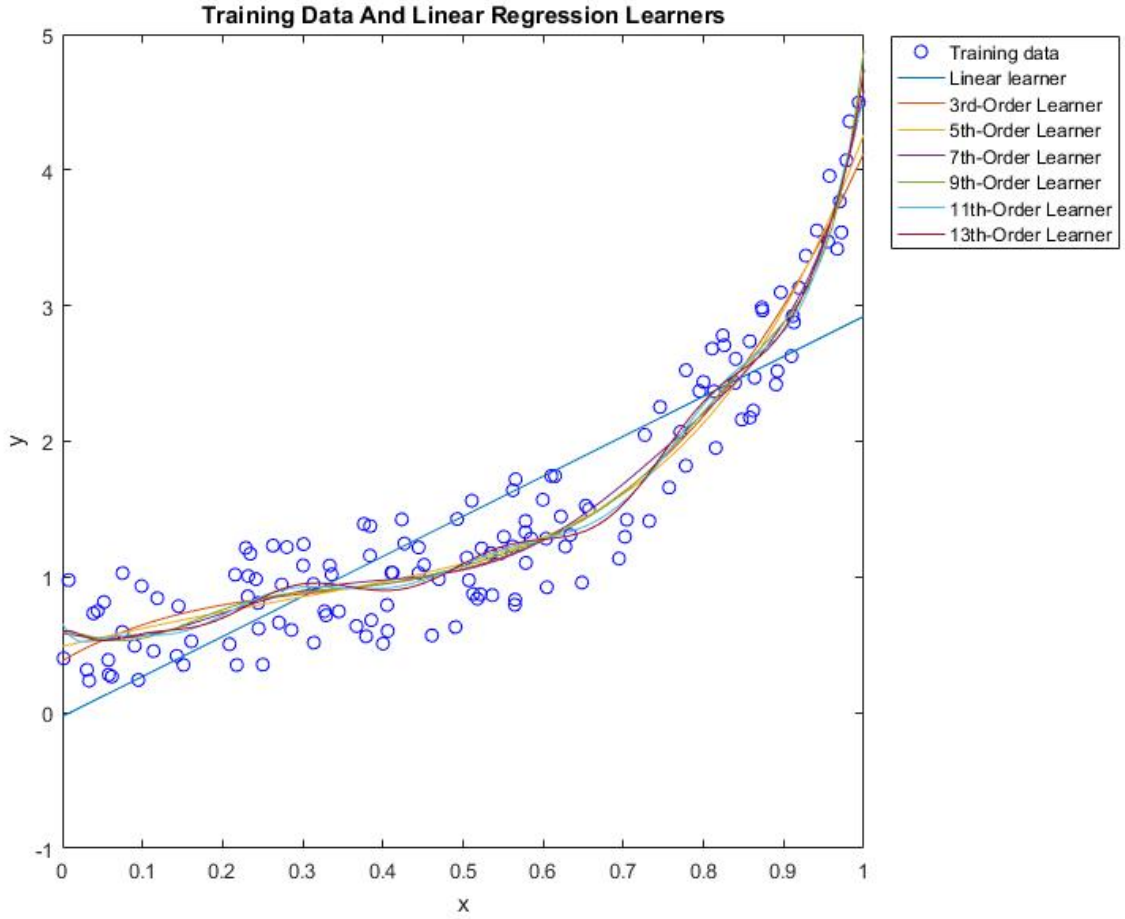
(Refer to Appendix 2 for code)

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



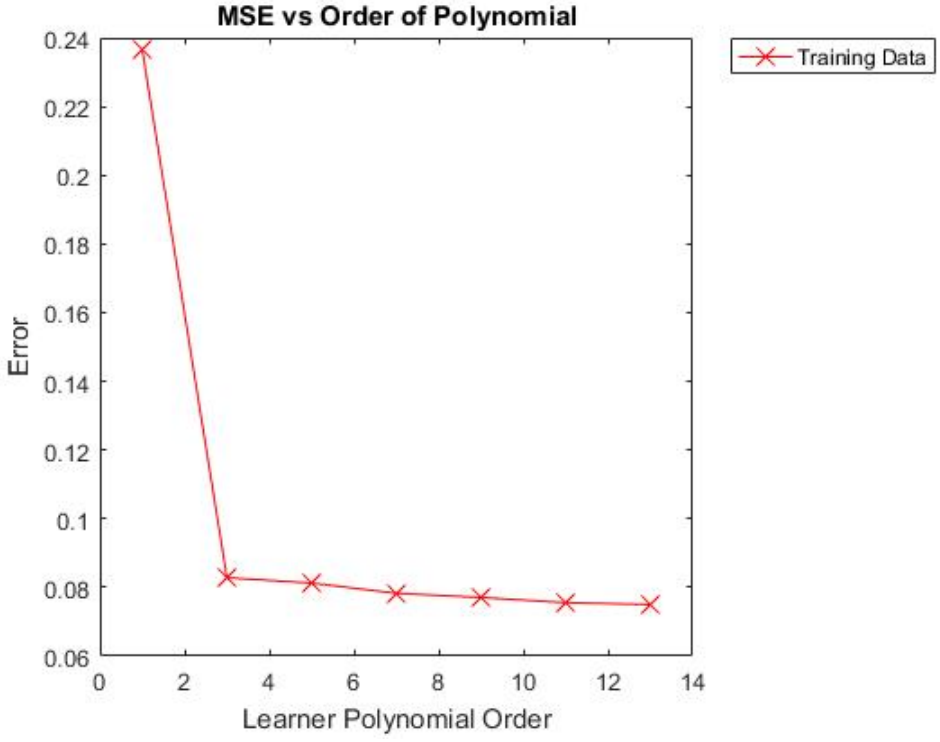
(Refer to Appendix 3 for code)

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



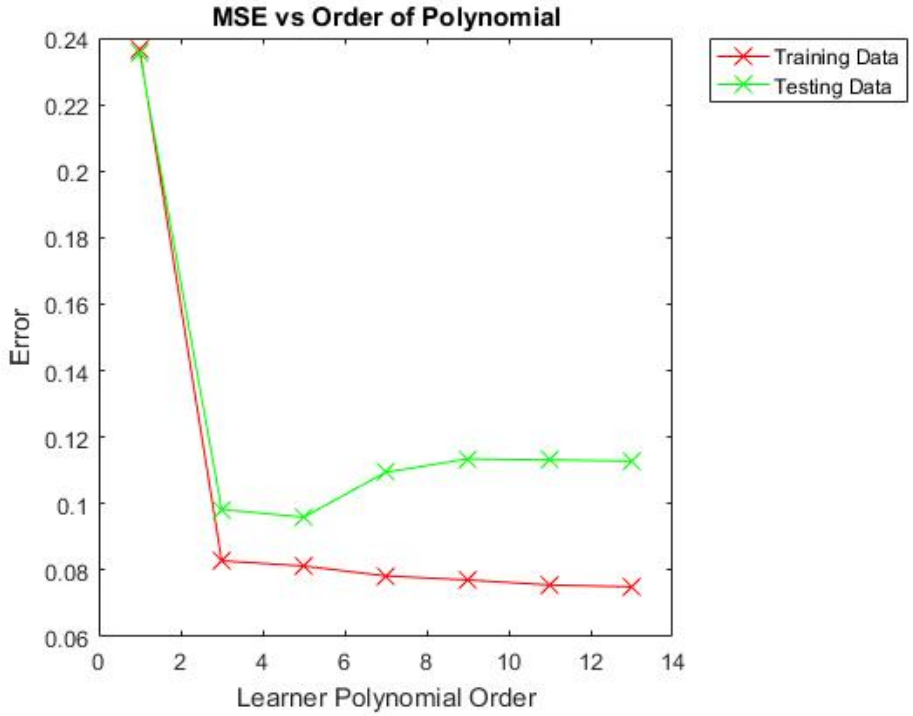
(Refer to Appendix 4 for code)

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



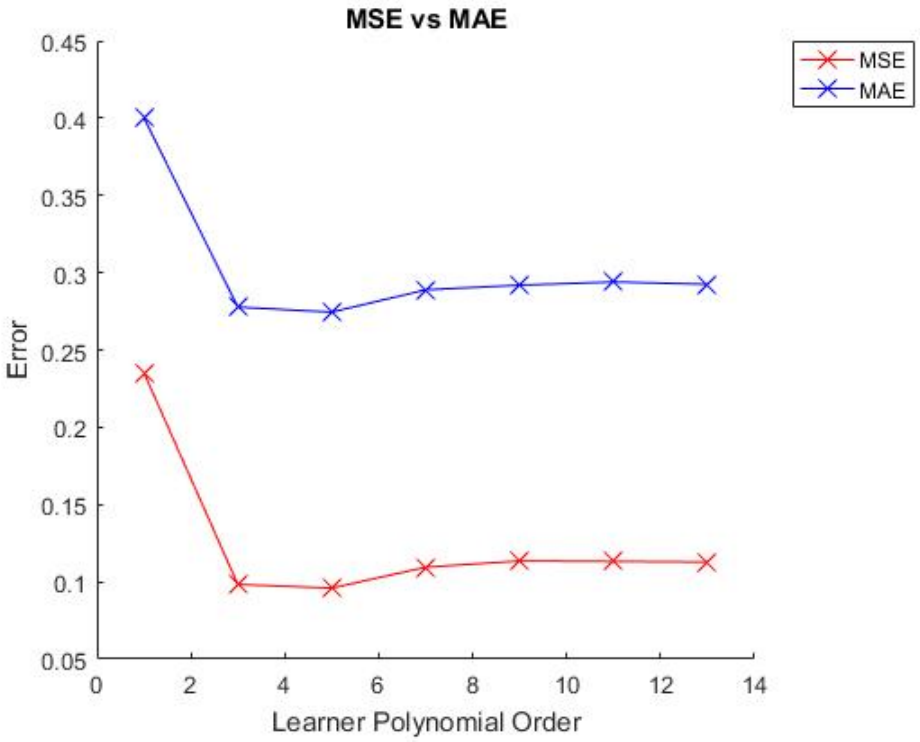
(Refer to Appendix 5 for code)

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



(Refer to Appendix 6 for code)

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



(Refer to Appendix 7 for code)

g) Don't forget to label your plots; see help legend.

2. kNN Regression

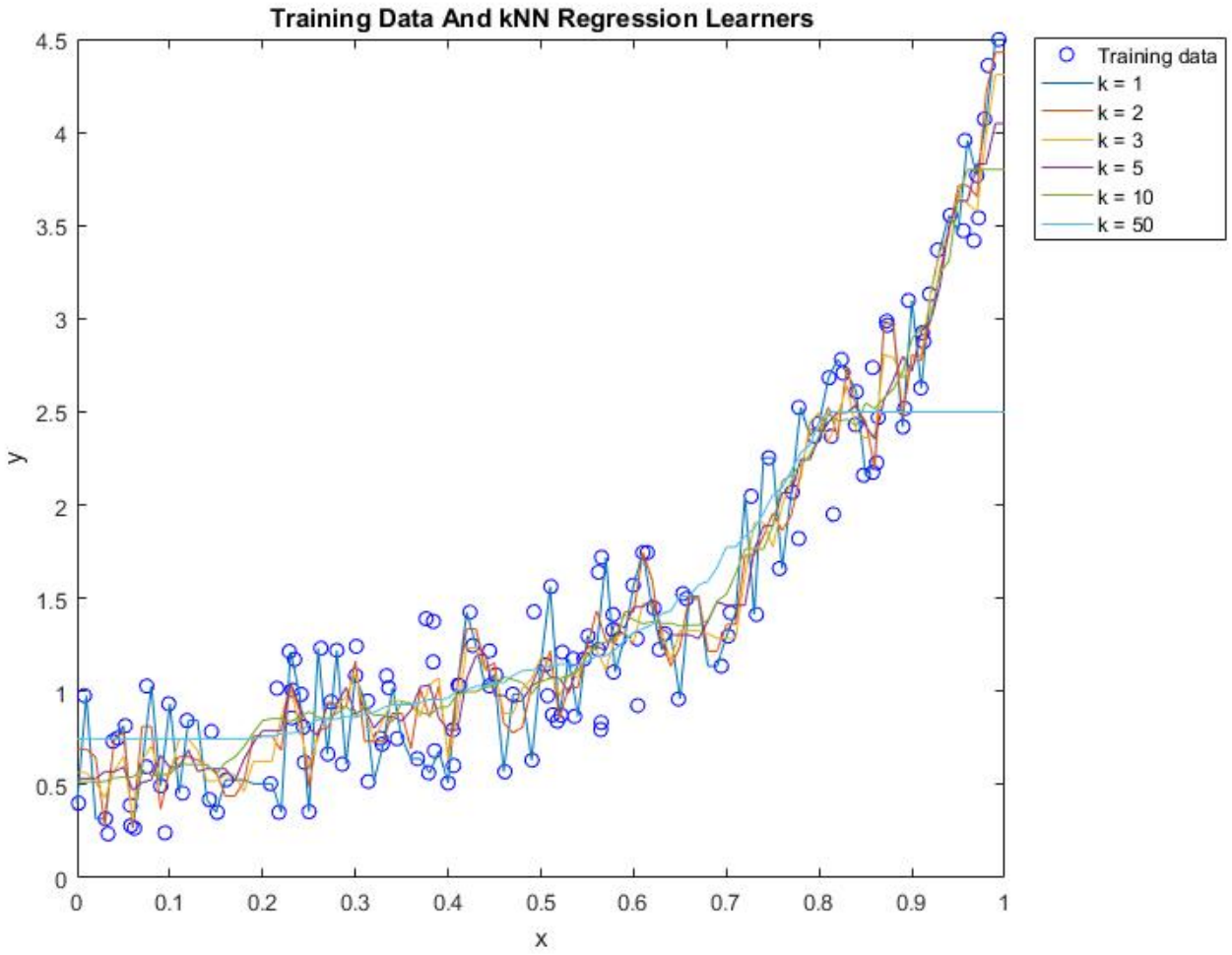
(Run setup code in Appendix 8)

