2018 - CAB420 - Machine Learning

Group Assignment 1 Alex Wilson and Christopher Ayling

Contents

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

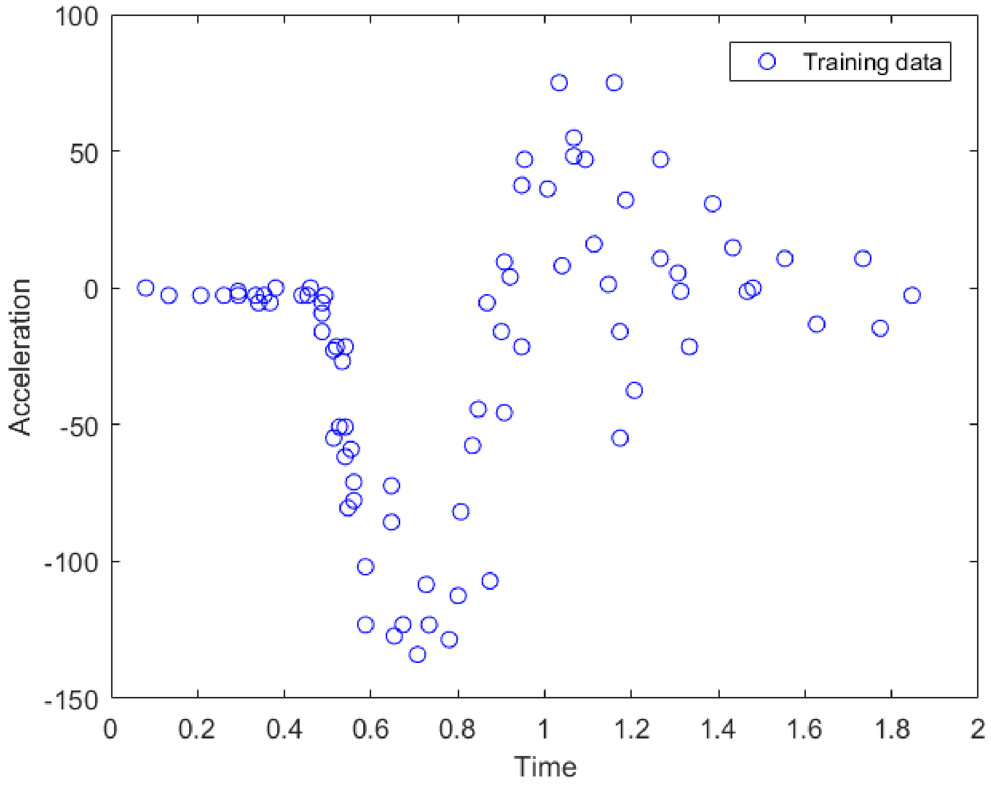
(c) From these results, show that $L(w)$ is a convex function.

Function $L(\mathbf{w})$ is convex because its Hessian H_L is PSD. H_L can be shown to satisfy the criteria $\mathbf{a}^T\mathbf{H}\mathbf{a} \geq 0$ because the output of all functions in the Hessian is always positive.

1. Feature, Classes and Linear Regression

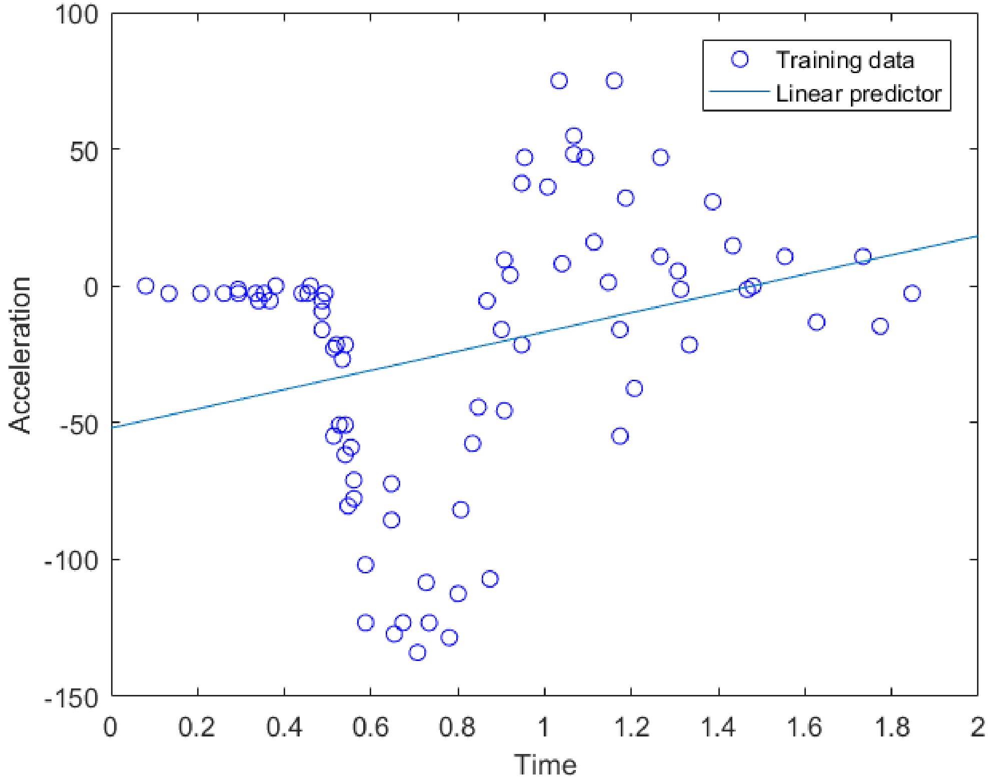
(a) Plot the training data in a scatter plot.

```
%Clean up
clc
clear
close all
%Load in and set up variables
mTrain = load('data/mcycleTrain.txt');
ytr = mTrain(:,1);
xtr = mTrain(:,2);
%Plot the training data
plot(xtr, ytr, 'bo');
hold on;
legend('Training data');
xlabel('Time');
ylabel('Acceleration');
```



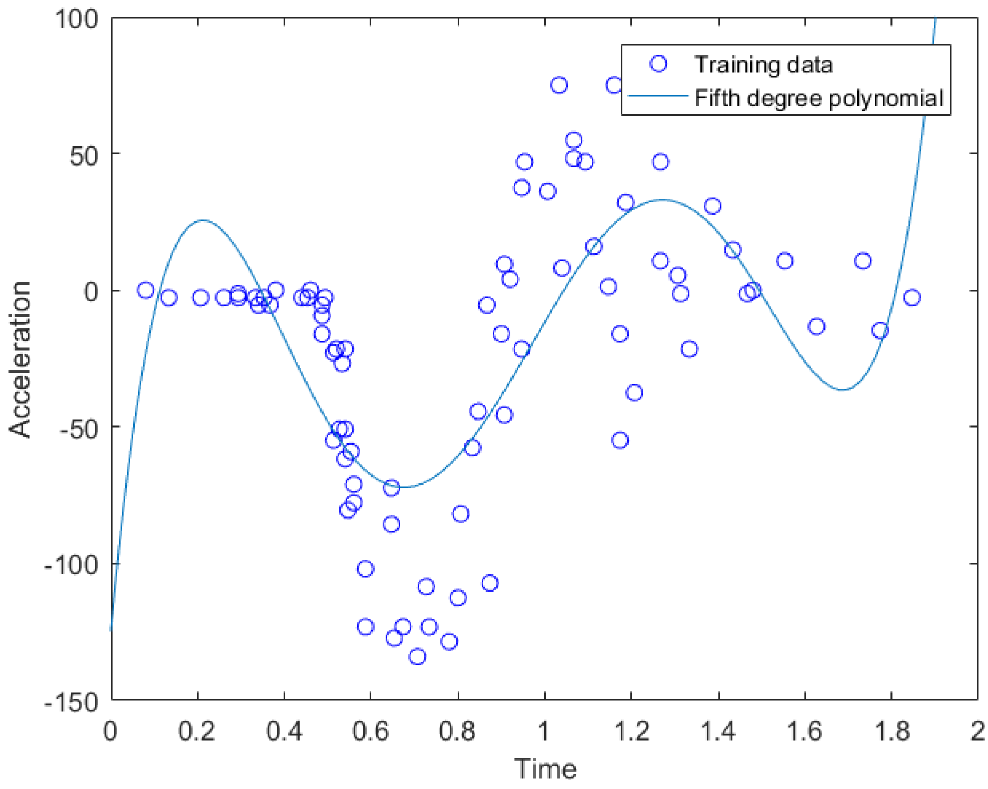
(b) Create a linear predictor (slope and intercept) using the above functions. Plot it on the same plot as the training data.

```
linXtr = polyx(xtr, 1);
linLearner = linearReg(linXtr, ytr);
xline = [0:.01:2]';
yline = predict(linLearner, polyx(xline, 1));
plot(xline, yline);
legend('Training data', 'Linear predictor');
xlabel('Time');
ylabel('Acceleration');
```



