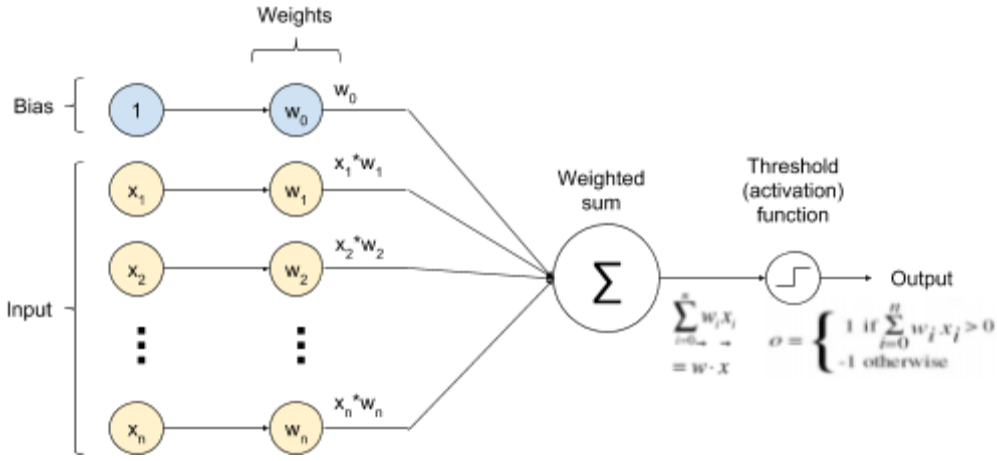
CAP-6619 Deep Learning Fall 2018
Homework 1
Christian Garbin

Question 1 [1 pt]

Please show the perceptron structure and explain the function of each component [0.5 pt].



- **Weights**: the result of training a perceptron. The weights are adjusted to correctly classify the training data and then used to predict the classification of unseen instances. In geometric terms, they define a hyperplane that separates the training data into its two classes.
- **Bias**: translates the the weighted sum, i.e. it shifts the decision boundary from the origin. Without bias we may not find a decision boundary (a hyperplane) because we wouldn't be able to shift (translate) the boundaries (hyperplanes) from the origin. It's usually set to "1" and only its weight (w_0) is adjusted.
- **Weighted sum**: linearly combines the weighted sum of the inputs: $w_0 + w_1x_1 + \ldots + w_nx_n$.
- **Threshold (activation) function**: a nonlinear function that limits the amplitude of the neuron's output, i.e. translates it from continuous values into discrete values.
- **Output**: a binary output (usually [0,1] or [-1,1]), result of the classification of the instance. Thus, a perceptron is used for *binary classification* tasks.

What is the purpose of using training examples in a neural network? Given proper weight values, what is expected output vs. actual output of an example? [0.5 pt]

The training examples are used to gain knowledge about the task by adjusting the weights of the inputs to minimize the errors.

The expected output is the label of the instance (as defined in the examples). The actual output is the result generated by the perceptron, using the trained weights and the activation function.

Question 2 [1 pt]

What is a Perceptron Learning Rule? [0.5 pt] Please use your own language, explain how perceptron learning rule updates/learns weight values for the network [0.5 pt]

The Perceptron Learning rule is a supervised training method. A set of labeled examples is used to adjust the weights of the network. Each misclassified example is used to adjust the weights to minimize the error. Assuming that the examples are linearly separable and a sufficiently small learning rate (η) is used, the Perceptron Learning Rule is guaranteed to converge.

The pseudocode for the rule is:

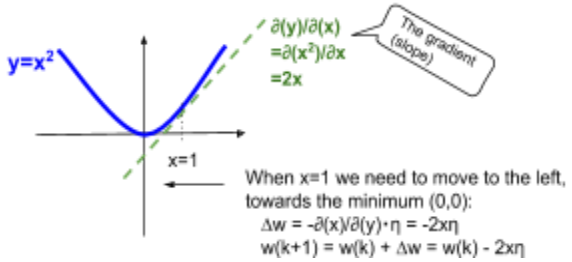
```
k = 1
initialize weights w_i(k) randomly
while there is a misclassified example
   select a misclassified example (x(n),d(n)) # n = instance number
   for i = 1 to m # m = number of features
      # Perceptron Training Rule:
      #   - Learning rate (η): how fast or slow to move
      #   - Expected ((d(n)) vs. actual value (a(n)) of this instance
      #   - The value of the i^th feature of this instance (x_i(n))
      Δw_i = η · (d(n) - a(n)) · x_i(n)
      w_i(k+1) = w_i + Δw_i
   end for
   k = k + 1
end while
```

Question 3 [1 pt]

What is a Gradient Descent Learning Rule? What is a Delta Rule, what are their differences?

