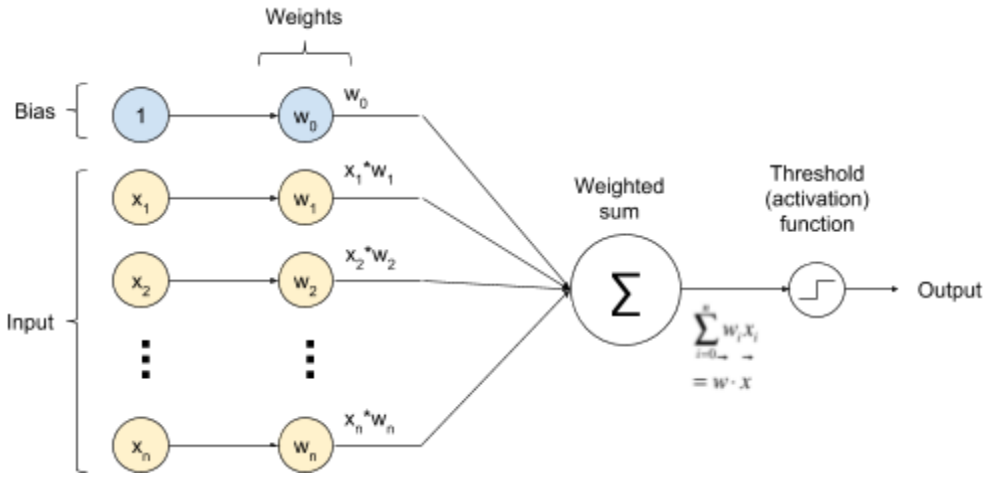
**CAP-5615 Intro to Neural Networks Summer 2018**
**Homework 3**
**Christian Garbin**

# 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.
- 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?

The training examples are used to gain knowledge about the task by adjusting the weights of the inputs until the neural network can correctly classify the training instances.

Given property weight values, what is expected output vs. actual output of an example? [0.5 pt]

Assuming a linearly separable set, the expected output and the actual output of an example will be the same, the class (label) of that example.

# Question 2 [1 pt]

## What is a Perceptron Learning Rule? [0.5 pt]

The Perceptron Learning Rule is a neuron learning rule that works by adjusting the weights and bias to separate the input values into classes. The rule assumes the training set is linearly separable.

## Please use your own language, explain how perceptron learning rule updates/learns weight values for the network [0.5 pt]

The Perceptron Training Rule iterates over the sample data to adjust the weights until all instances are correctly classified. As long as there are misclassified instances it updates the weight values using the difference between expected and actual value, adjusted with with a learning rate (η):

```
k = 1
initialize weight training value 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
```

Notes:

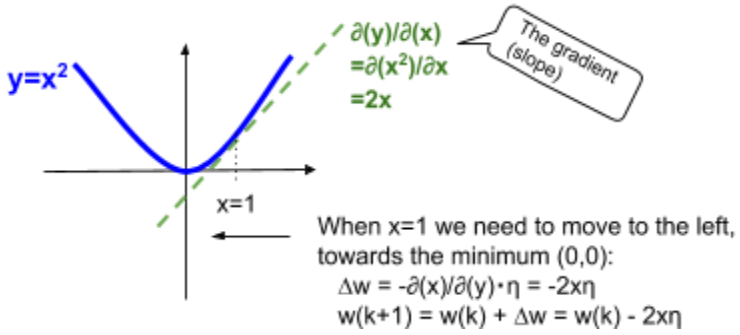
- The pseudocode above assumes the that the set is linearly separable. In real life the loop is also controlled by a variable to stop the loop after a predefined maximum number of iterations, in case the set is not linearly separable.
- Even when the set is linearly separable, we have to choose the learning rate (η) carefully. If it's too small it will take a long time to converge. If it's too large it may oscillate and not converge.
- As a consequence of the perceptron formula to adjust the weight, when the actual value matches the expected value (the instance is correctly classified: d(n) - a(n) = 0), the weight is not adjusted, i.e. the instance doesn't contribute to learning.