(c) Create another plot with the data and a fifth-degree polynomial

```
figure;
plot(xtr, ytr, 'bo');
hold on;
fifthXtr = polyx(xtr, 5);
fifthLearner = linearReg(fifthXtr, ytr);
xline = [0:.01:2]';
yline = predict(fifthLearner, polyx(xline, 5));
plot(xline, yline);
legend('Training data', 'Fifth degree polynomial');
xlabel('Time');
ylabel('Acceleration');
axis([0 2 -150 100]);
```



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

```
training_MSE_of_linear_predictor = mse(linLearner, linXtr, ytr)
training_MSE_of_fifth_degree_predictor = mse(fifthLearner, fifthXtr, ytr)
```

```
training_MSE_of_linear_predictor =
   2.2708e+03
training_MSE_of_fifth_degree_predictor =
   1.2546e+03
```

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

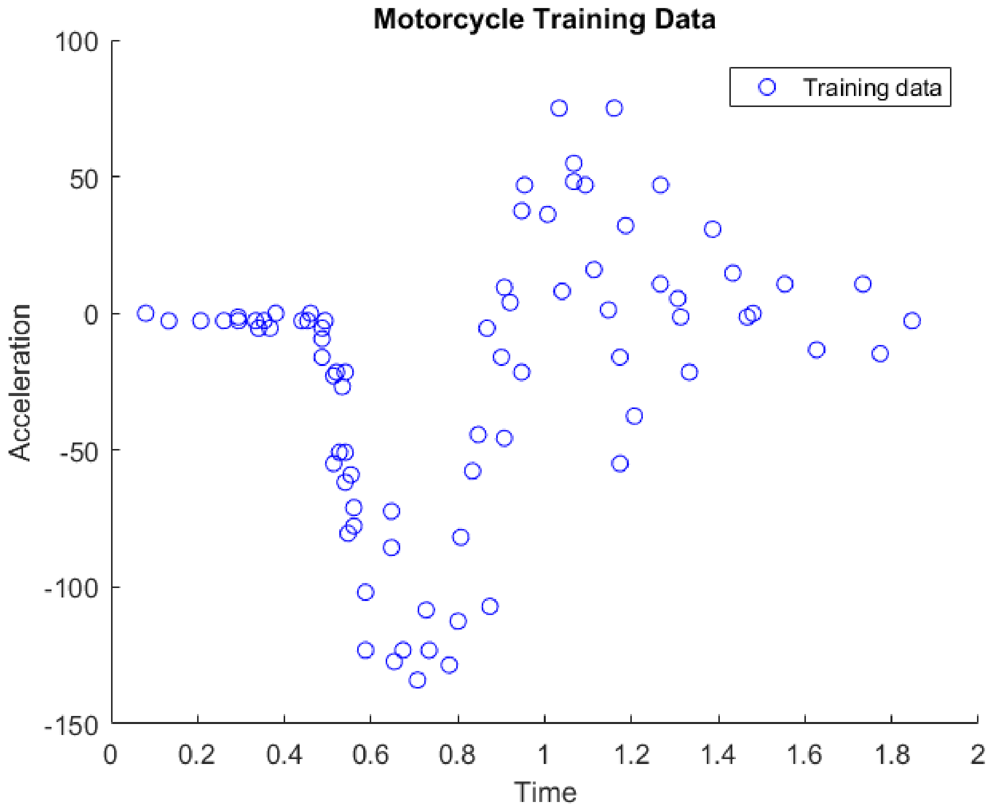
```
mTest = load('data/mcycleTest.txt');
ytst = mTest(:,1);
xtst = mTest(:,2);
linXtst = polyx(xtst, 1);
fifthXtst = polyx(xtst, 5);

test_MSE_of_linear_predictor = mse(linLearner, linXtst, ytst)
test_MSE_of_fifth_degree_predictor = mse(fifthLearner, fifthXtst, ytst)
```

```
test_MSE_of_linear_predictor =
   1.7049e+03
test_MSE_of_fifth_degree_predictor =
  999.6535
```

2. kNN Regression

```
clc
clear
```

```
% Load train data
mTrain = load('mcycleTrain.txt');
Ytrain = mTrain(:,1);
Xtrain = mTrain(:,2);
% Load test data
mTest = load('mcycleTest.txt');
Ytest = mTest(:,1);
Xtest = mTest(:,2);
% Plot Training Data
figure('name', 'Motorcycle Data');
hold on
plot(Xtrain, Ytrain, 'bo');
title('Motorcycle Training Data');
xlabel('Time');
ylabel('Acceleration');
legend('Training data');
hold off
```

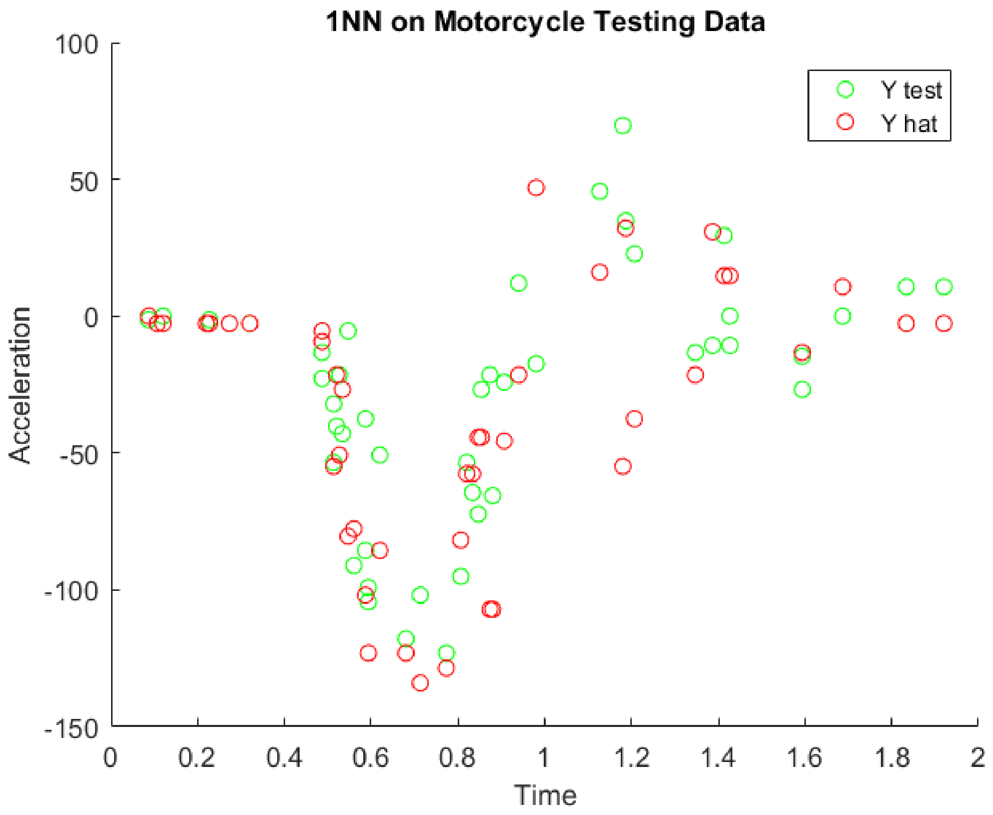


(a) Using the knnRegress class, implement (add code to) the predict function to make it functional.

```
% Test function: predict on Xtest
function Yte = predict(obj,Xte)
  [Ntr,Mtr] = size(obj.Xtrain);
  [Nte,Mte] = size(Xte);
  classes = unique(obj.Ytrain);
  Yte = repmat(obj.Ytrain(1), [Nte,1]);
  K = min(obj.K, Ntr);
  for i=1:Nte
    dist = sum( bsxfun( @minus, obj.Xtrain, Xte(i,:) ).^2 , 2);
    [tmp,idx] = sort(dist);

    % Our code here
    kclosest_vals = [];
    for j=1:K
        kclosest_vals = [kclosest_vals, obj.Ytrain(idx(j))];
    end
```

```
        Yte(i) = sum(kclosest_vals)/K;
      end;
    end
```