a) Using the knnRegress class, implement (add code to) the predict function to make it functional. Comment the line error('You should write prediction code here');, otherwise you will get an error.

```
% -- OUR CODE -- %
indices = idx(1:K);
kNearest = obj.Ytrain(indices);
Yte(i)=  mean(kNearest);            % predict ith test example's value from nearest neighbors
% -- END OF OUR CODE -- %
```

(Refer to Appendix 9 for complete code of predict.m)

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



(Refer to Appendix 10 for code)

```
% How does the choice of k relate to the "complexity" of the regression function?
%   From the plot it can be seen that as k increases, the regression
%   function's complexity decreases.
```

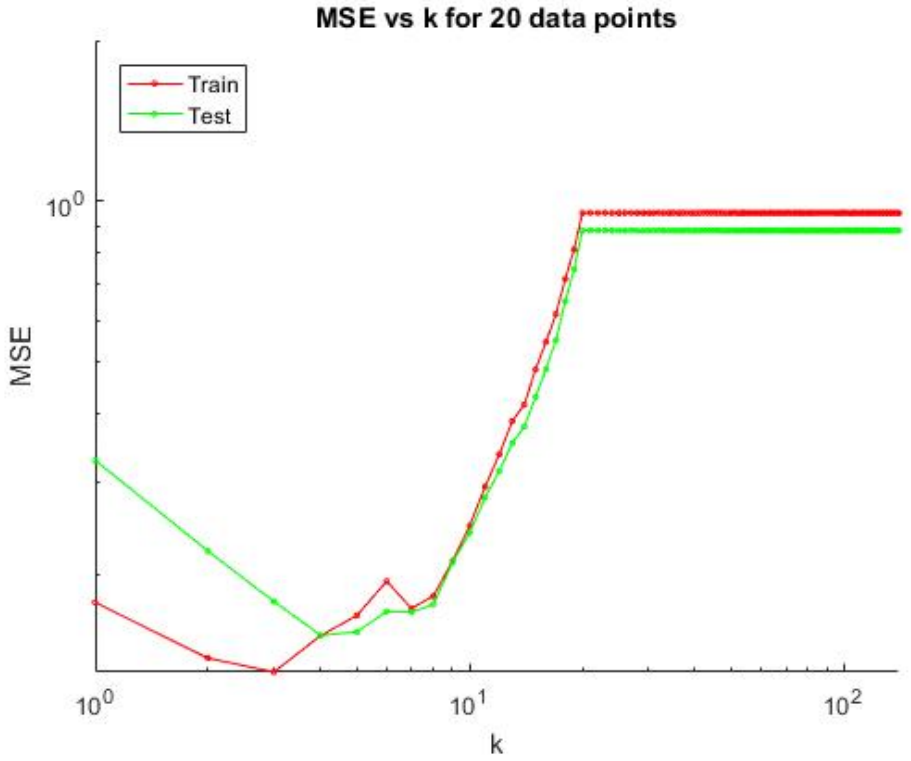
c) We discussed in class that the k-nearest-neighbor classifier's decision boundary can be shown to be piecewise linear. What kind of functions can be output by a nearest neighbor regression function? Briefly justify your conclusion. (You do not need to discuss the general case – just the 1-dimensional regression picture such as your plots.)

```
%% (c) What kind of functions can be output by a nearest neighbor regression
function?
%   The functions outputted by the nearest neighbor regression function are
%   all piece wise linear as all x values that share the same neighbors
%   will correspond to the same y value meaning it is piece wise and linear.
```

3. Hold-out and Cross-validation

(Run setup code in Appendix 11)

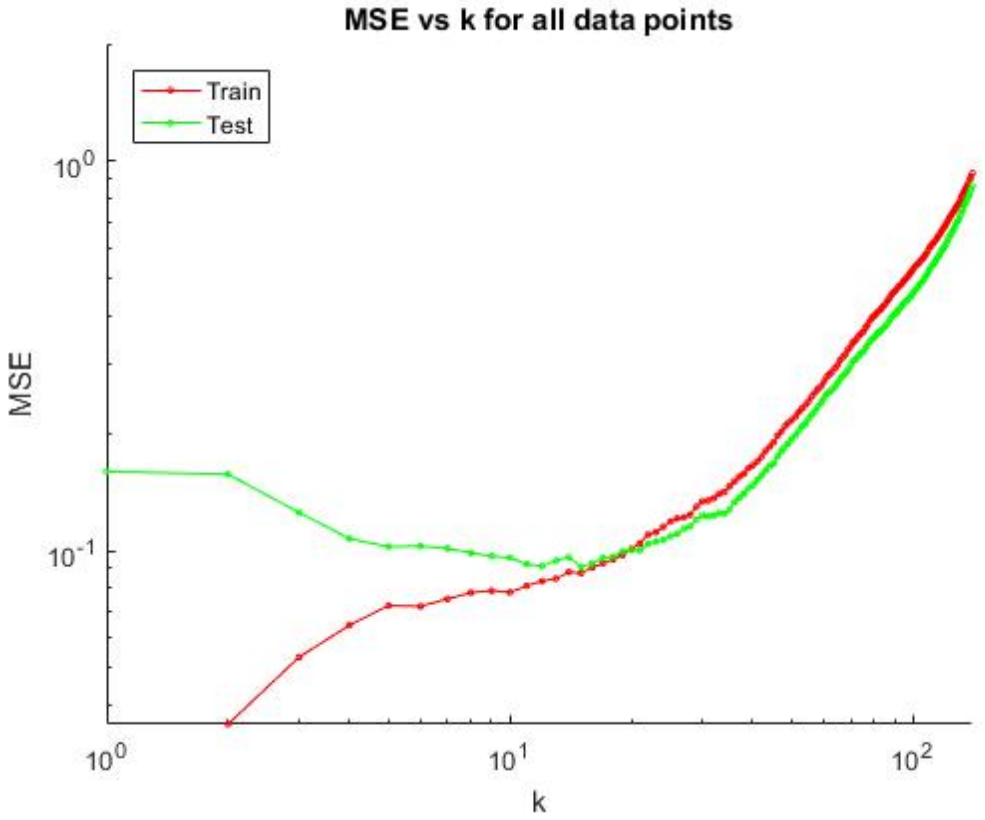
a) Similarly to Problem 1 and 2, compute the MSE of the test data on a model trained on only the first 20 training data examples for k = 1, 2, 3, . . . , 140. Plot both train and test MSE versus k on a log-log scale (see help loglog). Assign title to your figure (i.e. 20 data) and legends to your curves (i.e. test, train). Discuss what you observed from the figure.



(Refer to Appendix 12 for code)

```
% GRAPH OBSERVATIONS
% The following was observed from the figure:
%   - Initially MSE decreases
%   - As k approaches the number of training samples, MSE increases.
%   - When k >= number of training samples, MSE is constant
```

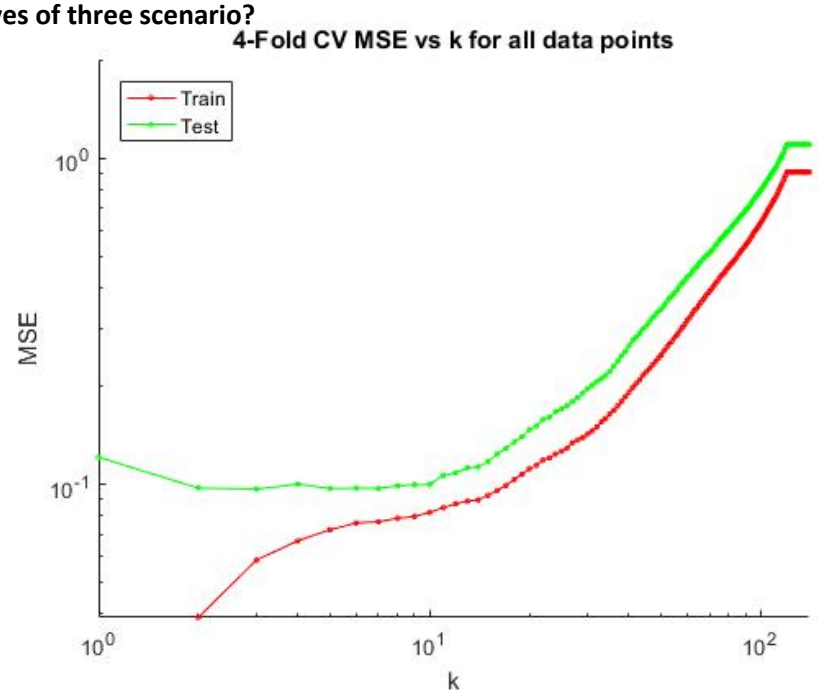
b) Repeat, but use all the training data. What happened? Contrast with your results from Problem 1 (hint: which direction is "complexity" in this picture?).

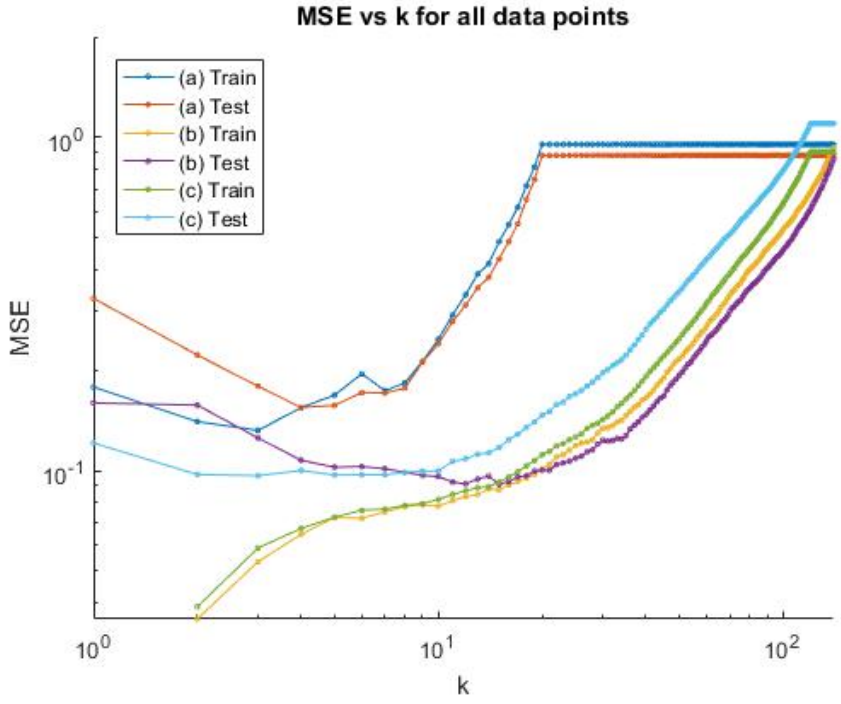


(Refer to Appendix 13 for code)

```
% GRAPH OBSERVATIONS
% The following differences between problem 1e and 3b were observed:
%   - In problem 1e, as the order of the model increased the complexity and
%   MSE of the model also increased. However, in problem 3b, as k increases
%   the complexity of the model decreases yielding a 'smoother' function.
```

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. Discuss why might we need to use this technique via comparing curves of three scenario?