# Question 3 [1 pt]

## What is a Gradient Descent Learning Rule?

The Gradient Descent Rule is a neuron training rules that works by adjusting the weights to decrease the network error by a small step until it 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, with a derivative that exists everywhere. Thus 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 w_i(k) randomly
calculate E(W) # E(W) = network error
while E(W) is unsatisfactory and k < max iterations
   initialize each Δw_i to zero # i = 1...m (number of features)
   # Calculate the weight adjustment using all instances
   # x(n) = training example with m features <x_1,...,x_m>
   # 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) = ∑_i w_i(k)x_i(n)
      for each weight dimension w_i(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 Δw_i
         #   - 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 = Δw_i + η · (d(n) - o(n)) · x_i(n)
      end for
   end for
   # Adjust the weight
   for each weight dimension w_i(k) # i = 1...m (number of features)
      w_i(k+1) = w_i(k) + Δw_i
   end for
   # Update network using the new weight values
   calculate E(W) based on the updated w_i(k+1)
   k = k + 1
end while
```

The major differences between the Gradient Descent Learning Rule and the Perceptron Learning Rule are:

- Uses correctly classified examples to adjust the weight, i.e. all instances contribute to learning.
- It is guaranteed to converge to a hypothesis with minimum squared error (given sufficiently small learning rate), i.e. it works even when the training set is not linearly separable.
- Uses a continuous activation function (as opposed to the binary output function of the perceptron).

## What is a Delta Rule, what are their differences?

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
```

## Question 4 [2 pts]

Given a set of samples which belong to two classes C1 and C2, assume C1 and C2 are linearly separable, please prove that perceptron learning algorithm applied to the samples will terminate after a finite number of iterations.

Assumptions and preconditions to simplify the proof:

- To simplify the proof, each element *x* with class label = -1 is replaced with -*x*, for example (-1,-1) becomes (1,1). With that in place we can use the same weight update rule for both classes.
- The initial weight value is $\mathbf{w}(1)=0$ (where $\mathbf{w}(1)$ is the vector $<w_0(1), w_1(1), w_2(1), \ldots>$)
- The learning rate is η=1 (to simplify weight update calculations)
- $x(1),\ldots,x(k) \in C$ is the sequence of inputs that have been used in *k* iterations (they are misclassified - need to update the weight)

Then:

$$
\begin{aligned}
w(2) &= w(1) + x(1) \\
w(3) &= w(2) + x(2) \\
&\vdots \qquad\qquad \vdots \\
w(k) &= w(k-1) + x(k-1) \\
w(k+1) &= w(k) + x(k)
\end{aligned}
$$

*From w(k+1) = w(k) + ηx(k), where η=1 and x(1) is the first misclassified instance*

If we add all equations above we have:

$$w(2) + w(3) + \ldots + w(k+1) = w(1) + x(1) + w(2) + x(2) + \ldots + w(k) + x(k)$$

Subtracting $w(n)$, where n = 2...k, from both sides results in:

$$w(2) - w(2) + w(3) - w(3) + \ldots + w(k+1) = w(1) + x(1) + w(2) - w(2) + x(2) + \ldots + w(k) - w(k) + x(k)$$

We can see that we are subtracting a $w(n)$ term on the right side for each existing $w(n)$ term, eliminating all of them except for $w(1)$. But since we assumed $w(1)=0$, we end up eliminating all $w$ terms on the right side and end up with:

$$w(k+1) = x(1) + x(2) + \ldots + x(k) \qquad (1)$$

Since C1 and C2 are linearly separable, then there must exist a vector of weights $\mathbf{w}_*$ such that $\mathbf{w}_*^T\mathbf{x} > 0$, $\forall \mathbf{x} \in C$ (i.e. $\mathbf{w}_*$ correctly classifies all instances $\boldsymbol{x}$ in the training data).