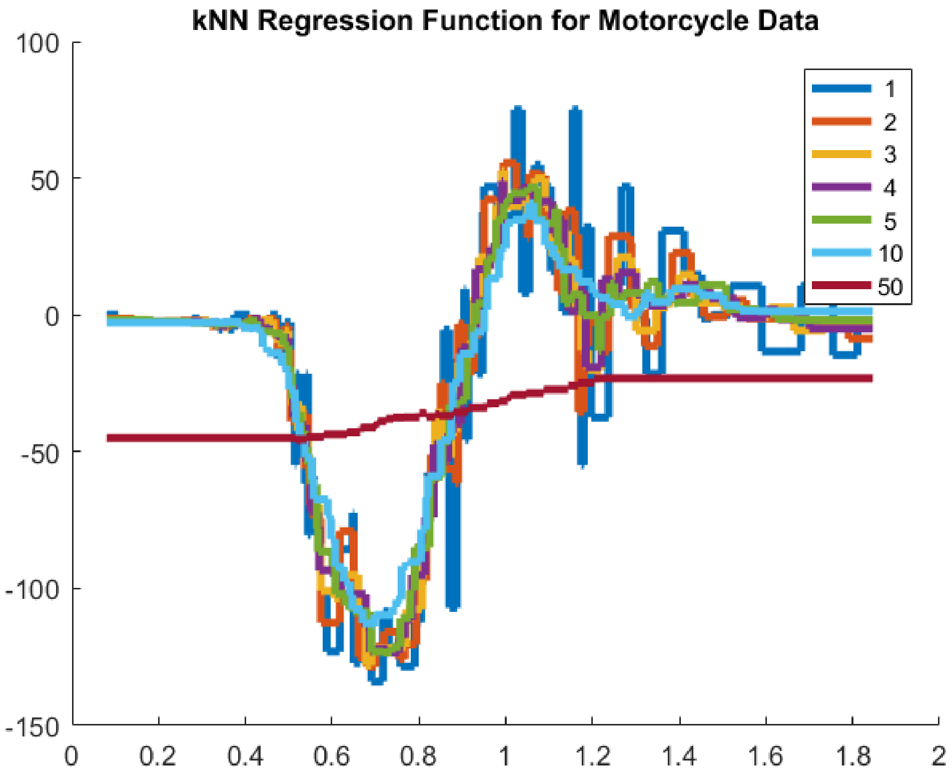
```
learner = knnRegress(1,Xtrain, Ytrain);
Yhat = predict(learner, Xtest);
figure('name', 'Knn Testing');
hold on
plot(Xtest, Ytest, 'go');
plot(Xtest, Yhat, 'ro');
legend('Y test', 'Y hat');
title('1NN on Motorcycle Testing Data');
xlabel('Time');
ylabel('Acceleration');
hold off
```



(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?

The higher the value of K, the lower the complexity.

```
ks = [1, 2, 3, 4, 5, 10, 50];
Xs = min(Xtrain):0.001:max(Xtrain); Xs = Xs';
figure('name', 'testing values of k')
hold on
learner = knnRegress(1,Xtrain, Ytrain);
for i=1:length(ks)
    learner = knnRegress(ks(i),Xtrain, Ytrain);
    stairs(Xs, predict(learner, Xs), '-', 'linewidth', 3);
end
title('kNN Regression Function for Motorcycle Data')
legend(cellstr(int2str(ks')))
hold off
```



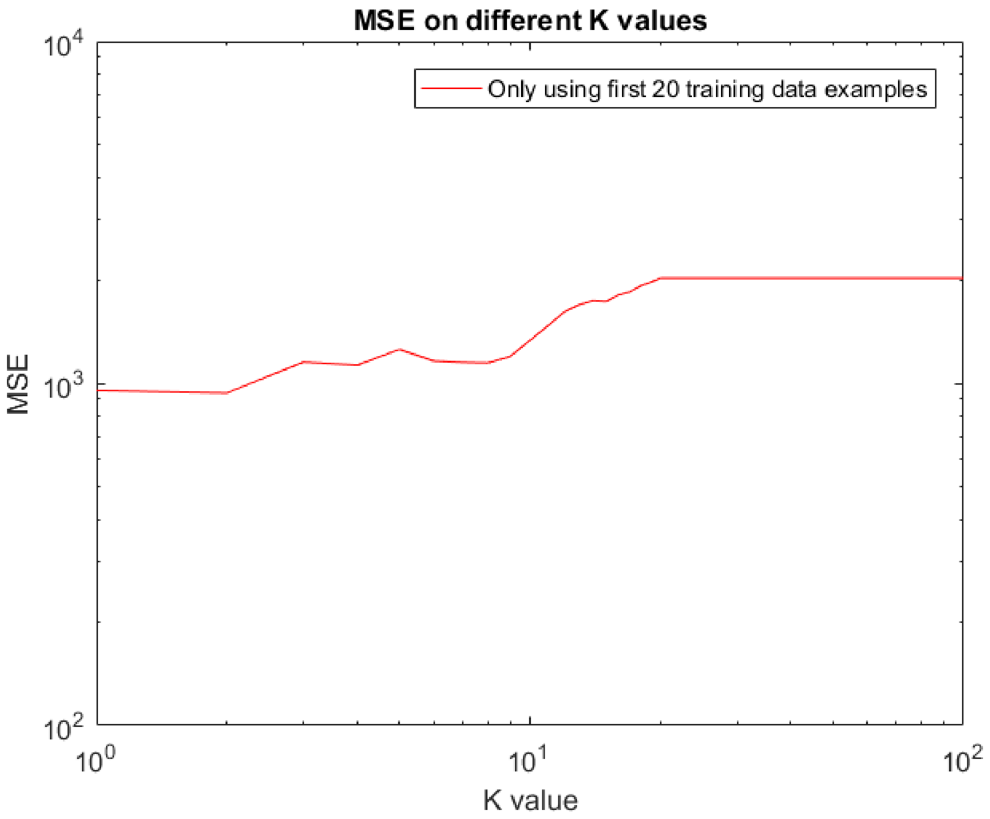
(c) What kind of functions can be output by a nearest neighbor regression function? Briefly justify your conclusion.

As a KNN regression produces a piecewise linear function, given the right data, any function can be approximated.

3. Hold-out and Cross-validation

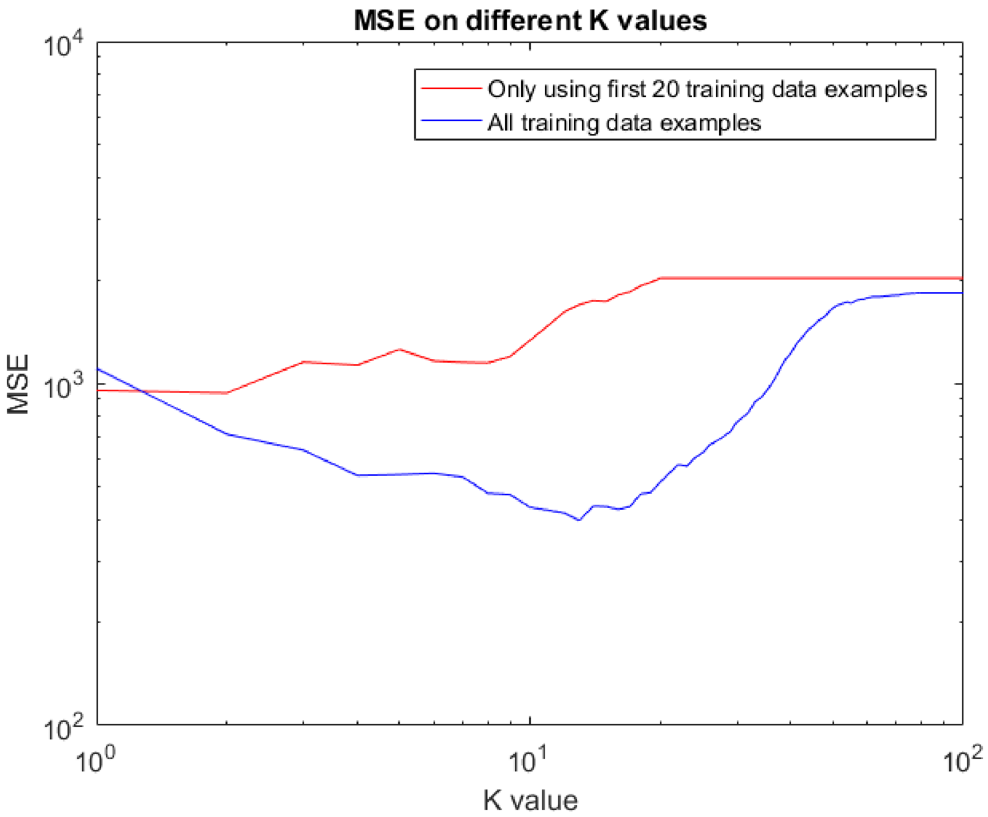
(a) Similarly to Problem 1 and 2, compute the MSE of the test data on a model trained on only the first 20 training data examples for $k = 1, 2, 3, \ldots, 100$**. Plot the MSE versus** k **on a log-log scale (see help loglog).**

```
%Clean up
clc
clear
close all
%Import data
mTrain = load('data/mcycleTrain.txt');
mTest = load('data/mcycleTest.txt');
ytst = mTest(:,1); %  testing data
xtst = mTest(:,2);

ytr = mTrain(1:20,1); % first 20 training data examples
xtr = mTrain(1:20,2);
MSE1 = zeros(100, 1); % initalise a vector for MSE of each k value

for k=1:100 % iterate on each k value
    learner = knnRegress(k, xtr, ytr); % create the learner using k
    yhat = predict(learner, xtst); % predict and find MSE relative to test data
    MSE1(k) = mean((yhat - ytst).^2);
end
figure;
loglog(1:100, MSE1, 'r'); % plot on a loglog graph
xlabel('K value');
ylabel('MSE');
title('MSE on different K values');
legend('Only using first 20 training data examples');
hold on;
```