The **Gradient Descent Rule** is a supervised training rule that works by adjusting the weights to decrease the network error by a small step until the error reaches a satisfactory value (or a maximum number of iterations is reached - to prevent looping forever). The network error is a quadratic function of the weights, thus its derivative exists everywhere. Decreasing the network error means moving in the opposite direction of the gradient (the slope of the curve - its derivative), towards the minimum of the quadratic function.

The picture below illustrates one step when $y=x^2$ and x=1. When x=1 we are to the right of the minimum (located at (0,0)). To minimize the network error we need to move to the left, towards the minimum. The direction we need to to move to is the negative of the derivative.



More formally: the network error is a quadratic function defined in terms of the expected value (label) and the actual value:

$$E(w_1, ..., w_m) = \frac{1}{2}\sum_{n \in D}(\underbrace{d(n)}_{\text{Expected value (label)}} - \underbrace{o(n)}_{\text{Actual value}})^2$$

To reduce the error (find a satisfactory error) we take small steps in the opposite direction of the gradient of E, thus the weight adjustment for feature in each step becomes:

$$w_i(k+1) = w_i(k) + \Delta w_i = w_i(k) - \eta(gradient of E(W)) = w_i(k) + \eta \sum_{n \in D}(d(n) - o(n))x_i(n)$$

The pseudocode for the Gradient Descent Training Rule is:

```
k = 1
initialize weight training value wᵢ(k) randomly
calculate E(W) # E(W) = network error
while E(W) is unsatisfactory and k < max iterations
   initialize each Δwᵢ to zero # i = 1...m (number of features)
   # Calculate the weight adjustment using all instances
   # x(n) = training example with m features <x₁, . . . , xₘ>
   # d(n) = the label/classification of that example (the desired output)
   for each instance (x(n), d(n)) in D do
      calculate network output o(n) = ∑ᵢwᵢ(k)xᵢ(n)
      for each weight dimension wᵢ(k) # i = 1...m (number of features)
         # Gradient Descent Learning Rule: take a small step in the opposite
         # direction of the gradient of E -- note that even correctly
         # classified instances contribute to learning
         #   - The current weight Δwᵢ
         #   - Learning rate (η), how fast or slow to move
         #   - Expected ((d(n)) vs. calculated value (a(n)) of this instance
         #   - The value of the iᵗʰ feature of this instance (xᵢ(n))
         Δwᵢ = Δwᵢ + η · (d(n) - o(n)) · xᵢ(n)
      end for
   end for
   # Adjust the weight
   for each weight dimension wᵢ(k) # i = 1...m (number of features)
      wᵢ(k+1) = wᵢ(k) + Δwᵢ
   end for
   # Update network using the new weight values
   calculate E(W) based on the updated wᵢ(k+1)
   k = k + 1
end while
```

The **Delta Rule** (AdaLine) is a modified version of the Gradient Descent Training Rule. It also uses the gradient descent method to reduce error, but it changes how the weights are adjusted. To increase the performance of the training rule it uses only one instance of the training data to adjust the weights in each step, instead of all instances of the training data.

Gradient Descent Learning Rule - "batch mode" because it uses the entire data set:

$$w_i(k+1) = w_i(k) + \eta \sum_{n \in D} (d(n) - o(n)) x_i(n)$$

Delta Learning Rule - "incremental mode" because it uses only one instance of the data set:

$$w_i(k+1) = w_i(k) + \eta (d(n) - o(n)) x_i(n)$$

Given a sufficiently small learning rate (η), it can arbitrarily approximate the Gradient Descent Learning Rule.