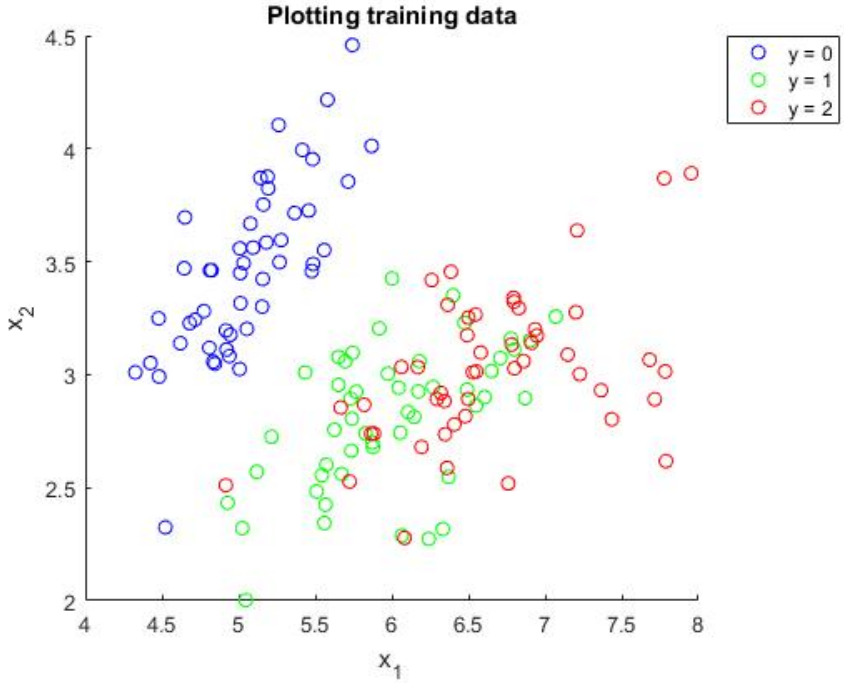
(Refer to Appendix 14 for code)

```
% GRAPH OBSERVATION
% 4-fold cross-validation yields similar accuracy to (b) while using less
% data. In part (b) 200 data samples (140 for training and 60 for testing)
% were used while 4-fold cross-validation use only 140. Thus, in cases
% where large amounts of data is not available, hold-out and
% cross-validation can be used to achieve accuracy with less data.
```

4. Nearest Neighbor Classifiers

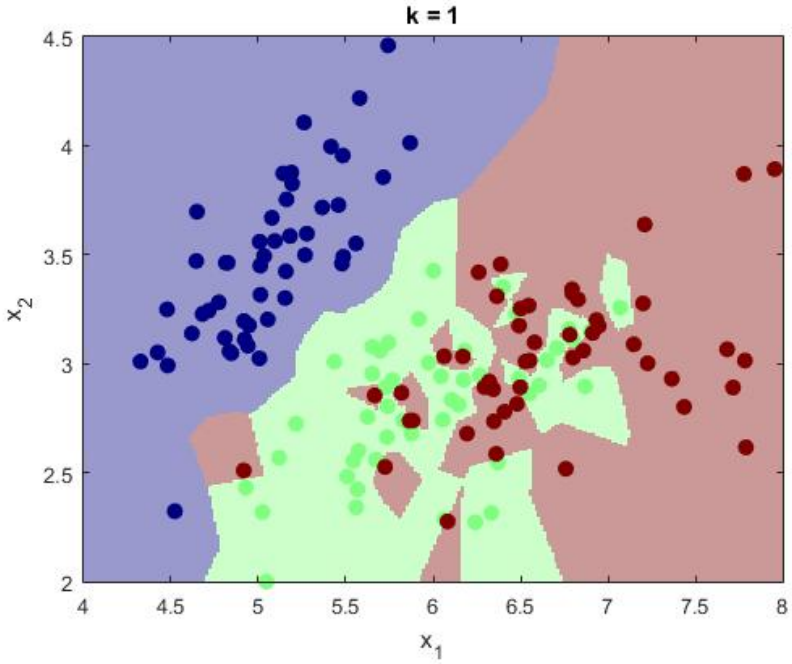
(Run setup code in Appendix 15)

a) Plot the data by their feature values, using the class value to select the color. The easiest way to do this is to use find to identify indices for which Y takes on each of its possible values. You can use unique to see what values Y contains. You can then plot each class in turn, using hold on to plot on top of the previous plot. (When you don't want this anymore, use hold off).



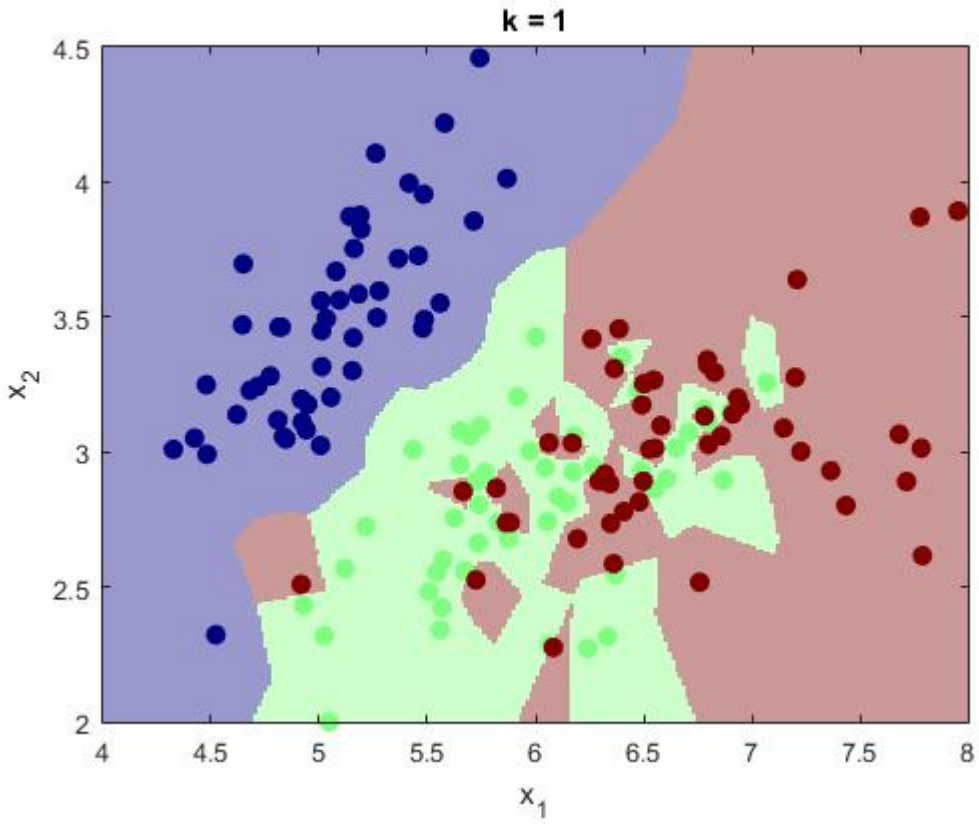
(Refer to Appendix 16 for code)

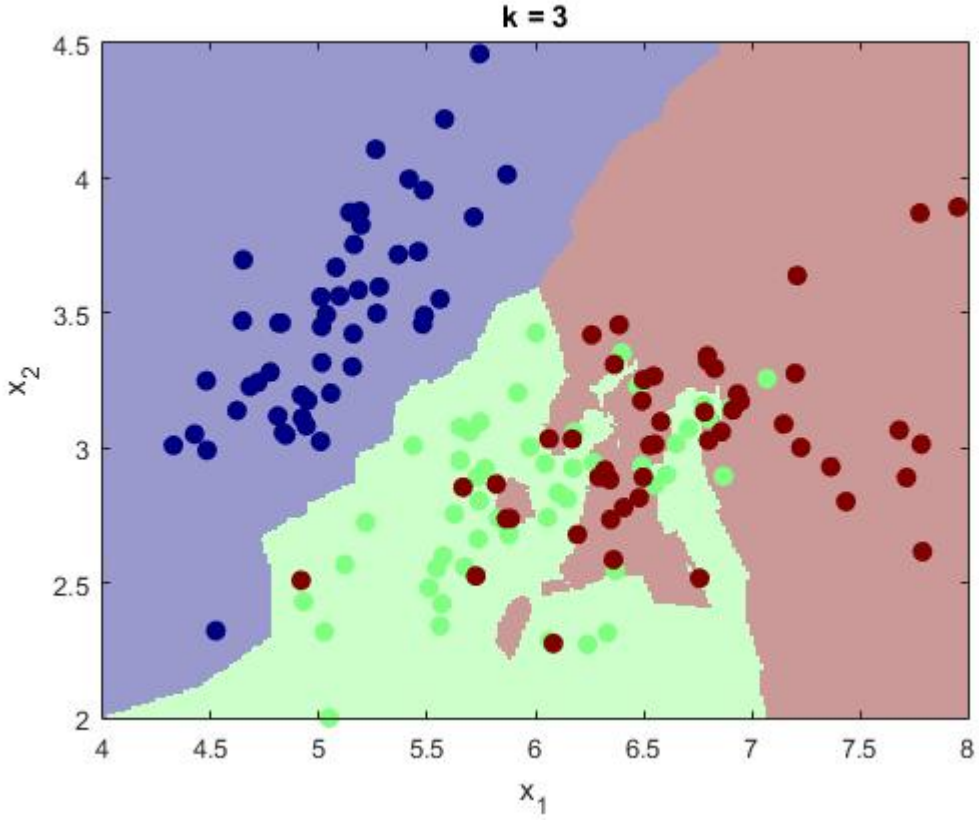
b) Use the provided knnClassify class to learn a 1-nearest-neighbor predictor. Use the function class2DPlot(learner, X, Y) to plot the decision regions and training data together.