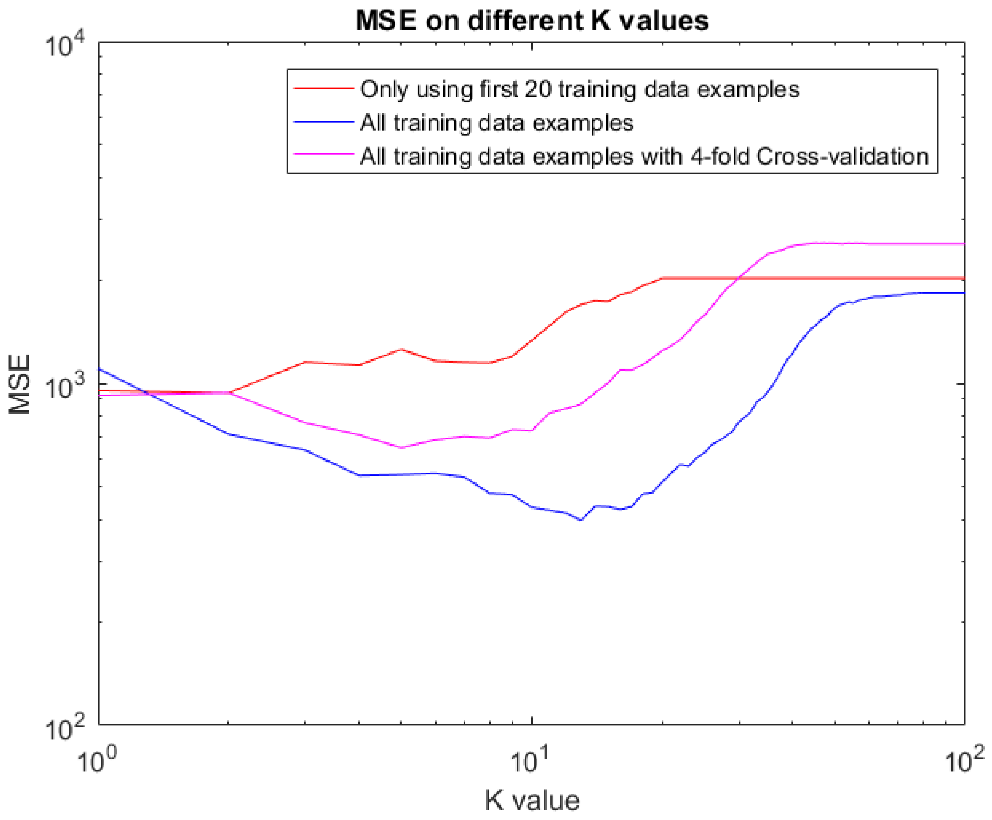
(b) Repeat, but use all the training data. What happened? Contrast with your results from problem 1 (hint: which direction is "complexity" in this picture?).

```
ytr = mTrain(:,1); % all training data examples
xtr = mTrain(:,2);
MSE2 = zeros(100, 1); % initalise a vector for MSE of each k value

for k=1:100 % iterate on each k value
    learner = knnRegress(k, xtr, ytr); % create the learner using k
    yhat = predict(learner, xtst); % predict and find MSE relative to test data
    MSE2(k) = mean((yhat - ytst).^2);
end

loglog(1:100, MSE2, 'b'); % plot on the loglog graph
legend('Only using first 20 training data examples', ...
    'All training data examples');
%
% _When using all training data, the MSE was able to be reduced much
% further.Contrast to problem 1, here we are increasing the complexity  of
% the sample (or training data) to reduce cost. Whereas in problem one we
% increased the model complexity by raising to 5th degree polynomial._
%
```



(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 21:80; 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. Add this curve to your plot. Why might we need to use this technique?

```
MSE3 = zeros(100, 1); % initalise a vector for MSE of each k value
for k=1:100 % test for 100 values of k
    MSEtemp = zeros(4, 1); % temp mse array for each i (averaged for each k)
    for i=1:4
        m = i*20; n = m-19; % local bounds
        iTest = mTrain(n:m,:); % 20 indicies for testing
        iTrain = setdiff(mTrain, iTest, 'rows'); % rest for training
        learner = knnRegress(k, iTrain(:,2), iTrain(:,1)); % train the learner
        yhat = predict(learner, iTest(:,2)); % predict at testing x values
        MSEtemp(i) = mean((yhat - iTest(:,1)).^2);
    end
    MSE3(k) = mean(MSEtemp); % average the MSE
end
loglog(1:100, MSE3, 'm'); % plot on the loglog graph
legend('Only using first 20 training data examples', ...
    'All training data examples', ...
    'All training data examples with 4-fold Cross-validation');
%
% _Using this technique may produce more accurate error values. This is
% because the effective 'testing data' changes multiple times per
% iteration, making it a better representation of the model. Also, in this
% case more data samples are used for testing because mTest only has 50
% samples relative to the 80 in mTrain.
%
```

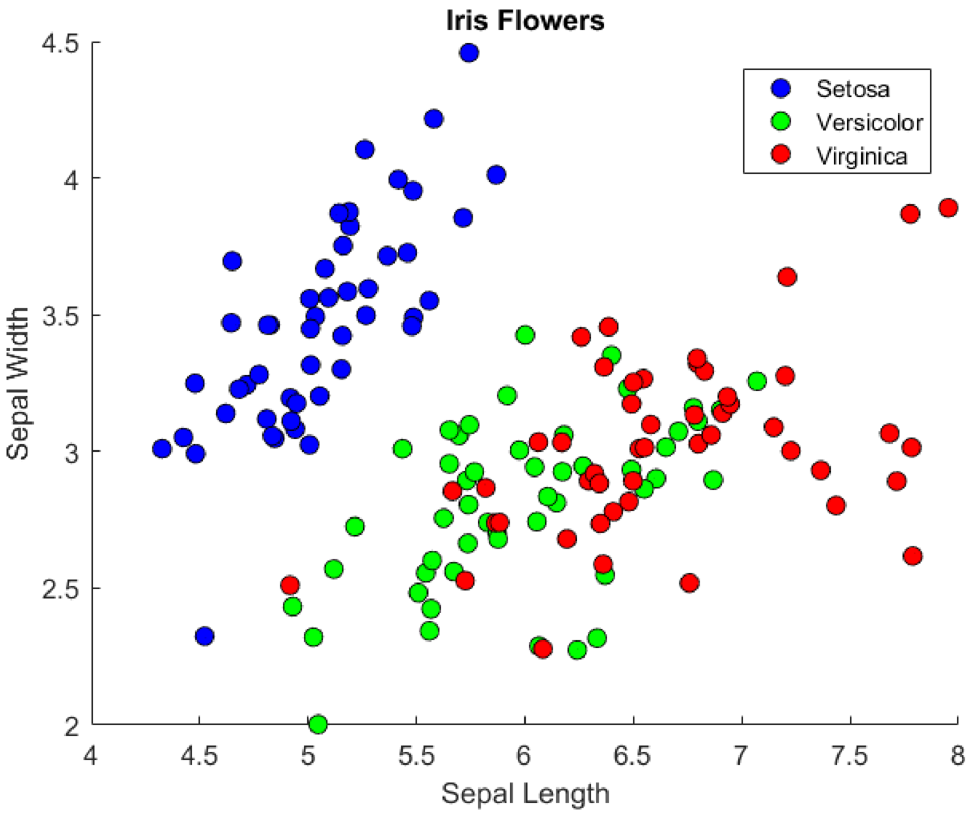


4. Nearest Neighbor Classifiers

```
clc
clear
% Load Iris Dataset
iris = load('iris.txt');
pi = randperm(size(iris, 1));
Y = iris(pi, 5);
X = iris(pi, 1:2);
m = length(Y);
```