The pseudocode for the Delta Rule shows how the body of the loop is closer to the Perceptron Rule, compared with the nested loops in the batch mode of the Gradient Descent Learning Rule:

```
k = 1
initialize weight training value w_i(k) randomly
calculate E_D(W) # E_D(W) = error calculated with individual instance
while E_D(W) is unsatisfactory and k < max iterations
   select an example (x(n),d(n)) # n = instance number
   # Delta Training Rule: same as the batch mode gradient descent, but
   # using only one instance of the training data
   #   - Learning rate (η), how fast or slow to move
   #   - Expected ((d(n)) vs. calculated value (a(n)) of this instance
   #   - The value of the i^th feature of this instance (x_i(n))
   Δw_i = η · {d(n) - o(n)} · x_i(n)
   w_i(k+1) = w_i + Δw_i
   calculate E_D(W) # error calculation with one instance only
   K = k + 1
end while
```

The table below shows the differences between the rules. The main difference between Gradient Descent and Delta rules is in bold.

	Perceptron	Gradient Descent Rule	Delta Rule
Architecture	Single layer	Single layer	Single layer
Linear separable	Linear separable	Doesn't need to be linear separable and may contain noise	Doesn't need to be linear separable and may contain noise
Learning algorithm	Minimize number of misclassified examples	Minimize the squared error	Minimize the squared error
Weight update	Using only incorrectly classified examples	**Using all examples at the same time (correctly and incorrectly classified)**	**Using one example at a time (correctly and incorrectly classified)**
Application	Linear classification	Linear classification and regression	Linear classification and regression

Question 4 [2 pts]

Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. The class label of C1 is 1, and the class label of C2 is 0. Assuming η=1, and the initial weights are w_0=1, w_1=1, and w_2=1. Please use perceptron learning rule to learn a linear decision surface for these two classes. List the results in the first two rounds by using tables in the following form.

Assume the activation is defined as follow.

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Input	Weight	v	Desired	Actual	Update?	New Weight	$\Delta w = \eta(d-a)x_i$ (η=1)
(1,1,0)	(1,1,1)	1+1+0=2	1	1	No	(1,1,1)	--
(1,1,1)	(1,1,1)	1+1+1=3	1	1	No	(1,1,1)	--
(1,0,-1)	(1,1,1)	1+0-1=0	1	1	No	(1,1,1)	--
(1,0,1)	(1,1,1)	1+0+1=2	0	1	Yes	(0,1,0)	1*(0-1)*1,1*(0-1)*0,1*(0-1)*1=(-1,0,-1)
(1,-1,0)	(0,1,0)	0-1+0=-1	0	0	No	(0,1,0)	--
(1,-1,-1)	(0,1,0)	0-1+0=-1	0	0	No	(0,1,0)	--

The Second Pass

Input	Weight	v	Desired	Actual	Update?	New Weight	$\Delta w = \eta(d-a)x_i$ (η=1)
(1,1,0)	(0,1,0)	0+1+0=1	1	1	No	(0,1,0)	--
(1,1,1)	(0,1,0)	0+1+0=1	1	1	No	(0,1,0)	--
(1,0,-1)	(0,1,0)	0+0+0=0	1	1	No	(0,1,0)	--
(1,0,1)	(0,1,0)	0+0+0=0	0	1	Yes	(-1,1,-1)	1*(0-1)*1,1*(0-1)*0,1*(0-1)*1=(-1,0,-1)
(1,-1,0)	(-1,1,-1)	-1-1+0=-2	0	0	No	(-1,1,-1)	--
(1,-1,-1)	(-1,1,-1)	-1-1+1=-1	0	0	No	(-1,1,-1)	--

Question 5 [2 pts]

Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. The class label of C1 is 1, and the class label of C2 is 0. Assuming η=0.1, and the initial weights are w0=1, w1=1, and w2=1. Please use gradient learning rule to learn a linear decision surface for these two classes. List the results in the first two rounds by using tables in the following form.