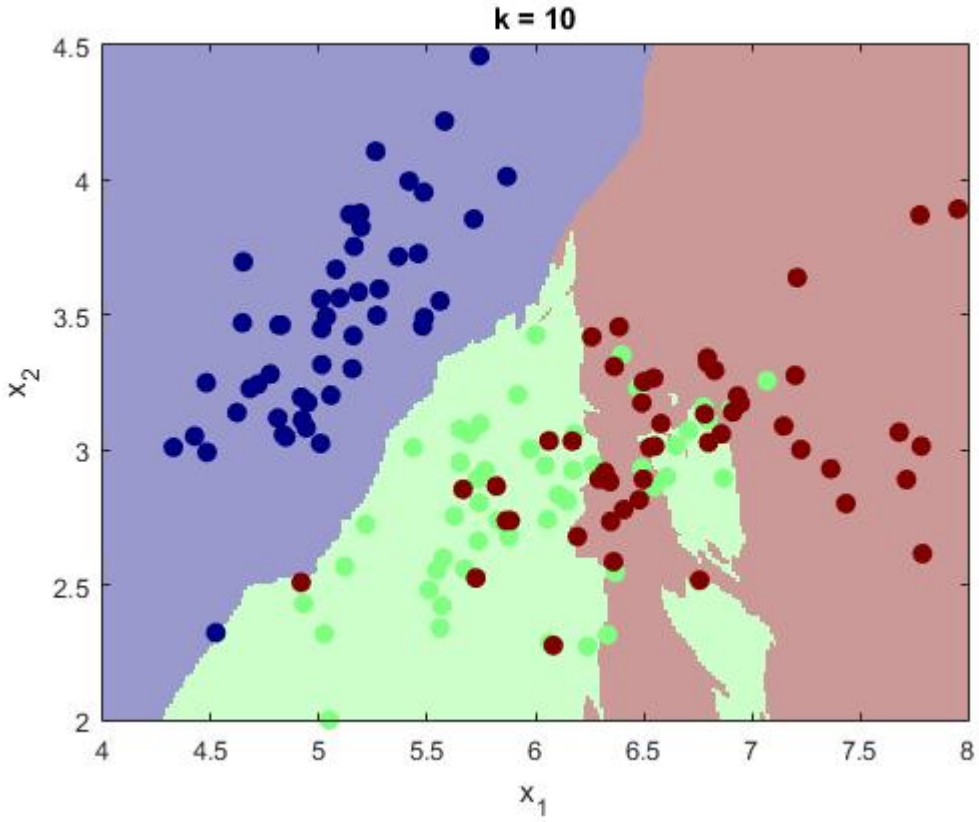


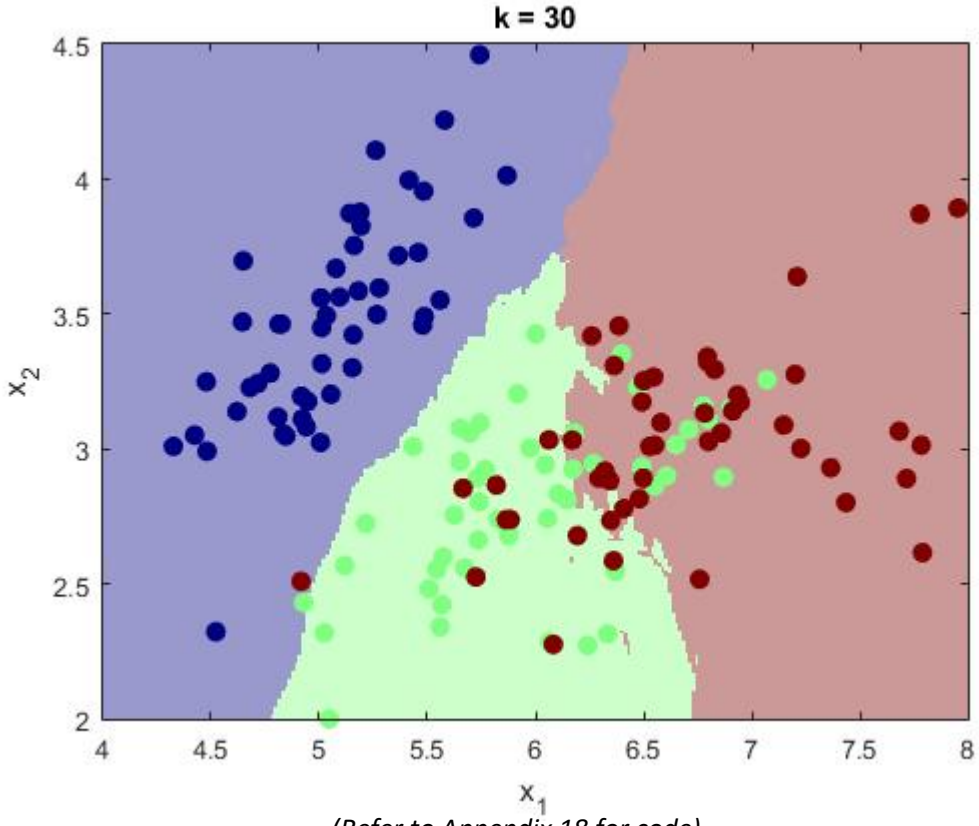
(Refer to Appendix 17 for code)

c) Do the same thing for several values of k (say, [1, 3, 10, 30]) and comment on their appearance.





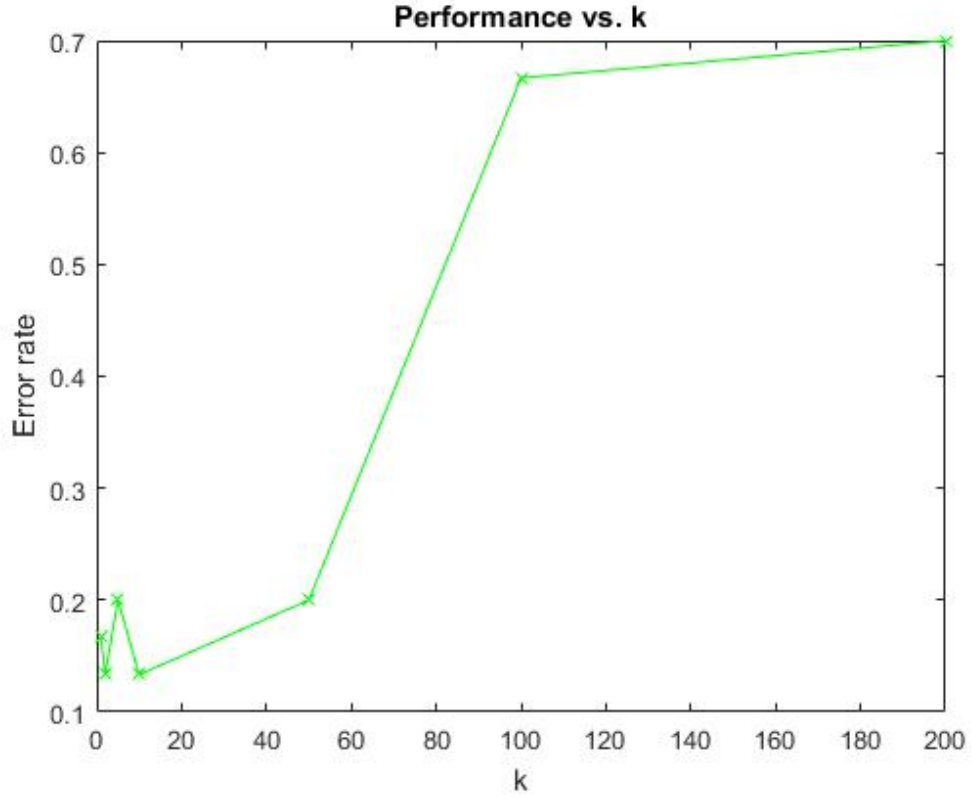




(Refer to Appendix 18 for code)

```
% COMMENT
%   The decision boundary becomes simpler for higher values of k
%   whereas lower values of k result in more complex decision boundaries.
```

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.



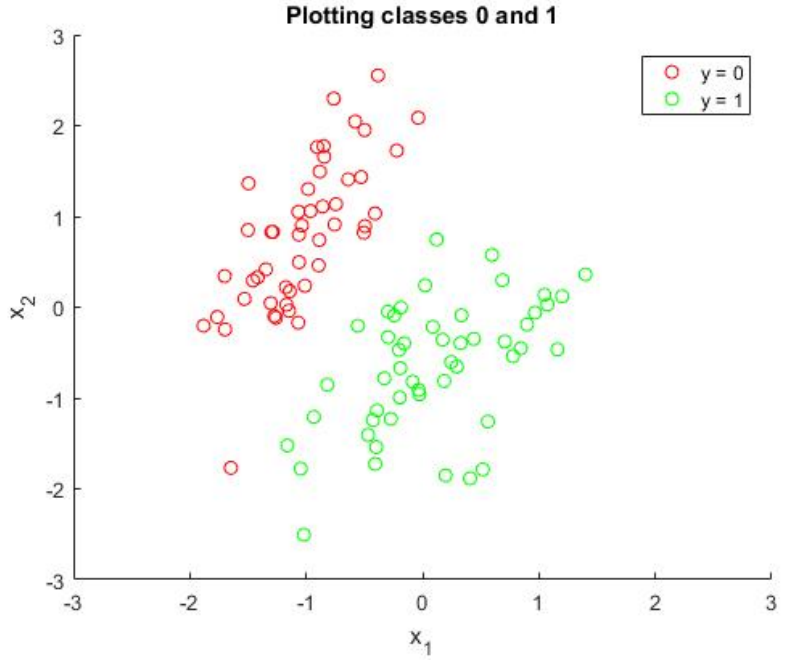
(Refer to Appendix 19 for code)

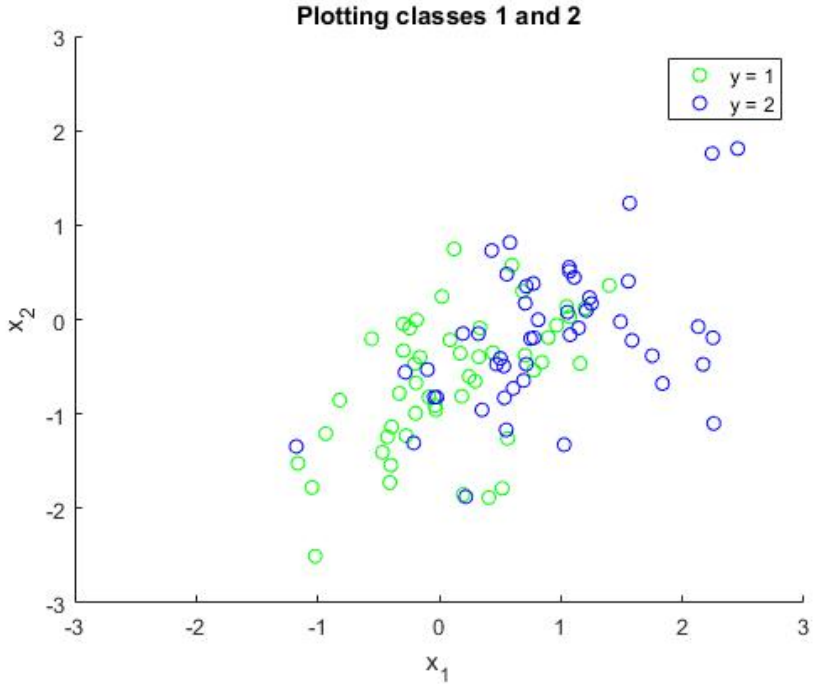
```
% COMMENTS
%   From the figure, it can be seen that k = 10:20 generalizes the best given
%   the training data. At the very left endpoint (k = 1), the model seems to
%   over-fit the data resulting a poor performance. For the very right
%   endpoint (k = 200), the model seems to under-fit the data and simply
%   predicts base on the majority class of the training data which also
%   results in a poorly performing model.
```

5. Perceptron and Logistic Regression

(Run setup code in Appendix 20)

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

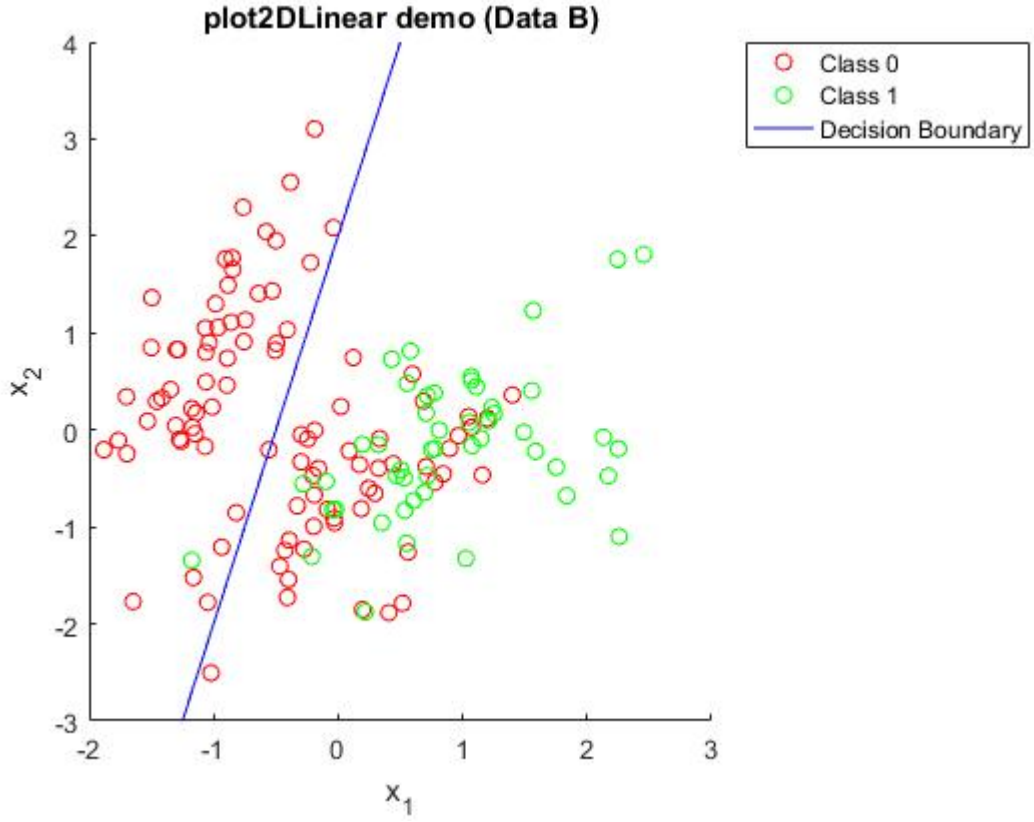




(Refer to Appendix 21 for code)

```
% COMMENT
%   The first graph shows linearly separable data while second shows linearly
%   inseparable data.
```

b) Write (fill in) the function @logisticClassify2/plot2DLinear.m so that it plots the two classes of data in different colours, along with the decision boundary (a line). Include the listing of your code in your report. To demo you function plot the decision boundary corresponding to the classifier $sign(.5 + 1x_1 - .25x_2)$ along with the A data, and again with the B data.