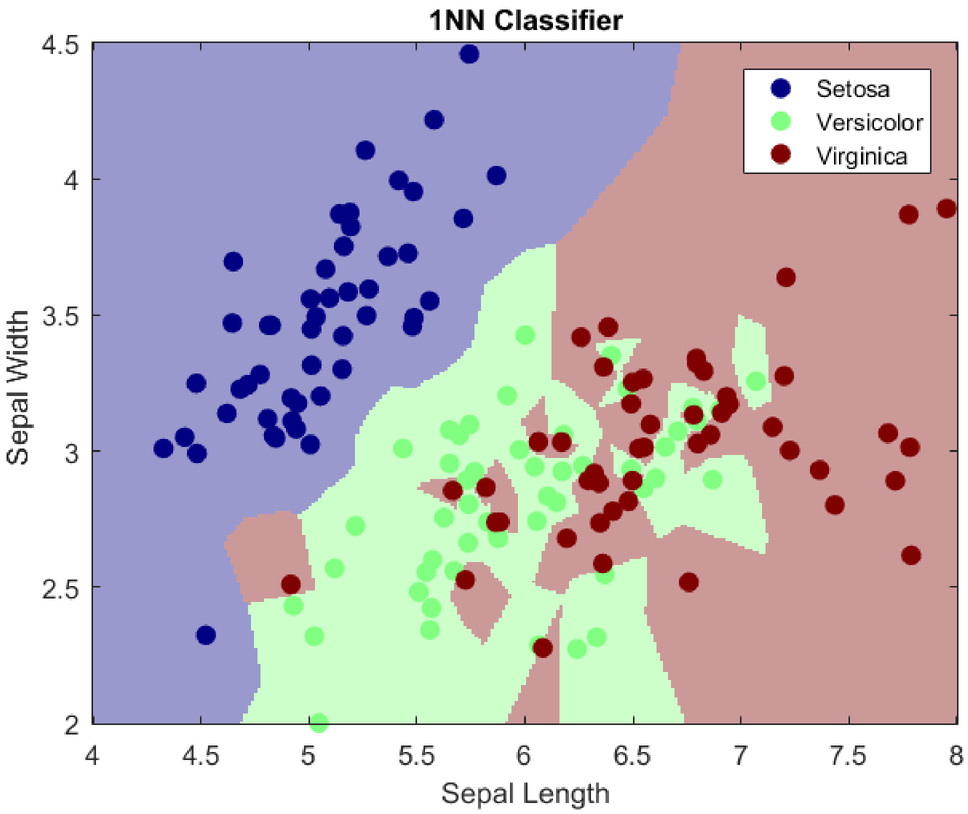
(a) Plot the data by their feature values, using the class value to select the color.

```
figure('name', 'Iris Flowers')
title('Iris Flowers')
xlabel('Sepal Length')
ylabel('Sepal Width')
hold on
plot(X(Y==0,1), X(Y==0,2), 'ko', 'markersize',7, 'markerfacecolor', 'blue');
plot(X(Y==1,1), X(Y==1,2), 'ko', 'markersize',7, 'markerfacecolor', 'green');
plot(X(Y==2,1), X(Y==2,2), 'ko', 'markersize',7, 'markerfacecolor', 'red');
legend('Setosa', 'Versicolor', 'Virginica')
hold off
```



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

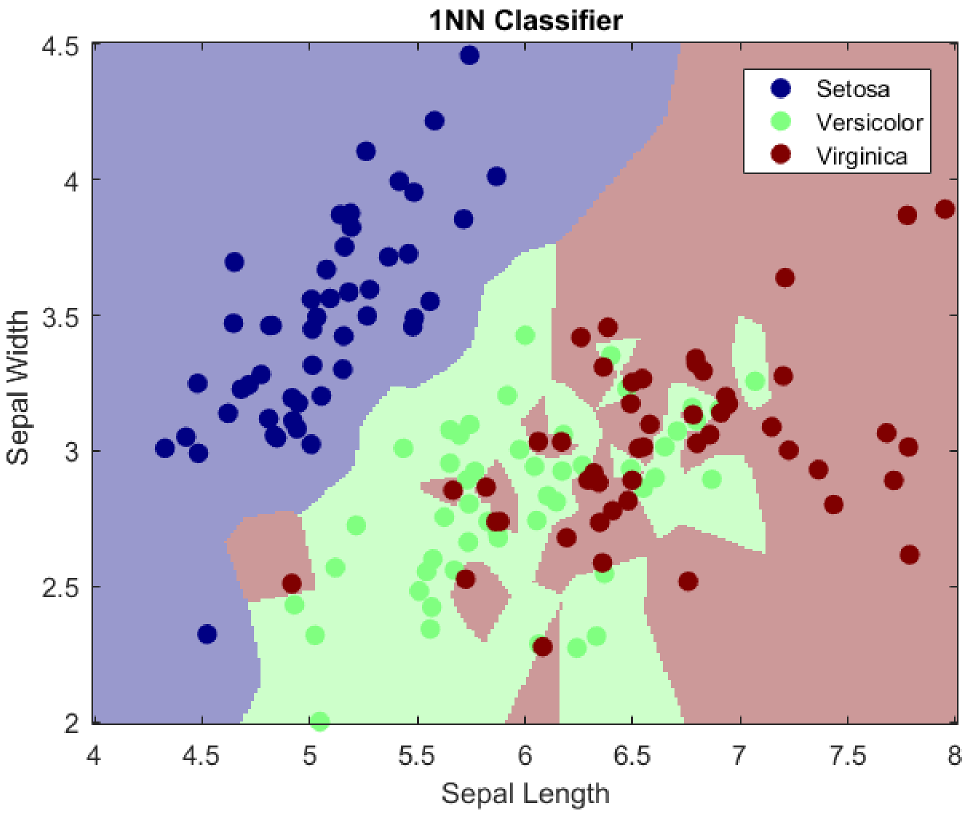
```
k = 1;
nnlearner = knnClassify(k, X, Y);
class2DPlot(nnlearner, X, Y);
title(string(k) + 'NN Classifier');
xlabel('Sepal Length');
ylabel('Sepal Width ');
legend('Setosa', 'Versicolor', 'Virginica')
```