First Round

Input	Weight	v (w_i*x_i)	Desired	Output	Δw	New Weight	$\eta(d(n)-o(n))x_i(n)$
(1,1,0) (1,1,1) (1,0,-1) (1,0,1) (1,-1,0) (1,-1,-1)	(1,1,1) (1,1,1) (1,1,1) (1,1,1) (1,1,1) (1,1,1)	1+1+0=2 1+1+1=3 1+0-1=0 1+0+1=2 1-1+0=0 1-1-1=-1	1 1 1 0 0 0	2 3 0 2 0 -1	(-0.1,-0.1,0) (-0.3,-0.3,-0.2) (-0.2,-0.3,-0.3) (-0.4,-0.3,-0.5) (-0.4,-0.3,-0.5) (-0.3,-0.4,-0.6)	(1,1,1) (1,1,1) (1,1,1) (1,1,1) (1,1,1) (0.7,0.6,0.4)	0.1*(-1*1,-1*1,-1*0)=(-0.1,-0.1,0) 0.1*(-2*1,-2*1,-2*1)=(-0.2,-0.2,-0.2) 0.1*(1*1,1*0,1*-1)=(0.1,0,-0.1) 0.1*(-2*1,-2*0,-2*1)=(-0.2,0,-0.2) (0,0,0) 0.1*(1*1,1*-1,1*-1)=(0.1,-0.1,-0.1)
					E(W)=5.5	w(k+1)	(0.7,0.6,0.4)

Second Round

Input	Weight	v (w_i*x_i)	Desired	Output	Δw	New Weight	$\eta(d(n)-o(n))x_i(n)$
(1,1,0) (1,1,1) (1,0,-1) (1,0,1) (1,-1,0) (1,-1,-1)	(0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4)	0.7+0.6+0=1.3 0.7+0.6+0.4=1.7 0.7+0-0.4=0.3 0.7+0+0.4=1.1 0.7-0.6+0=0.1 0.7-0.6-0.4=-0.3	1 1 1 0 0 0	1.3 1.7 0.3 1.1 0.1 -0.3	(-0.03,-0.03,0) (-0.1,-0.1,-0.07) (-0.03,-0.1,-0.14) (-0.14,-0.1,-0.25) (-0.15,-0.09,-0.25) (-0.12,-0.12,-0.28)	(0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.7,0.6.,0.4) (0.58,0.48,0.12)	0.1*(-0.3*1,-0.3*1,-0.3*0)=(-0.03,-0.03,0) 0.1*(-0.7*1,-0.7*1,-0.7*1)=(-0.07,-0.07,-0.07) 0.1*(0.7*1,0.7*0,0.7*-1)=(0.07,0,-0.07) 0.1*(-1.1*1,-1.1*0,-1.1*1)=(-0.11,0,-0.11) 0.1*(-0.1*1,-0.1*-1,-0.1*0)=(-0.01,0.01,0) 0.1(0.3*1,0.3*-1,0.3*-1)=(0.03,-0.03,-0.03)
					E(W)=1.19	w(k+1)	(0.58,0.48,0.12)

Question 6 [2 pts]

Assuming we have two sets of instances, which belong to two classes, with each class containing three instances. C1={(1, 0), (1, 1), (0, -1)}; C2={(0, 1), (-1, 0), (-1, -1)}. The class label of C1 is 1, and the class label of C2 is 0. Assuming η=0.1, and the initial weights are w0=1, w1=1, and w2=1. Please use Delta rule (AdaLine) to learn a linear decision surface for these two classes. List the results in the first round by using tables in the following form.