(Refer to Appendix 22a for completed plot2DLinear.m and 22b for graphing code)

c) Complete the predict.m function to make predictions for your linear classifier.

```
Error rate of model on data set A: 0.050505
Error rate of model on data set B: 0.46465
```

(Refer to Appendix 23a for completed predict.m function and 23b for error rate output code)

d) Derive the gradient of the regularized negative long likelihood for logistic regression.

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

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

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

Using chain rule to find:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\left(\sigma\left(\theta x^{(j)^T}\right)\right) &= \frac{\partial}{\partial\theta_i}\left(\frac{1}{1+e^{-\theta x^{(j)^T}}}\right)\\ &= \frac{\partial}{\partial\theta_i}\left(1+e^{-\theta x^{(j)^T}}\right)^{-1}\\ &= -\left(1+e^{-\theta x^{(j)^T}}\right)^{-2}\cdot\frac{\partial}{\partial\theta_i}\left(1+e^{-\theta x^{(j)^T}}\right)\\ &= -\left(1+e^{-\theta x^{(j)^T}}\right)^{-2}\cdot -x_i^{(j)}e^{-\theta x^{(j)^T}}\\ &= \left(1+e^{-\theta x^{(j)^T}}\right)^{-2}\cdot x_i^{(j)}e^{-\theta x^{(j)^T}}\\ &= \left(\frac{e^{-\theta x^{(j)^T}}}{1+e^{-\theta x^{(j)^T}}}\right)\left(\frac{1}{1+e^{-\theta x^{(j)^T}}}\right)x_i^{(j)}\\ &= \left(1-\frac{1}{1+e^{-\theta x^{(j)^T}}}\right)\left(\frac{1}{1+e^{-\theta x^{(j)^T}}}\right)x_i^{(j)}\\ &= \left(1-\sigma\left(\theta x^{(j)^T}\right)\right)\sigma\left(\theta x^{(j)^T}\right)x_i^{(j)}\end{aligned}$$

Using chain rule to find:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\left(-y^{(j)}\log\left(\sigma\left(\theta x^{(j)^T}\right)\right)\right) &= -y^{(j)}\cdot\frac{1}{\sigma\left(\theta x^{(j)^T}\right)}\cdot\frac{\partial}{\partial\theta_i}\left(\sigma\left(\theta x^{(j)^T}\right)\right)\\ &= -y^{(j)}\cdot\frac{1}{\sigma\left(\theta x^{(j)^T}\right)}\cdot\left(1-\sigma\left(\theta x^{(j)^T}\right)\right)\sigma\left(\theta x^{(j)^T}\right)x_i^{(j)}\\ &= -y^{(j)}x_i^{(j)}\cdot\left(1-\sigma\left(\theta x^{(j)^T}\right)\right)\end{aligned}$$

Using chain rule to find:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\left(-\left(1-y^{(j)}\right)\log(1-\sigma\left(\theta x^{(j)^T}\right)\right) &= -\left(1-y^{(j)}\right)\cdot\frac{1}{1-\sigma\left(\theta x^{(j)^T}\right)}\cdot\frac{\partial}{\partial\theta_i}\left(1-\sigma\left(\theta x^{(j)^T}\right)\right)\\ &= -\left(1-y^{(j)}\right)\cdot\frac{1}{1-\sigma\left(\theta x^{(j)^T}\right)}\left(\frac{\partial}{\partial\theta_i}(1)-\frac{\partial}{\partial\theta_i}\left(\sigma\left(\theta x^{(j)^T}\right)\right)\right)\\ &= -\left(1-y^{(j)}\right)\cdot\frac{1}{1-\sigma\left(\theta x^{(j)^T}\right)}\left(-\left(1-\sigma\left(\theta x^{(j)^T}\right)\right)\sigma\left(\theta x^{(j)^T}\right)x_i^{(j)}\right)\\ &= \left(1-y^{(j)}\right)\sigma\left(\theta x^{(j)^T}\right)x_i^{(j)}\end{aligned}$$

L2 norm can be derived trivial as follows:

$$\frac{\partial}{\partial\theta_i}\left(\alpha\sum_i\theta_i^2\right) = 2\alpha\sum_i\theta_i$$

∴ The gradient of the regularized negative log likelihood J_j is:

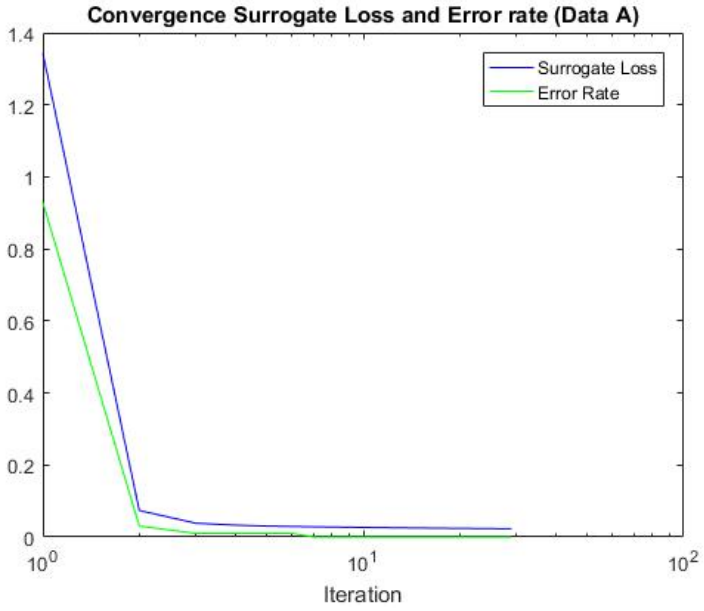
$$\frac{\partial J_j}{\partial\theta_i} = -y^{(j)}x_i^{(j)}\left(1-\sigma\left(\theta x^{(j)^T}\right)\right)+x_i^{(j)}\left(1-y^{(j)}\right)\sigma\left(\theta x^{(j)^T}\right)+2\alpha\sum_i\theta_i$$

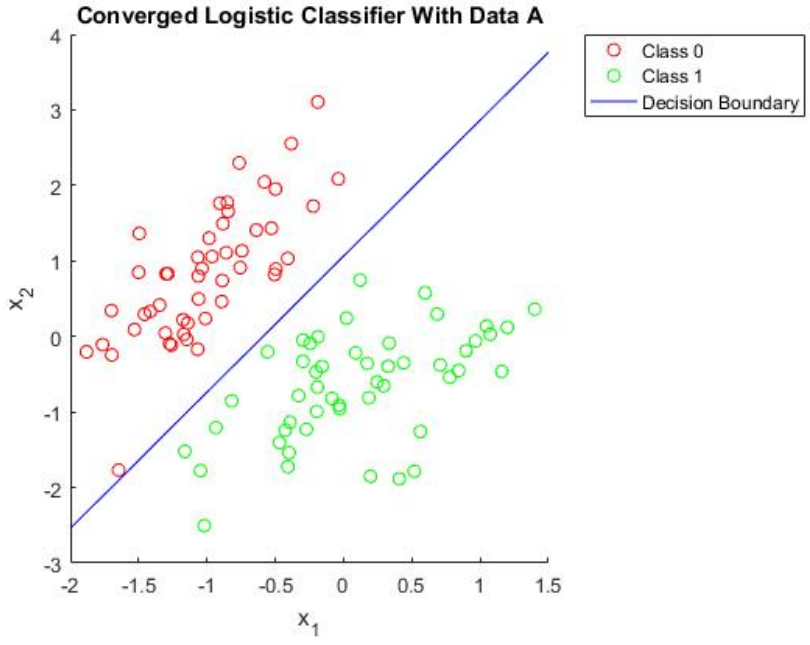
e) Complete your train.m function to perform stochastic gradient descent on the logistic loss function.

(Refer to Appendix 24 for code)

f) Run your logistic regression classifier on both data sets (A and B); for this problem, use no regularization (α = 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 (using e.g. plotClassify2D). In your report, please also include the functions that you wrote (at minimum, train.m, but possibly a few small helper functions as well).

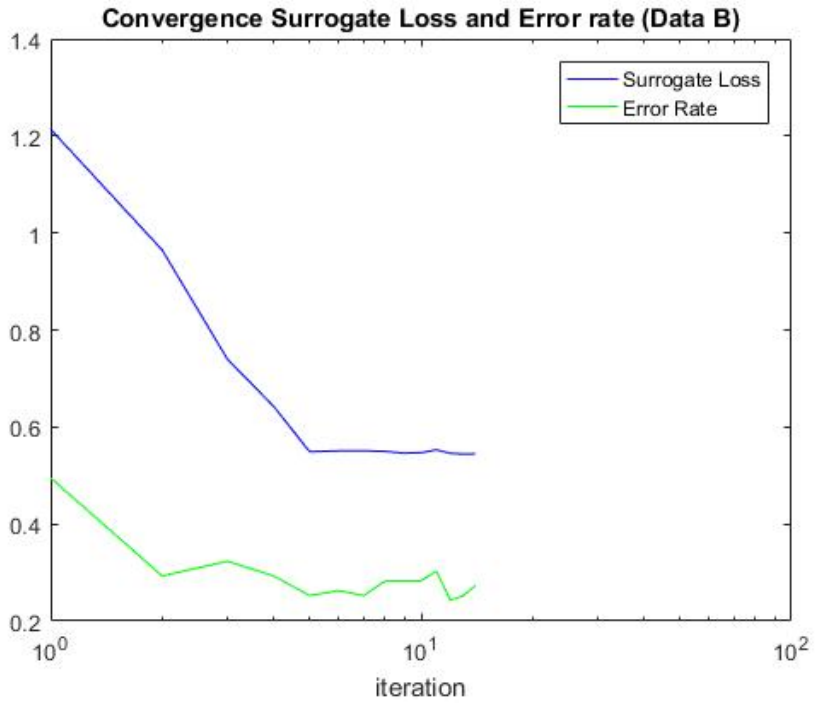
(Run setup code in Appendix 25)

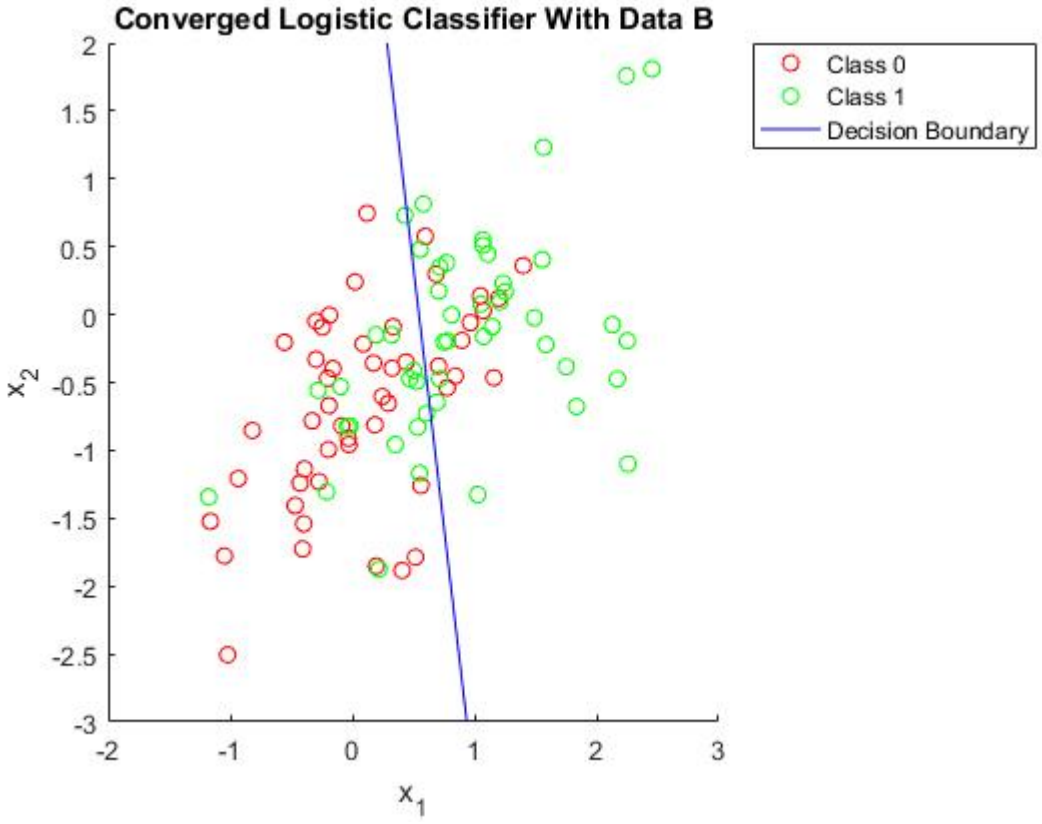





```
Tolerance reached. Training done. Took 29 iterations.
```