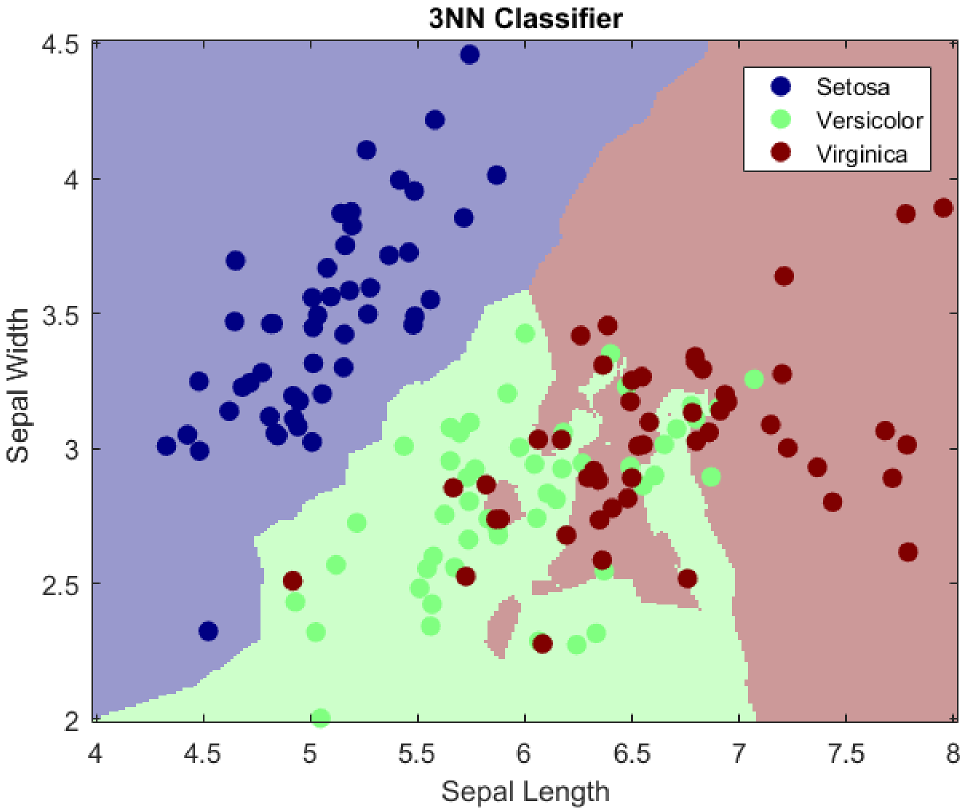


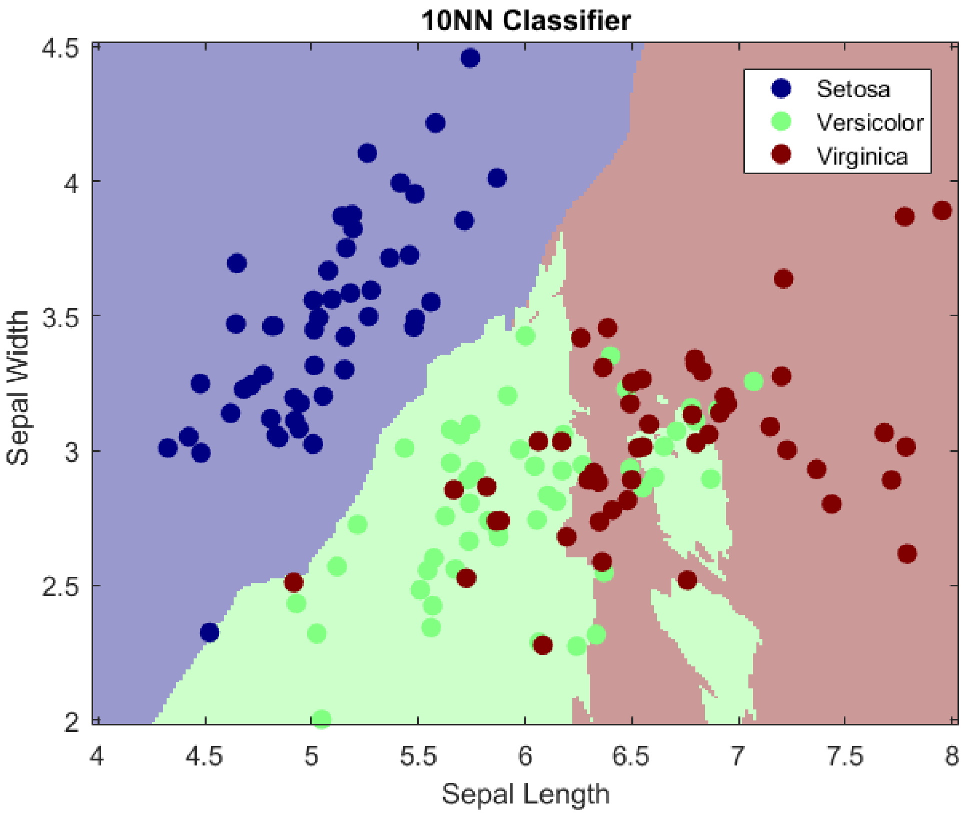
(c) Do the same thing for several values of k (say, [1, 3, 10, 30]) and comment on their appearance.

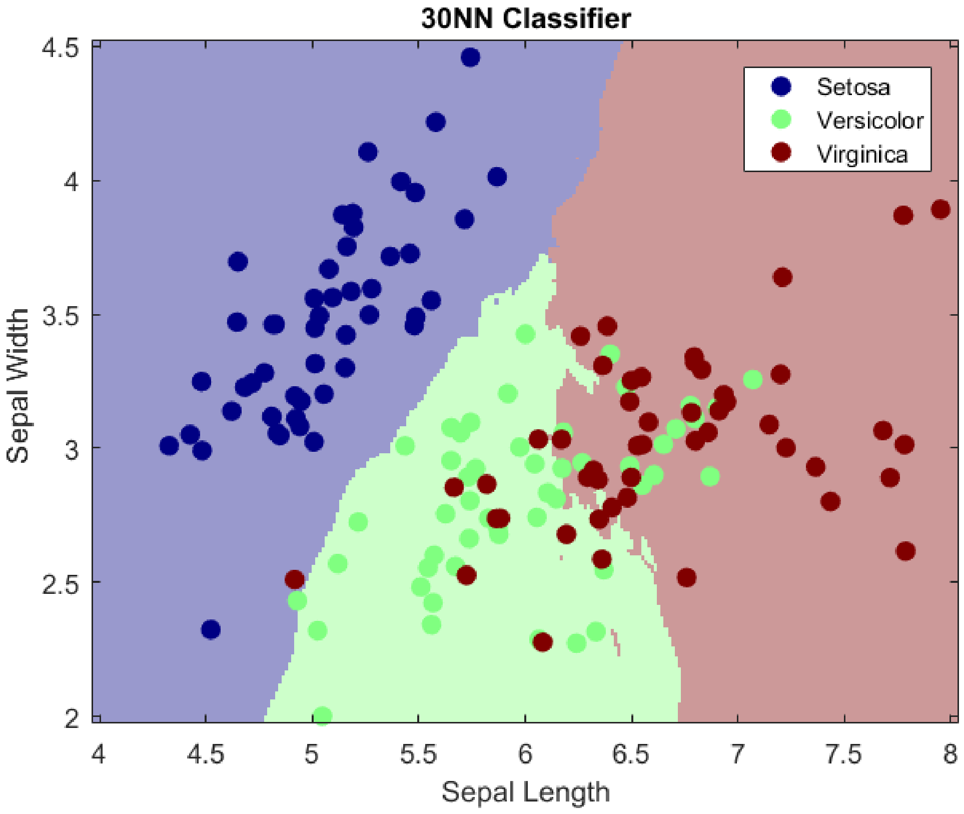
The higher the value of k, the smoother the desicion boundary. When k is close to 1, the classifier is prone to outliers. In general, this means that a larger value of K will likely lead to better generalisation and therefore a higher test time accuracy.

```
ks = [1, 3, 10, 30];
for i=1:length(ks)
    learner = knnClassify(ks(i), X, Y);
    class2DPlot(learner, X, Y);
    title(string(ks(i)) + 'NN Classifier');
    xlabel('Sepal Length');
    ylabel('Sepal Width ');
    legend('Setosa', 'Versicolor', 'Virginica')
end
```


1NN Classifier


3NN Classifier

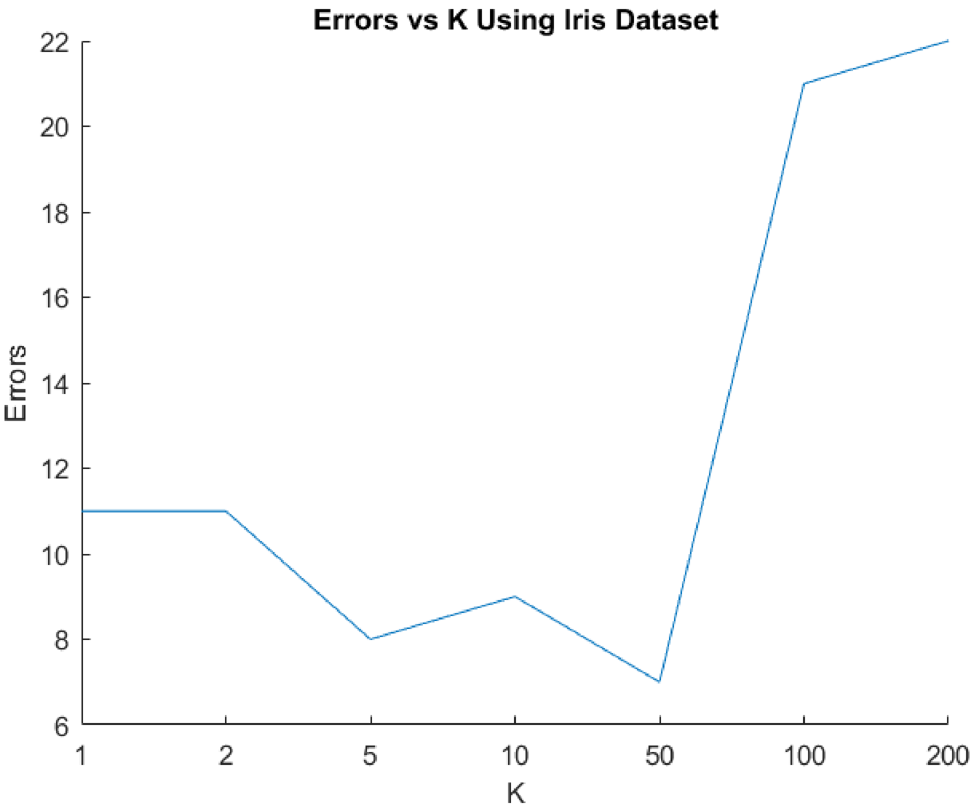




(d) Now split the data into an 80/20 training/validation split. For $k = [1, 2, 5, 10, 50, 100, 200]$, learn a model on the 80% and calculate its performance (# of data classified incorrectly) on the validation data. What value of k appears to generalize best given your training data? Comment on the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

k=10 to k=50 appears to generalize best on the training data. When k is near 1, the classifier overfits to the training data (k=1 has an accuracy of 1.0) when k gets too large however, the classifier underfits. When over fitting/underfitting occur depend on the size and shape of the dataset.

```
split = 0.8; train_size = floor(split*m);

Xtrain = X(1:train_size,:);
Ytrain = Y(1:train_size,:);
Xtest = X(train_size+1:end,:);
Ytest = Y(train_size+1:end,:);

figure('name', 'Errors vs K');
hold on

ks = [1, 2, 5, 10, 50, 100, 200];
errors = [];
for i=1:length(ks)
    learner = knnClassify(ks(i), Xtrain, Ytrain);
    Yhat = predict(learner, Xtest);
    errors = [errors, sum(Yhat ~= Ytest)];
end

title('Errors vs K Using Iris Dataset');
xlabel('K');
ylabel('Errors');
plot(errors);
xticklabels(ks)
hold off
```