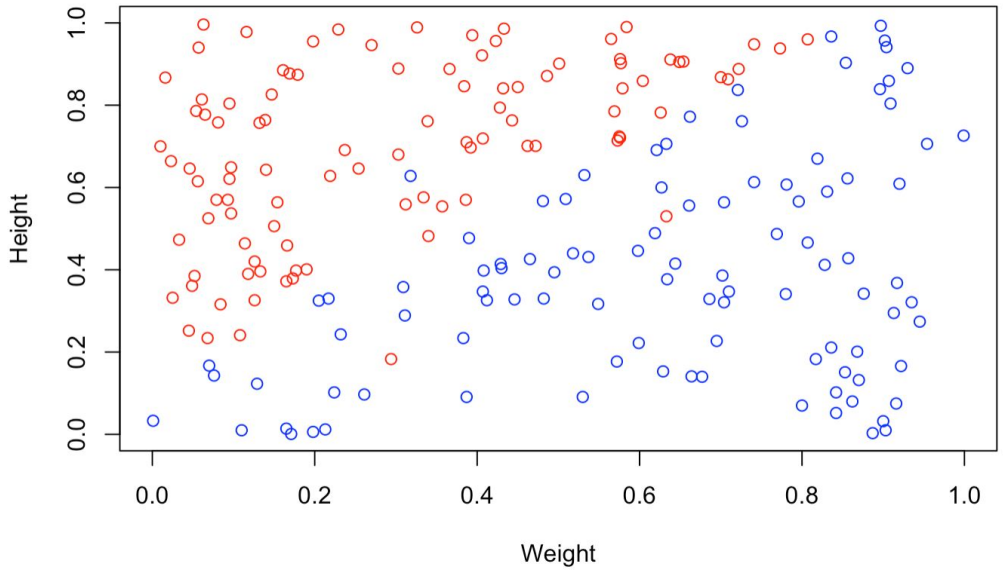
Input	Weight	v (w_i*x_i)	Desired	Output	Δw $\eta(d(n)-o(n))x_i(n)$	New Weight
(1,1,0) (1,1,1) (1,0,-1) (1,0,1) (1,-1,0) (1,-1,-1)	(1,1,1) (0.9,0.9,1) (0.72,0.72,0.82) (0.83,0.72,0.71) (0.68,0.72,0.56) (0.68,0.72,0.56)	1+1+0=2 0.9+0.9+1=2.8 0.72+0-0.82=-0.1 0.83+0+0.71=1.54 0.68-0.72+0=-00.4 0.68-0.72-0.56=-0.60	1 1 1 0 0 0	2 2.8 -0.1 1.54 -0.04 -0.60	0.1*(-1*1,-1*1,-1*0)=(-0.1,-0.1,0) 0.1*(-1.8*1,-1.8*1,-1.8*1)=(-0.18,-0.18,-0.18) 0.1*(1.1*1,1.1*0,1.1*-1)=(0.11,0,-0.11) 0.1*(-1.54*1,-1.54*0,-1.54*1)=(-0.154,0,-0.154) 0.1*(0.04*1,0.04*-1,0.04*0)=(0.004,-0.004,0) 0.1*(0.60*1,0.60*-1,0.60*-1)=(0.06,-0.06,-0.06)	(0.9,0.9,1) (0.72,0.72,0.82) (0.83,0.72,0.71) (0.68,0.72,0.56) (0.68,0.72,0.56) (0.74,0.66,0.50)

Question 7 [2 pts]

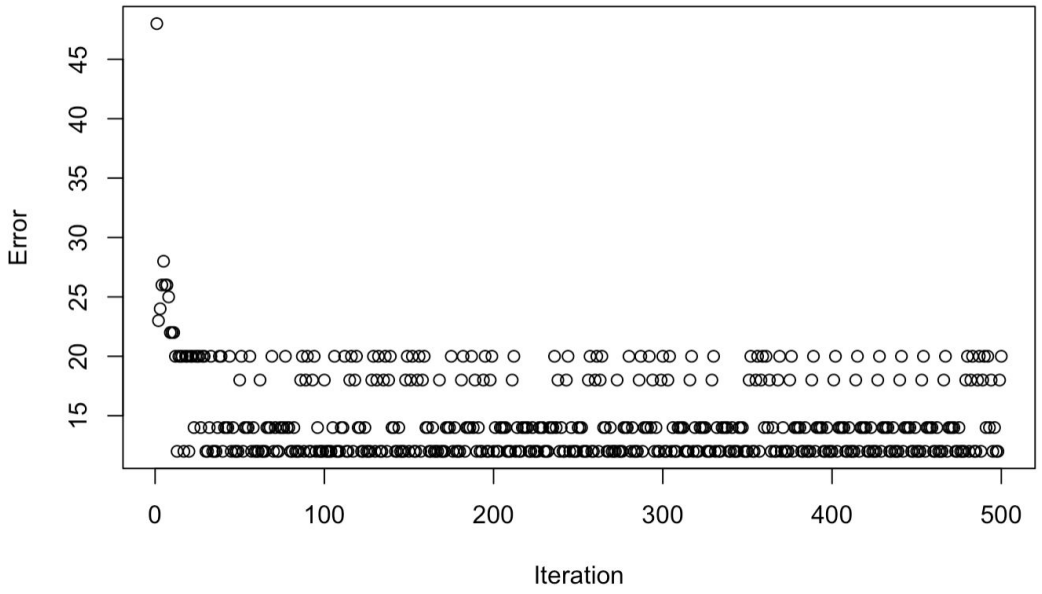
Class1.txt and Class2.txt files in the Canvas contain two-dimensional instances in two classes (C1 and C2 respectively, with 100 instances in each class). In these two files, the first column denotes the x values and the second column represents the y values (separated by comma). Please use the code showing in the Perceptron Learning R Notebook in the Canvas to implement the following tasks:

Please refer to cap6619-homework1-question7.Rmd and cap6619-homework1-question7.nb.html for the code used for the answers below.

Report the scatter plot of the all 200 instances in the same plot, using different color to show instance in different class [0.5 pt]



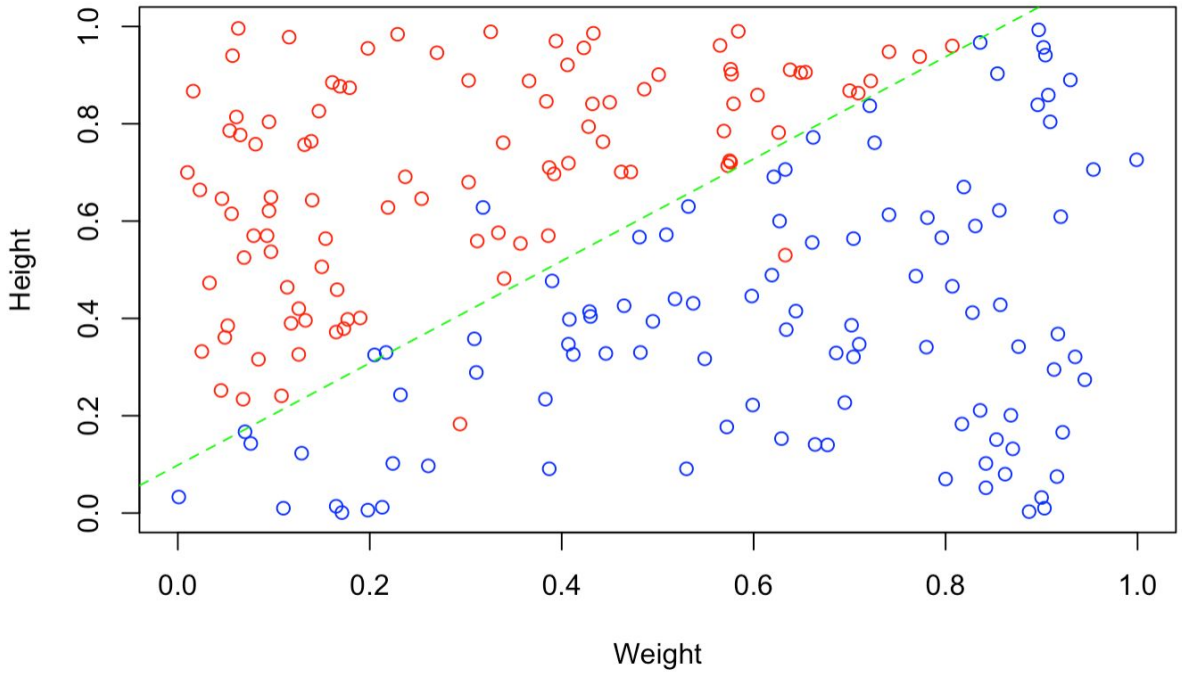
Use learning rate 0.05 and iteration 500, and report the error rates of the perceptron learning with respect to different iterations (using a plot where the x-axis denotes the iterations, and the y-axis shows the error rate. [0.5 pt]



Report final weight values and the slope and y-intercept of the decision surface [0.5 pt]

Weights: -0.1 -1.0651 1.0154
Slope: 1.048946
Intercept: 0.09848336

Report the final decision surface on the same scatter plot which shows the 200 instances [0.5 pt]



Question 8 [2 pts]

For datasets (Class1.txt and Class2.txt) in question 7, please write a Gradient Descent Learning R Notebook implement the following tasks.