(Refer to Appendix 26a for code)

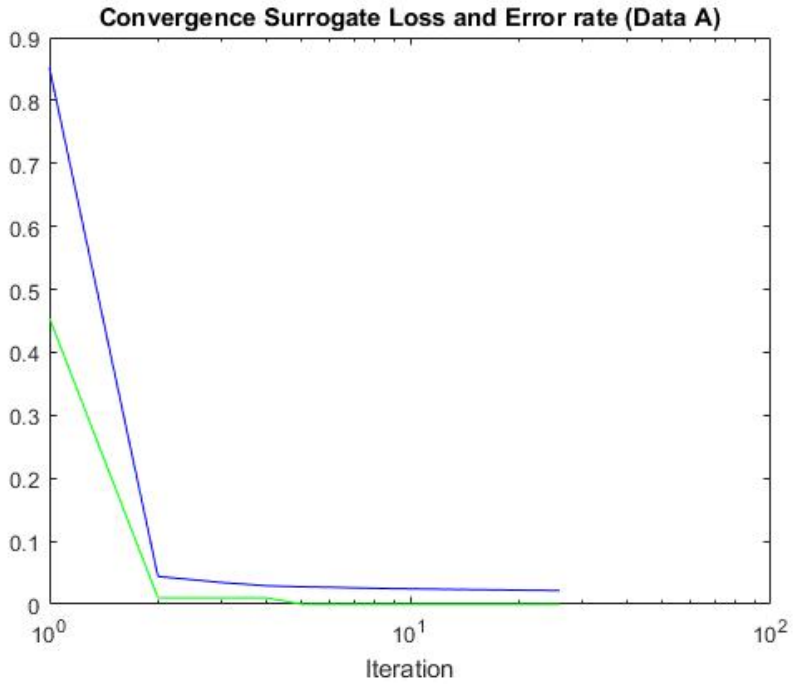


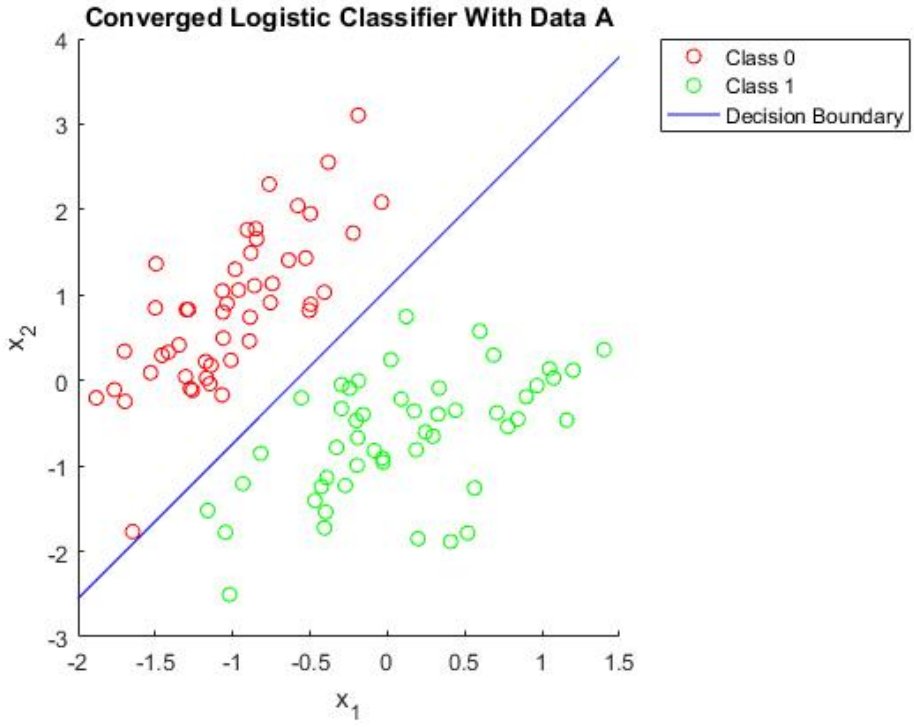



```
Tolerance reached. Training done. Took 14 iterations.
```

(Refer to Appendix 26b for code)

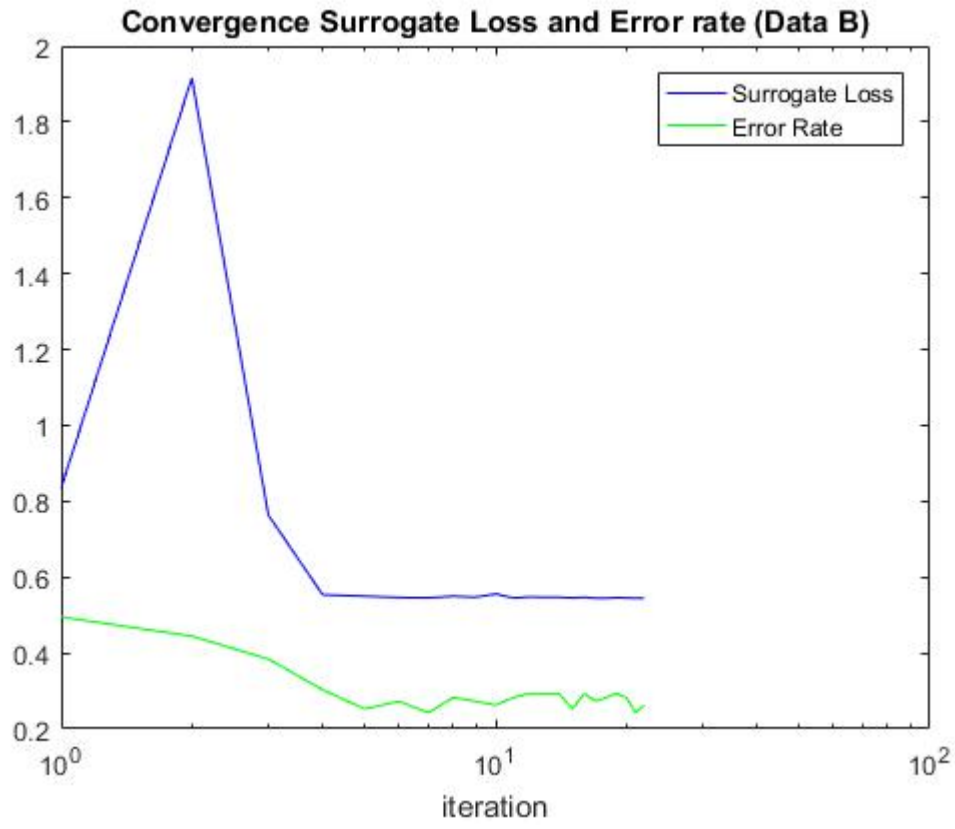
g) Implement the mini batch gradient descent on the logistic function by completing train_in_batches.m function.

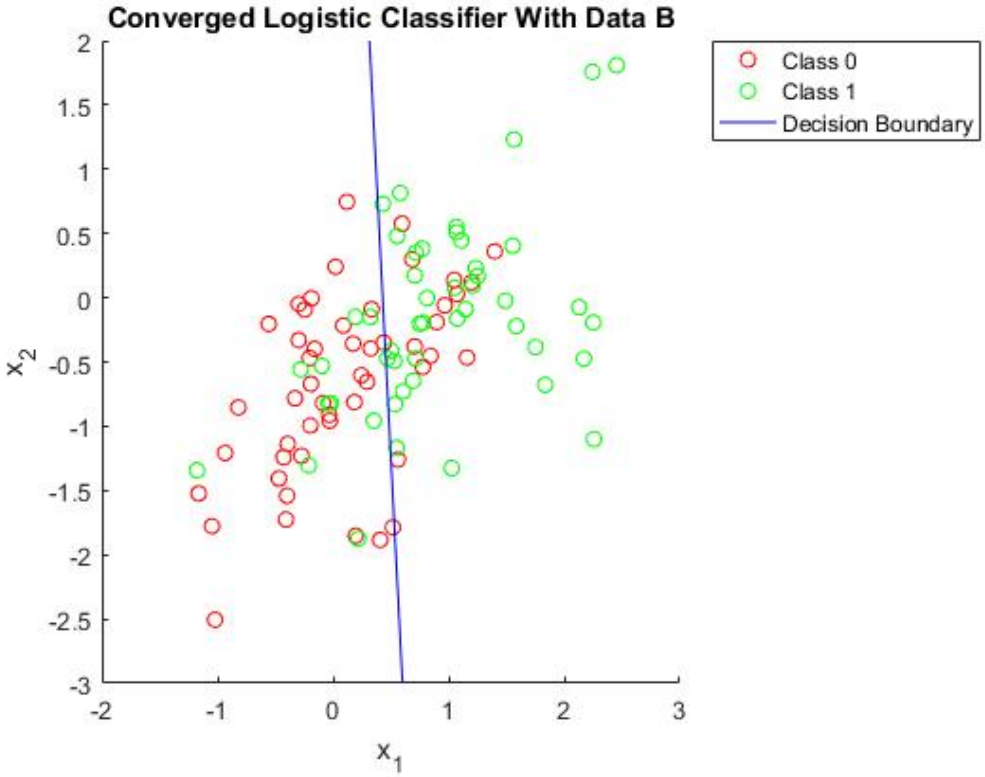





```
Tolerance reached. Training done. Took 26 iterations.
```

(Refer to Appendix 27a for create_mini_batches.m, Appendix 27b for train_in_batches.m, Appendix 27c for logisticClassify2.m, and Appendix 27d for the running code on Data A)






```
Tolerance reached. Training done. Took 22 iterations.
```

(Refer to Appendix 27e for the running code on Data B)

Appendices

The following code have been pasted from MATLAB, please refer to the source files for better formatting and code clarity.

Appendix 1: Setup

```
%% Setup
clear; clc; close all;

% Load training data
mTrain = load('data\mTrainData.txt');

% Split training data into features and labels (Xtr and Ytr)
Xtr = mTrain(:, 1);
Ytr = mTrain(:, 2);

% Load test data
mTest = load('data\mTestData.txt');

% Split data test data into features and labels (Xte and Yte)
Xte = mTest(:, 1);
Yte = mTest(:, 2);
```

Appendix 2: Plotting training data in a scatter plot

```
%% (a) Plot the training data in a scatter plot
figure;
plot(Xtr, Ytr,'bo');
title('Training Data And Linear Regression Learners');
legend('Data', 'Location', 'northeastoutside');
xlabel('x');
ylabel('y');
```

Appendix 3: Create linear regression learner

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

% Creating learner
learnerLinear = linearReg(polyx(Xtr,1), Ytr);

% Using learner to get predictions for 0:0.01:1
xline = 0:0.01:1;
yHatLinear = predict(learnerLinear, polyx(xline',1));

% Plotting
hold on
plot(xline, yHatLinear);
hold off
legend('Data', 'Linear learner');
```

Appendix 4: Higher-order Polynomials

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

orders = 3:2:13;
learners(size(orders)) = linearReg;
xline = 0:0.01:1;
yhat = zeros(6,length(xline));
for i = 1:length(learners)
    learners(i) = linearReg(polyx(Xtr, orders(i)), Ytr);
    yhat(i,:) = predict(learners(i), polyx(xline',orders(i)));
end

hold on
plot(xline, yhat);
hold off

xlabel('x');
ylabel('y');
legend('Training data', 'Linear learner', '3rd-Order Learner',...
    '5th-Order Learner', '7th-Order Learner', '9th-Order Learner',...
    '11th-Order Learner',  '13th-Order Learner', 'Location', 'northeastoutside');
```

Appendix 5: MSE

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

% changing orders to be 1:2:13
orders = 1:2:13;

% Add linear learner to the array of learners
learners = [learnerLinear, learners];

errorsTr = zeros(1,7);
for i = 1:7
    errorsTr(i) = mse(learners(i), polyx(Xtr, orders(i)), Ytr);
end

figure;
plot(orders, errorsTr, '-rx', 'MarkerSize', 10);
title('MSE vs Order of Polynomial');
xlabel('Learner Polynomial Order');
ylabel('Error');
legend('Training Data', 'Location', 'northeastoutside');
```

Appendix 6: MSE on test data

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

errorsTe = zeros(1,7);
for i = 1:7
    errorsTe(i) = mse(learners(i), polyx(Xte, orders(i)), Yte);
end

hold on
plot(orders, errorsTe, '-gx', 'MarkerSize', 10);
hold off

title('MSE vs Order of Polynomial');
xlabel('Learner Polynomial Order');
ylabel('Error');
legend('Training Data', 'Testing Data', 'Location', 'northeastoutside');
```