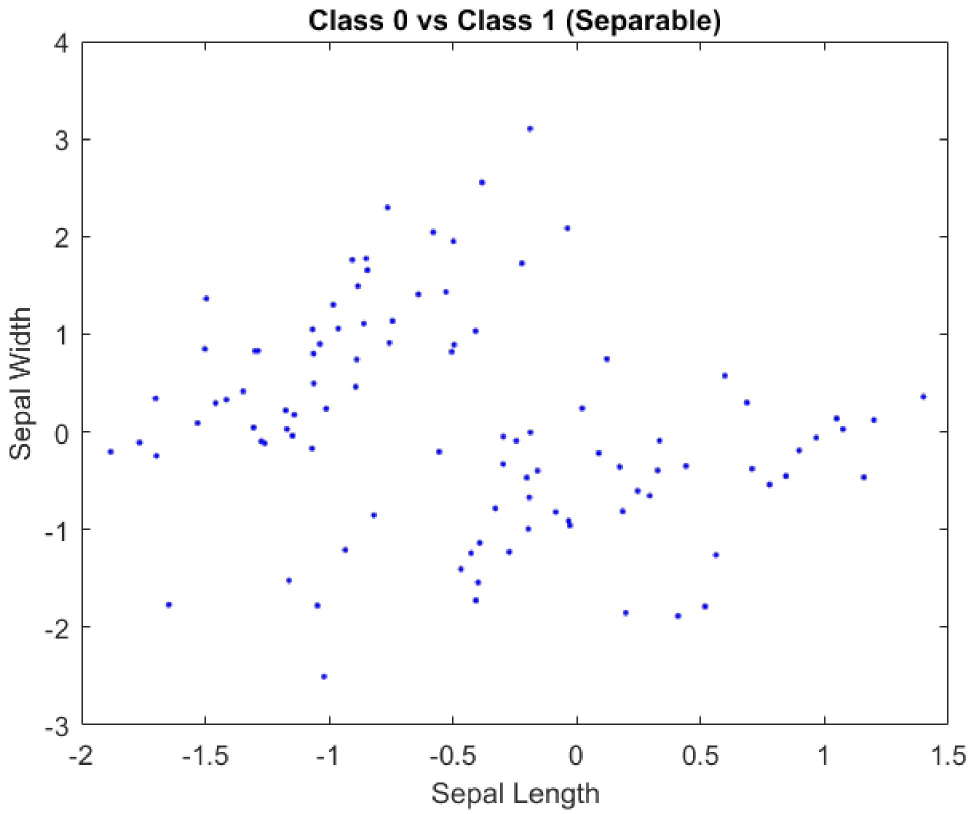


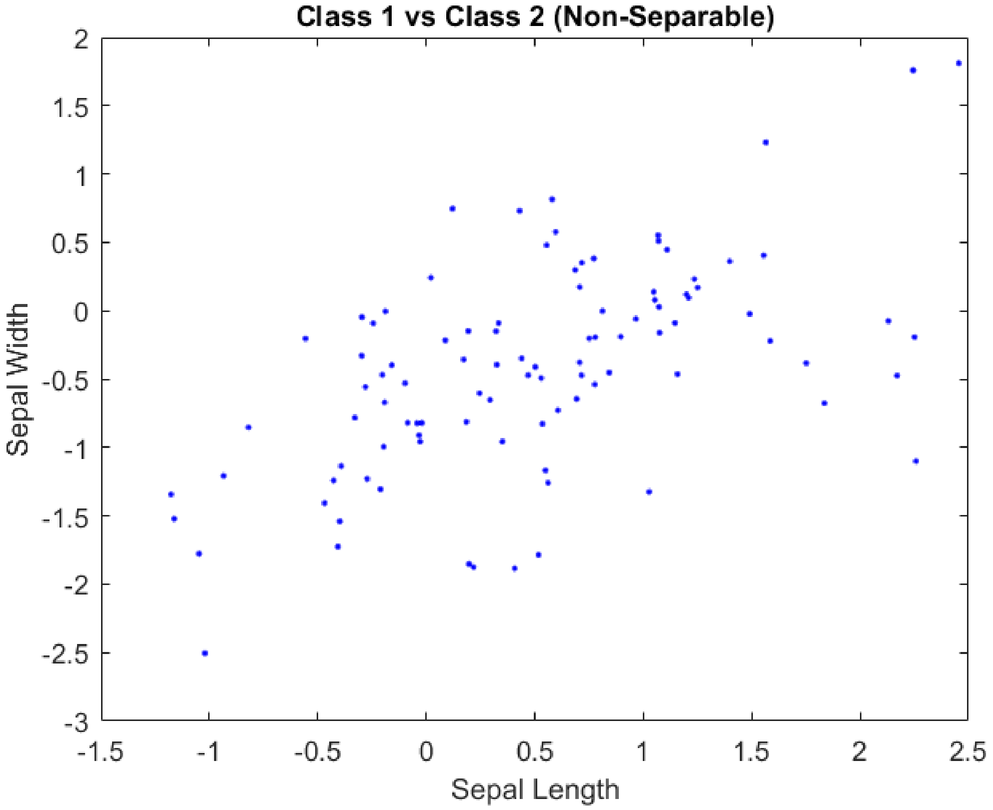
5. Perceptron and Logistic Regression

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

```
clear;
clc;
close all;
% Import
iris = load('data/iris.txt');
X = iris(:,1:2); Y = iris(:,end); % use the first two features and classifier
```

```
% Reformat data
[X, Y] = shuffleData(X,Y); % shuffle the data
X = rescale(X);
XA = X(Y<2,:); YA=Y(Y<2); % split into classes 0 and 1
XB = X(Y>0,:); YB=Y(Y>0); % and 1 and 2

% Plot class 0 vs 1
figure;
plot(XA(:,1), XA(:,2), 'b.');
title('Class 0 vs Class 1 (Separable)');
xlabel('Sepal Length');
ylabel('Sepal Width');
% Plot class 1 vs 2
figure;
plot(XB(:,1), XB(:,2), 'b.');
title('Class 1 vs Class 2 (Non-Separable)');
xlabel('Sepal Length');
ylabel('Sepal Width');
```