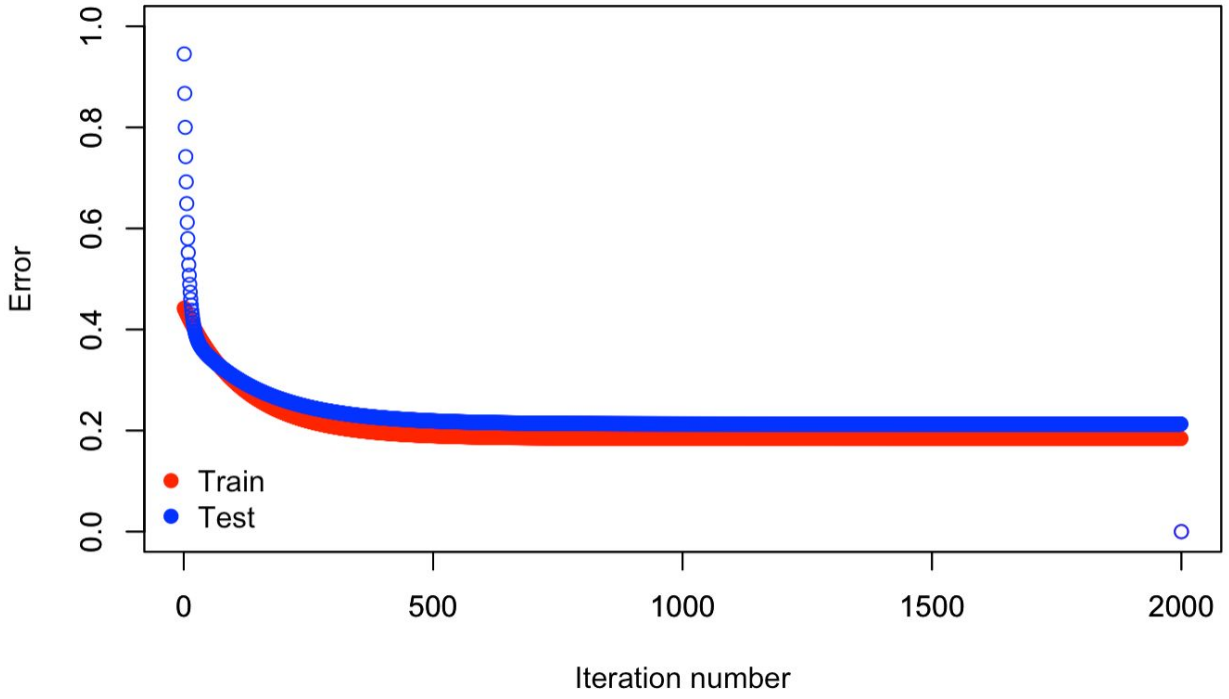
Please refer to cap6619-homework1-question8.Rmd and cap6619-homework1-question8.nb.html for the code used for the answers below.

Please randomly select 80% instances from class1.txt and 80% instances from class2.txt to train a perceptron classifier (using gradient descent learning rule), and use the classifier to classify remaining 20% instances in class1.txt and class2.txt. Please report the classification accuracy of the perceptron classifier on the 20% test instances (using learning rate 0.05, error threshold 0.1, and iteration numbers 2000) [0.5 pt]

```
predictions
 -1  1
-1 19  1
1   0 20
```

```
[1] "Accuracy: 0.9750"
```

Please report the training errors and test errors of the perceptron classifier with respect to each iteration. Please show the two error rates on the same chart, where the x-axis denotes the iteration and the y-axis denotes the mean classification errors [1 pt]



Report the final decision surface on the same scatter plot which shows the 200 instances [0.5 pt]