Appendix 7: MAE vs MSE

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

msError = zeros(1,7);
maError = zeros(1,7);

for i = 1:7
    msError(i) = mse(learners(i), polyx(Xte, orders(i)), Yte);
    maError(i) = mae(learners(i), polyx(Xte, orders(i)), Yte);
end

figure;
hold on
plot(orders, msError, '-rx', 'MarkerSize', 10);
plot(orders, maError, '-bx', 'MarkerSize', 10);
hold off

title('MSE vs MAE');
xlabel('Learner Polynomial Order');
ylabel('Error');
legend('MSE', 'MAE', 'Location', 'northeastoutside');
```

Appendix 8: Setup

```
%% Setup
clear; clc; close all;

% Load training data
mTrain = load('data\mTrainData.txt');

% Split training data into features and labels (Xtr and Ytr)
Xtr = mTrain(:, 1);
Ytr = mTrain(:, 2);

% Load test data
mTest = load('data\mTestData.txt');

% Split data test data into features and labels (Xte and Yte)
Xte = mTest(:, 1);
Yte = mTest(:, 2);
```

Appendix 9: Implementing code

```
% Test function: predict on Xtest
function Yte = predict(obj,Xte)
  [Ntr,Mtr] = size(obj.Xtrain);          % get size of training, test data
  [Nte,Mte] = size(Xte);
  classes = unique(obj.Ytrain);          % figure out how many classes & their labels
  Yte = repmat(obj.Ytrain(1), [Nte,1]);  % make Ytest the same data type as Ytrain
  K = min(obj.K, Ntr);                   % can't have more than Ntrain neighbors
  for i=1:Nte,                           % For each test example:
    dist = sum( bsxfun( @minus, obj.Xtrain, Xte(i,:) ).^2 , 2);     % compute sum of
squared differences
    %dist = sum( (obj.Xtrain - repmat(Xte(i,:),[Ntr,1]) ).^2 , 2);  % compute sum of
squared differences
    [tmp,idx] = sort(dist);              % find nearest neighbors over Xtrain (dimension
2)
                                         % idx(1) is the index of the nearest point,
etc.; see help sort
    % -- OUR CODE -- %
    indices = idx(1:K);
    kNearest = obj.Ytrain(indices);
    Yte(i)=  mean(kNearest);             % predict ith test example's value from nearest
neighbors
    % -- END OF OUR CODE -- %
  end;
end
```

Appendix 10: kNN for different k's

```
%% (b) Plotting kNN regression for k: 1,2,3,5,10,50

k = [1 2 3 5 10 50];
nnLearners(size(k)) = knnRegress;
xline = 0:0.01:1;
yhat = zeros(length(k),length(xline));
for i = 1:length(k)
    nnLearners(i) = knnRegress(k(i), Xtr, Ytr);
    yhat(i,:) = predict(nnLearners(i), xline');
end

figure
% Plot data
plot(Xtr, Ytr,'bo');

% Plot learners
hold on
plot(xline, yhat);
hold off

xlabel('x');
ylabel('y');
title('Training Data And kNN Regression Learners');
legend('Training data', 'k = 1', 'k = 2', 'k = 3', 'k = 5', 'k = 10', 'k = 50',
'Location', 'northeastoutside');
```

Appendix 11: Setup

```
%% Setup
clear; clc; close all;

% Load training data
mTrain = load('data\mTrainData.txt');

% Split training data into features and labels (Xtr and Ytr)
Xtr = mTrain(:, 1);
Ytr = mTrain(:, 2);

% Load test data
mTest = load('data\mTestData.txt');

% Split data test data into features and labels (Xte and Yte)
Xte = mTest(:, 1);
Yte = mTest(:, 2);
```

Appendix 12: MSE vs k

```
%% (a) Plot both train and test MSE versus k on a log-log scale

% First 20 training data examples
Xtr20 = Xtr(1:20);
Ytr20 = Ytr(1:20);

k = 1:140;
nnLearners(size(k)) = knnRegress;       % holds knnRegress learners
yhatTr = zeros(length(k),length(Xtr));  % holds predictions on training data
yhatTe = zeros(length(k),length(Xte));  % holds predictions on testing data
mseTr = zeros(size(k));                 % holds MSE of training data
mseTe = zeros(size(k));                 % hold MSE of testing data

for i = 1:length(k)
    % Training learner with k=i and first 20 training examples
    nnLearners(i) = knnRegress(k(i), Xtr20, Ytr20);

    % Using model to predict training x values
    yhatTr(i,:) = predict(nnLearners(i), Xtr);

    % Using model to predict testing x values
    yhatTe(i,:) = predict(nnLearners(i), Xte);

    % Calculating MSE of training and test data
    mseTr(i) = immse(yhatTr(i,:)', Ytr);
    mseTe(i) = immse(yhatTe(i,:)', Yte);
end

% Storing MSE's for later use in (c)
mseArrayTr = mseTr;
mseArrayTe = mseTe;

figure;
hold on
loglog(k, mseTr, '-ro', 'MarkerSize', 2);
loglog(k, mseTe, '-go', 'MarkerSize', 2);
hold off

% 'hold on' breaks 'loglog' so have to manually set the axis to log again
set(gca, 'XScale', 'log');
set(gca, 'YScale', 'log');

title('MSE vs k for 20 data points');
xlabel('k');
ylabel('MSE');
legend('Train', 'Test', 'Location', 'northwest');
axis([0 140 0 2]);
```

Appendix 13: MSE vs k with all training data

```
k = 1:140;
nnLearners(size(k)) = knnRegress;       % holds knnRegress learners
yhatTr = zeros(length(k),length(Xtr));  % holds predictions on training data
yhatTe = zeros(length(k),length(Xte));  % holds predictions on testing data
mseTr = zeros(size(k));                 % holds MSE of training data
mseTe = zeros(size(k));                 % hold MSE of testing data

for i = 1:length(k)
    % Training learner with k=i and all training examples
    nnLearners(i) = knnRegress(k(i), Xtr, Ytr);

    % Using model to predict training x values
    yhatTr(i,:) = predict(nnLearners(i), Xtr);

    % Using model to predict testing x values
    yhatTe(i,:) = predict(nnLearners(i), Xte);

    % Calculating MSE of training and test data
    mseTr(i) = immse(yhatTr(i,:)', Ytr);
    mseTe(i) = immse(yhatTe(i,:)', Yte);
end

% Storing MSE's for later use in (c)
mseArrayTr = [mseArrayTr; mseTr];
mseArrayTe = [mseArrayTe; mseTe];

figure;
hold on
loglog(k, mseTr, '-ro', 'MarkerSize', 2);
loglog(k, mseTe, '-go', 'MarkerSize', 2);
hold off

% 'hold on' breaks 'loglog' so have to manually set the axis to log again
set(gca, 'XScale', 'log');
set(gca, 'YScale', 'log');

title('MSE vs k for all data points');
xlabel('k');
ylabel('MSE');
legend('Train', 'Test', 'Location', 'northwest');
axis([0 140 0 2]);
```

Appendix 14: MSE vs k repeating three times

```
%% (c) 4-fold cross-validation
nCV = 1:4;          % cross-validatons
kValues = 1:140;    % k values

nnLearners(size(nCV)) = knnRegress; % Allocating space for learners

mseTr = zeros(length(kValues), length(nCV));    % All MSE of training data
mseTe = zeros(length(kValues), length(nCV));    % All MSE of testing data

for k = kValues
    for i = nCV
        idxS = (i-1)*20 + 1;            % Starting index of testing data
        idxE = idxS + 19;               % Ending index of testing data
        iTest = idxS:idxE;              % Indices for testing data
        iTrain = setdiff(1:140, iTest); % Indices for training data

        nnLearners(i) = knnRegress(k, Xtr(iTrain), Ytr(iTrain));

        % Validating learner
        yHatTr = predict(nnLearners(i), Xtr(iTrain));
        mseTr(k,i) = immse(yHatTr, Ytr(iTrain));

        yHatTe = predict(nnLearners(i), Xtr(iTest));
        mseTe(k,i) = immse(yHatTe, Ytr(iTest));
    end
end

% Averaging MSE's
mseAvgTr = mean(mseTr, 2);
mseAvgTe = mean(mseTe, 2);

% Storing MSE's for later use in (c)
mseArrayTr = [mseArrayTr; mseAvgTr'];
mseArrayTe = [mseArrayTe; mseAvgTe'];

figure;
hold on
loglog(kValues, mseAvgTr, '-ro', 'MarkerSize', 2);
loglog(kValues, mseAvgTe, '-go', 'MarkerSize', 2);
hold off

% 'hold on' breaks 'loglog' so have to manually set the axis to log again
set(gca, 'XScale', 'log');
set(gca, 'YScale', 'log');

title('4-Fold CV MSE vs k for all data points');
xlabel('k');
ylabel('MSE');
legend('Train', 'Test', 'Location', 'northwest');
axis([0 140 0 2]);

% Plotting (a), (b), and (c)
figure;
hold on
for i = 1:3
    loglog(kValues, mseArrayTr(i,:), '-o', 'MarkerSize', 2);
    loglog(kValues, mseArrayTe(i,:), '-o', 'MarkerSize', 2);
end
hold off
```

```
% 'hold on' breaks 'loglog' so have to manually set the axis to log again
set(gca, 'XScale', 'log');
set(gca, 'YScale', 'log');

title('MSE vs k for all data points');
xlabel('k');
ylabel('MSE');
legend('(a) Train', '(a) Test', '(b) Train', '(b) Test',...
    '(c) Train', '(c) Test', 'Location', 'northwest');
axis([0 140 0 2]);
```

Appendix 15: Setup

```
%% Setup
clear; clc; close all;

iris = load( 'data\iris.txt' );         % load data
pi = randperm( size (iris ,1));         % Random permuation of row indices
Y = iris(pi ,5); X = iris(pi ,1:2);     % Retrieve Y, X based on pi

uniqueY = unique(Y);
plotCfg = {'bo', 'go', 'ro'};   % Plot configs for classes
```

Appendix 16: Plotting by features

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

figure;
hold on;
for i = 1:length(uniqueY)
    mask = (Y == uniqueY(i));
    plot(X(mask, 1), X(mask, 2), plotCfg{uniqueY(i)+1},...
        'DisplayName', ['y = ', num2str(uniqueY(i))]);
end
legend('Location', 'northeastoutside');
legend show;
title('Plotting training data');
xlabel('x_1');
ylabel('x_2');
hold off;
```

Appendix 17: Decision boundaries

```
%% (b) Use the provided knnClassify class to learn a 1-nearest-neighbor predictor.

k1Learner = knnClassify(1, X, Y);
class2DPlot(k1Learner,X,Y);
title('k = 1');
xlabel('x_1');
ylabel('x_2');
```

Appendix 18: Complexity of decision boundaries

```
%% (c) Do the same thing for several values of k (say, [1, 3, 10, 30])
...and comment on their appearance.

kVals = [1 3 10 30];
kLearners(size(kVals)) = knnClassify;

for i = 1:length(kVals)
    kLearners(i) = knnClassify(kVals(i), X, Y);
    class2DPlot(kLearners(i), X, Y);
    title(['k = ' num2str(kVals(i))]);
    xlabel('x_1');
    ylabel('x_2');
end
```

Appendix 19: Performance vs k

```
%% (d) 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.

% Spliting data into 80/20 (train/test)
idx = round(size(iris, 1) * 0.2);

Xvalid = X(1:idx, :);
Yvalid = Y(1:idx, :);

Xtrain = X((idx+1):end, :);
Ytrain = Y((idx+1):end, :);

kVals = [1 2 5 10 50 100 200];
perf = zeros(size(kVals));      % Performance

for i = 1:length(kVals)
    kLearner =  knnClassify(kVals(i), Xtrain, Ytrain);
    Yhat = predict(kLearner, Xvalid);   % Predictions
    mask = (Yhat ~= Yvalid);            % Incorrect predictions mask
    perf(i) = sum(mask)/length(mask);   % Accuracy of learner
end

figure;
plot(kVals, perf, 'gx-');
title('Performance vs. k');
ylabel('Error rate');
xlabel('k');
```