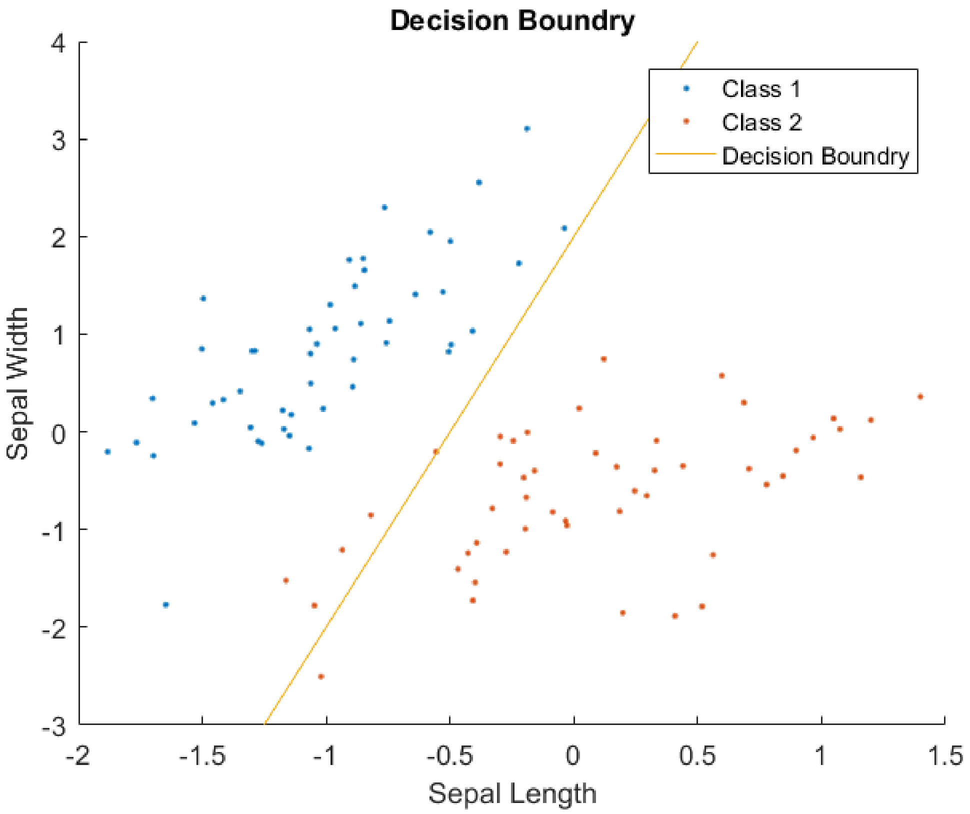


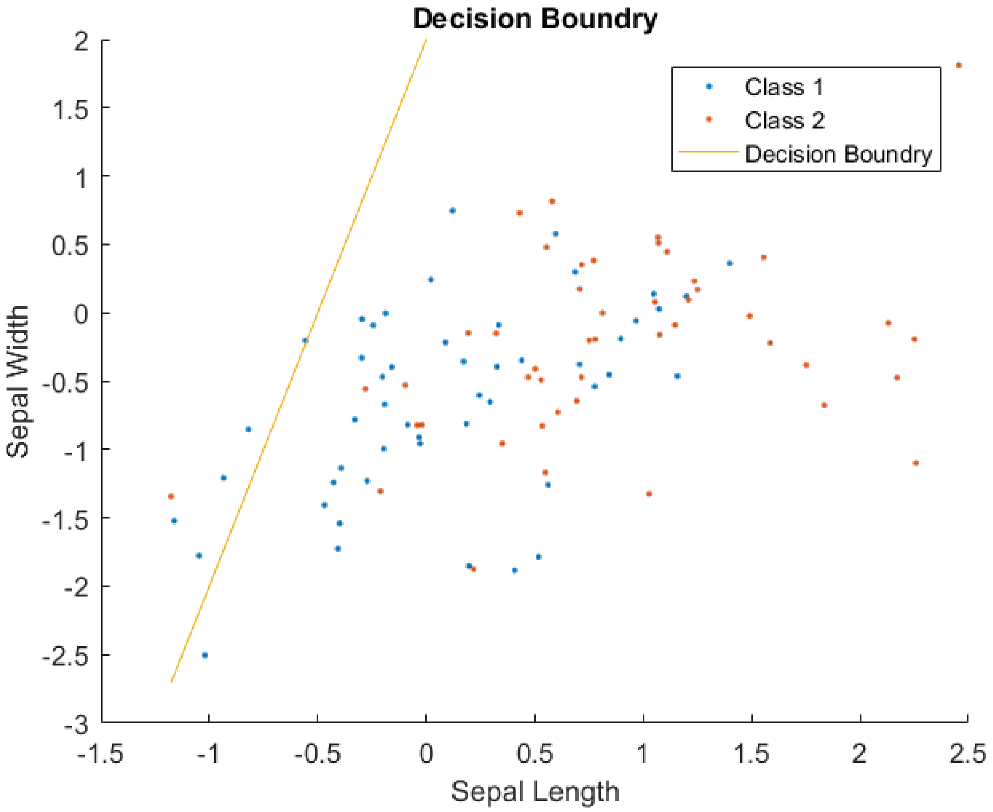
(b) Write (fill in) the function @logisticClassify2/plot2DLinear.m so that it plots the two classes of data in different colors, along with the decision boundary (a line). Include the listing of your code in your report. To demo your function plot the decision boundary corresponding to the classifier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

```
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;

%% Plot X seperately (by Y)
classes = getClasses(obj); % get the classes
figure;
hold on;

for i = 1:size(classes) % loop over the classes
    plot(X(Y==classes(i),1), X(Y==classes(i),2), '.'); % plot each point in this class
end;

%% Plot decision boundary
wts = getWeights(obj);
x = linspace(min(X(:,1)), max(X(:,1)));
y = -(wts(2)/wts(3))*x - (wts(1)/wts(3)); % re-arranged formula of theta'X=0

p = plot(x,y); % plotting in a way that doesn't change axis
set(p, 'YLimInclude', 'off');

%% Some Beautification
xlabel('Sepal Length');
ylabel('Sepal Width');
title('Decision Boundry');
legend('Class 1', 'Class 2', 'Decision Boundry');
```

```
% Plot decision boundry using provided weights
wts = [0.5 1 -0.25]; % set up the weights
learnerA = logisticClassify2(); % create "blank" learners
learnerB = logisticClassify2();
learnerA = setClasses(learnerA, unique(YA)); % define class labels
learnerB = setClasses(learnerB, unique(YB));

learnerA=setWeights(learnerA, wts); % set the learner's parameters
learnerB=setWeights(learnerB, wts);
plot2DLinear(learnerA, XA, YA); % plot
plot2DLinear(learnerB, XB, YB);
```





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

```
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] );
[n,d] = size(Xte);

wts = getWeights(obj);
```

```
X1 = [ones(n,1), Xte];
Yte = sign(wts*X1')';
Yte(Yte==-1) = obj.classes(1);
Yte(Yte==1) = obj.classes(2);
```

(d)

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

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

Derive the gradient of the regularized negative log likelihood J_j for logistic regression, and give it in your report

Expand $J_j(\theta)$ by substituting in σ and simplify:

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

Now find partial derivative:

$$\frac{\partial}{\partial\theta} y^j \log\left(1 + e^{-\theta x^j}\right)$$

$$y^j \frac{\partial}{\partial\theta} \log\left(1 + e^{-\theta x^j}\right)$$

$$\frac{y^j \frac{\partial}{\partial\theta}\left(1 + e^{-\theta x^j}\right)}{1 + e^{-\theta x^j}}$$

$$\frac{y^j \frac{\partial}{\partial\theta}\left(e^{-\theta x^j}\right)}{1 + e^{-\theta x^j}}$$

$$\frac{-x^j y^j e^{-\theta x^j}}{1 + e^{-\theta x^j}}$$

Simplifies to:

$$\frac{-x^j y^j}{1 + e^{\theta x^j}}$$

Now find partial derivative:

$$\frac{\partial}{\partial\theta} - (1 - y^{(j)}) \log\left(1 - \left(1 + e^{-\theta x^j}\right)^{-1}\right)$$

$$(y^{(j)} - 1) \frac{\partial}{\partial\theta} \log\left(1 - \left(1 + e^{-\theta x^j}\right)^{-1}\right)$$

$$\frac{(y^{(j)} - 1) \frac{\partial}{\partial\theta}\left(\frac{-1}{1+e^{-\theta x^j}}\right)}{1 - \frac{1}{1+e^{-\theta x^j}}}$$

$$\frac{(y^{(j)} - 1)\left(\frac{\frac{\partial}{\partial\theta} e^{-\theta x^j}}{(1+e^{-\theta x^j})^2}\right)}{1 - \frac{1}{1+e^{-\theta x^j}}}$$

$$\frac{(y^{(j)} - 1)\left(e^{-\theta x^j} \frac{\partial}{\partial\theta} - \theta x^j\right)}{\left(1 - \frac{1}{1+e^{-\theta x^j}}\right)\left(1 + e^{-\theta x^j}\right)^2}$$

$$\frac{(y^{(j)}-1)\left(-x^j e^{-\theta x^j}\right)}{\left(1-\frac{1}{1+e^{-\theta x^j}}\right)\left(1+e^{-\theta x^j}\right)^2}$$

Simplifies to:

$$-\frac{x^j e^{\theta x^j}\left(y^j-1\right)}{e^{\theta x^j}+1}$$

Now find partial derivative:

$$\frac{\partial}{\partial\theta}\alpha\sum_i \theta_i^2$$

$$\alpha\sum_i \frac{\partial}{\partial\theta}\theta_i^2$$

$$\alpha\sum_i 2\theta_i$$

Add together to find $\frac{dJ}{d\theta}$:

$$\frac{-x^j y^j}{1+e^{\theta x^j}}-\frac{x^j e^{\theta x^j}\left(y^j-1\right)}{e^{\theta x^j}+1}+\alpha\sum_i 2\theta_i$$

$$\frac{x^j\left(e^{\theta x^j}-y^j-y^j e^{\theta x^j}\right)}{1+e^{\theta x^j}}+\alpha\sum_i 2\theta_i$$

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

(1) computing the surrogate loss function at each iteration

```
Jsur(iter) = 1/n*sum((-X(:,2)'*log((1+exp(-obj.wts.*X(:,1))).^-1))...
    - ((1-X(:,2))'*log(1-((1+(exp(-obj.wts.*X(:,1)))).^-1)))...
    + (stepsize*sum(obj.wts.^2))...
    );  %compute surrogate (neg log likelihood) loss
```

```
Undefined function or variable 'n'.
Error in report (line 501)
Jsur(iter) = 1/n*sum((-X(:,2)'*log((1+exp(-obj.wts.*X(:,1))).^-1))...
```

(2) computing the prediction and gradient associated with each data point

(3) a gradient step on the parameters θ

(4) a stopping criterion.

```
if iter >= stopIter
    done = true;
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 (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)

```
close all
```

Published with MATLAB® R2016b