Appendix 20: Setup

```
clear; clc; close all;

iris=load( 'data/iris.txt' );   % load the text file
X = iris(:,1:2); Y=iris(:,end); % get first two features
[X, Y] = shuffleData(X,Y);      % reorder randomly
X = rescale(X);                 % works much better for rescaled data
XA = X(Y<2,:); YA=Y(Y<2);       % get class 0 vs 1
XB = X(Y>0,:); YB=Y(Y>0);       % get class 1 vs 2
```

Appendix 21: Separable and inseparable data

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

% Plotting class 0 vs 1
figure('Name', 'Linearly Separable');
hold on;
title('Plotting classes 0 and 1');
plot(XA(YA==0,1), XA(YA==0,2), 'ro', 'DisplayName', 'y = 0');
plot(XA(YA==1,1), XA(YA==1,2), 'go', 'DisplayName', 'y = 1');
axis([-3 3, -3, 3]);
xlabel('x_1');
ylabel('x_2');
legend show;
hold off;


% Plotting class 1 vs 2
figure('Name', 'Linearly Inseparable');
title('Plotting classes 1 and 2');
hold on;
plot(XB(YB==1,1), XB(YB==1,2), 'go', 'DisplayName', 'y = 1');
plot(XB(YB==2,1), XB(YB==2,2), 'bo', 'DisplayName', 'y = 2');
axis([-3 3, -3, 3]);
xlabel('x_1');
ylabel('x_2');
legend show;
hold off;
```

Appendix 22a: Completing plot2DLinear

```
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;

    %%% TODO: Fill in the rest of this function...

    classes = unique(Y);        % Get classes from Y
    iY1 = (Y == classes(1));    % Get mask of Y == first class
    iY2 = (Y == classes(2));    % Get mask of Y == second class

    % Plotting data
    hold on;
    plot(X(iY1, 1), X(iY1, 2), 'ro');
    plot(X(iY2, 1), X(iY2, 2), 'go');

    % Plotting decision boundary
    wts = obj.wts;
    f = @(x1,x2) wts(1) + wts(2)*x1 + wts(3)*x2;
    fimplicit(f, axis, 'b');

    % Plot config
    xlabel('x_1');
    ylabel('x_2');
    legend('Class 0', 'Class 1', 'Decision Boundary',...
        'Location', 'northeastoutside'...
    );
    hold off;
end
```

Appendix 22b: Plotting

```
%% (b) Write ( fill in) the functio n @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.

learner = logisticClassify2(); % create "blank" learner
wts = [0.5 1 -0.25];
learner = setWeights(learner, wts); % set the learner's parameters

learner = setClasses(learner, unique(YA)); % define class labels using YA
plot2DLinear(learner, XA, YA);
title('plot2DLinear demo (Data A)');

learner = setClasses(learner, unique(YB)); % define class labels using YB
plot2DLinear(learner, XB, YB);
title('plot2DLinear demo (Data B)');
```

Appendix 23a: Completing predict.m

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

    % -- OUR CODE -- %
    % (1) make predictions based on the sign of wts(1) + wts(2)*x(:,1) + ...
    wts = obj.wts;
    f = wts(1) + wts(2)*Xte(:,1) + wts(3)*Xte(:,2);
    fSign = sign(f);

    % (2) convert predictions to saved classes: Yte = obj.classes( [1 or 2] );
    yh = zeros(size(Xte,1), 1);

    negMask = fSign < 0;
    posMask = fSign > 0;

    yh(negMask) = obj.classes(1);
    yh(posMask) = obj.classes(2);

    Yte = yh;
    % -- END OF OUR CODE -- %
end
```

Appendix 23b: Calculating Errors

```
%% (c) Complete the predict.m function to make predictions for your
...linear classifier.

learner = logisticClassify2();                  % create "blank" learner
learner = setWeights(learner, [0.5 1 -0.25]);   % set the learner's parameters

% Calculating error in data set A
learner = setClasses(learner, unique(YA));      % define class labels using YA
YteA = predict(learner, XA);                    % Use learner to predict XA

wrMaskA = (YteA ~= YA); % Wrong Mask
errA = sum(wrMaskA)/length(wrMaskA);
disp(['Error rate of model on data set A: ', num2str(errA)]);

% Calculating error in data set B
learner = setClasses(learner, unique(YB));      % define class labels using YB
YteB = predict(learner, XB);                    % Use learner to predict XB

wrMaskB = (YteB ~= YB); % Wrong Mask
errB = sum(wrMaskB)/length(wrMaskB);
disp(['Error rate of model on data set B: ', num2str(errB)]);
```

Appendix 24: Completing train.m

```
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

  % -- OUR CODE -- %
  % computing surrogate (neg log likelihood) loss
  costY0 = -Y.*log(logistic(obj, X));       % 0 When Y = 0
  costY1 = -(1-Y).*log(1-logistic(obj,X));  % 0 When Y = 1
  regTerm = reg * sum(obj.wts.^2);          % Regularisation term
  Jsur(iter) = (1/n)*sum(costY0 + costY1 + regTerm);    % Computing cost
  %-- END OF OUR CODE -- %

  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-');
  legend('Surrogate lost', 'Error rate');
  drawnow;



  for j=1:n,
    % -- OUR CODE -- %
    % Compute linear responses and activation for data point j
    LR = logistic(obj, X(j, :));    % Response for data point j

    % Compute gradient:
    Y0 = -Y(j).*X1(j,:).*(1-LR);         % 0 When Y = 0
    Y1 = ((1-Y(j)).*X1(j,:)).*LR;        % 0 When Y = 1
    regTerm = 2*reg*sum(obj.wts);        % Regularized term

    grad = Y0 + Y1 + regTerm;            % Gradient
    % -- END OF OUR CODE -- %

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

  % -- OUR CODE -- %
  if(iter > 1 && abs(Jsur(iter) - Jsur(iter-1)) < stopTol)
    done = true;
    disp(['Tolerance reached. Training done. Took ' num2str(iter) ' iterations.']);
  end

  if(iter > stopIter)
    done = true;
    disp('Max iter reached. Training done');
  end
  % -- END OF OUR CODE -- %

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

Appendix 25: Setup

```
%% (f) Setup
clear; clc; close all;
iris=load( 'data/iris.txt' );   % load the text file
X = iris(:,1:2); Y=iris(:,end); % get first two features
[X, Y] = shuffleData(X,Y);      % reorder randomly
X = rescale(X);                 % works much better for rescaled data
XA = X(Y<2,:); YA=Y(Y<2);       % get class 0 vs 1
XB = X(Y>0,:); YB=Y(Y>0);       % get class 1 vs 2
```

Appendix 26a: Data A

```
%% (f1) Run logistic regression classifier on data set A
close all; clc;

learnerA = logisticClassify2(XA, YA,...        % Train learner on data set A
    'reg', 0,...                % no regularization (alpha = 0)
    'stepsize', 1,...            % Step size = 1
    'stoptol', 0.0001,...        % Stop tolerance = 0.0001
    'plot', false...            % Plot true
);

title('Convergence Surrogate Loss and Error rate (Data A)');
xlabel('Iteration');
legend('Surrogate Loss', 'Error Rate');

% Plot final converged classifier
figure;
plot2DLinear(learnerA, XA, YA);
title('Converged Logistic Classifier With Data A');
```

Appendix 26b: Data B

```
%% (f2) Run logistic regression classifier on data set B
close all; clc;

learnerB = logisticClassify2(XB, YB,...        % Train learner on data set B
    'reg', 0,...                % no regularization (alpha = 0)
    'stepsize', 1,...            % Step size = 1
    'stoptol', 0.0001,...        % Stop tolerance = 0.0001
    'plot', false...            % Plot true
);

title('Convergence Surrogate Loss and Error rate (Data B)');
xlabel('iteration');
legend('Surrogate Loss', 'Error Rate');

figure;
plot2DLinear(learnerB, XB, YB);
title('Converged Logistic Classifier With Data B');
```

Appendix 27a: Completed create_mini_batches.m

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

n = length(y);

data_values = [X,y];
rIdx = randperm(n);

data_values = data_values(rIdx, :);
n_mini_batches = n/batch_size;
mini_batches = zeros(batch_size,3,n_mini_batches);

for i = 1:n_mini_batches
    bStart = (i-1)*batch_size + 1;
    bEnd = bStart + batch_size - 1;

    mini_batches(:, :, i) = data_values(bStart:bEnd, :);
end

end
```

Appendix 27b: Completed train_in_batches.m

```
function obj = train_in_batches(obj, X, Y, batch_size, 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

  % -- OUR CODE -- %
  % computing surrogate (neg log likelihood) loss
  costY0 = -Y.*log(logistic(obj, X));       % 0 When Y = 0
  costY1 = -(1-Y).*log(1-logistic(obj,X));  % 0 When Y = 1
  regTerm = reg * sum(obj.wts.^2);          % Regularisation term
  Jsur(iter) = (1/n)*sum(costY0 + costY1 + regTerm);    % Computing cost
  %-- END OF OUR CODE -- %

  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;

  % -- OUR CODE -- %
  % Batch size = 11 is passed in from logisticClassify2.m
  mini_batches = create_mini_batches(obj, X,Y, batch_size);
  number_of_batches = n/batch_size;
  % -- END OF OUR CODE -- %

  for j=1:number_of_batches,
    % -- OUR CODE -- %
    % Split batch into XX and YY
    XX = mini_batches(:, 1:2,j);
    XX1 = [ones(batch_size,1), XX];
    YY = mini_batches(:, 3, j);

    % Compute linear responses and activation for minibatch j
    LR = logistic(obj, XX);

    % Compute gradient:
    Y0 = -YY.*XX1.*(1-LR);              % 0 When Y = 0
    Y1 = (1-YY).*XX1.*LR;               % 0 When Y = 1
```

```
    regTerm = 2*reg*sum(obj.wts);       % Regularized term

    grad = sum(Y0 + Y1 + regTerm);      % Gradient
    % -- END OF OUR CODE -- %

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

  % -- OUR CODE -- %
  if(iter > 1 && abs(Jsur(iter) - Jsur(iter-1)) < stopTol)
    done = true;
    disp(['Tolerance reached. Training done. Took ' num2str(iter) ' iterations.']);
  end

  if(iter > stopIter)
    done = true;
    disp('Max iter reached. Training done');
  end
  % -- END OF OUR CODE -- %

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

Appendix 27c: Completed logisticClassify2.m

```
function obj = logisticClassify2(Xtr,Ytr, varargin)
% logisticClassify(X,Y,...) : construct a logistic classifier (linear classifier with
saturated output)
% can take no arguments, or see logisticClassify/train for training options

  obj.wts=[];         % linear weights on features (1st weight is constant term)
  obj.classes=[];     % list of class values used in input
  obj=class(obj,'logisticClassify2');
  if (nargin > 0)
    % -- OUR CODE -- %
    % Train using single data point j
    % obj=train(obj,Xtr,Ytr, varargin{:});

    % Train using mini batches sized 11
    obj = train_in_batches(obj, Xtr, Ytr, 11, varargin{:});
    % -- END OF OUR CODE -- %
  end;
end
```

Appendix 27d: Running logistic regression classifier (batch) on Data A

```
%% (g1) Run logistic regression classifier on data set A (batch version)
close all; clc;

learnerA = logisticClassify2(XA, YA,...     % Train learner on data set A
    'reg', 0,...                % no regularization (alpha = 0)
    'stepsize', 1,...           % Step size = 1
    'stoptol', 0.0001,...       % Stop tolerance = 0.0001
    'plot', false...            % Plot true
);

title('Convergence Surrogate Loss and Error rate (Data A)');
xlabel('Iteration');

% Plot final converged classifier
figure;
plot2DLinear(learnerA, XA, YA);
title('Converged Logistic Classifier With Data A');
```

Appendix 27e: Running logistic regression classifier (batch) on Data B

```
%% (g2) Run logistic regression classifier on data set B (batch version)
close all; clc;

learnerB = logisticClassify2(XB, YB,...     % Train learner on data set A
    'reg', 0,...                % no regularization (alpha = 0)
    'stepsize', 1,...           % Step size = 1
    'stoptol', 0.0001,...       % Stop tolerance = 0.0001
    'plot', false...            % Plot true
);

title('Convergence Surrogate Loss and Error rate (Data B)');
xlabel('Iteration');

% Plot final converged classifier
figure;
plot2DLinear(learnerB, XB, YB);
title('Converged Logistic Classifier With Data B');
```