Let:

$$\alpha = \min \mathbf{w}_*^{\mathrm{T}}\mathbf{x}$$ (of all *x* instances, this is the minimum value, but still > 0) (2)

Then:

$$\mathbf{w}_*^{\mathrm{T}}\mathbf{w}(k+1) = \mathbf{w}_*^{\mathrm{T}}\mathbf{x}(1) + \ldots + \mathbf{w}_*^{\mathrm{T}}\mathbf{x}(k) \geq k\alpha$$ (since from *(2)* all $\mathbf{w}_*^{\mathrm{T}}\mathbf{x}$ terms are $\geq \alpha$) (3)

By the Cauchy-Schwarz inequality we have:

$$\| \mathbf{w}_*^{\mathrm{T}} \|^2 \| \mathbf{w}(k+1) \|^2 \geq [\mathbf{w}_*^{\mathrm{T}}\mathbf{w}(k+1)]^2$$ ( $\| v \|$ = Euclidean norm, or magnitude: $\| v \|^2 = v \cdot v = (\sqrt{(a_1^2+a_2^2+\ldots)})^2$

$$\| \mathbf{w}_*^{\mathrm{T}} \|^2 \| \mathbf{w}(k+1) \|^2 \geq \| \mathbf{w}_*^{\mathrm{T}}\mathbf{w}(k+1) \|^2$$

$$\| \mathbf{w}_*^{\mathrm{T}} \|^2 \| \mathbf{w}(k+1) \|^2 \geq \| \mathbf{w}_*^{\mathrm{T}}\mathbf{x}(1) + \ldots + \mathbf{w}_*^{\mathrm{T}}\mathbf{x}(k) \|^2$$ (expanding from *(1)*)

$$\| \mathbf{w}_*^{\mathrm{T}} \|^2 \| \mathbf{w}(k+1) \|^2 \geq (k\alpha)^2$$ (from *(3)*)

$$\| \mathbf{w}(k+1) \|^2 \geq k^2\alpha^2 / \| \mathbf{w}_* \|^2$$ (every term here is a constant, except for $k$) (4)

At this point we have a formula for $\geq$, where all terms are numbers (constants), except for $k$ (the iteration counter).

Now will develop a formula for $\leq$, so that we can arrive at an equality formula, starting with:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k)$$

$$\| \mathbf{w}(k+1) \|^2 = \| \mathbf{w}(k) + \mathbf{x}(k) \|^2 = \| \mathbf{w}(k) \|^2 + \| \mathbf{x}(k) \|^2 + 2\mathbf{w}_*^{\mathrm{T}}(k)\mathbf{x}(k)$$

Because $\mathrm{x}(k)$ is a misclassified instance, the term $2\mathbf{w}_*^{\mathrm{T}}(k)\mathbf{x}(k)$ must be negative. Once we remove it we get:

$$\| \mathbf{w}(k+1) \|^2 \leq \| \mathbf{w}(k) \|^2 + \| \mathbf{x}(k) \|^2$$

For the sequence:

$$\| \mathbf{w}(2) \|^2 \leq \| \mathbf{w}(1) \|^2 + \| \mathbf{x}(1) \|^2$$

$$\| \mathbf{w}(3) \|^2 \leq \| \mathbf{w}(2) \|^2 + \| \mathbf{x}(2) \|^2$$

$$\vdots$$

$$\| \mathbf{w}(k+1) \|^2 \leq \sum_{i=1}^{k} \| \mathbf{x}(i) \|^2$$ (similar to the development of *(1)*) (5)

Let:

$$\beta = \max \| \mathbf{x}(i) \|^2, \ \mathbf{x}(n) \in C$$

Then:

$$\| \mathbf{w}(k+1) \|^2 \leq k\beta \quad \text{(since each term in } (5) \text{ is} \geq \beta \text{)} \qquad (6)$$

To satisfy (4) and (6) we have:

$$k^2_{max} \alpha^2 / \| \mathbf{w}_*^T \|^2 = k_{max} \beta$$

And thus:

$$k_{max} = \frac{\|\mathbf{w}_*\|^2}{\alpha} \beta$$

This formula shows that we have a maximum number of iterations $k_{max}$ until a solution is found that depends only on the constants $\mathbf{w}_*$, $\alpha$ and $\beta$.

## Question 5 [3 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)}. Assuming η=1, and the initial weights are $w_0$=1, $w_1$=0, 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.

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

### The First Round

**Note**: assuming that C1 has label 1 and C2 has label 0, as the examples we used in class.

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

The Second Pass

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

## Question 6 [2 pts]

A perceptron model is trained from a given training set, and is validated on a test set with 10 instances. The results generated from the perceptron model is shown in the following table, where the first column is the index of the test instances, the 2nd column denotes the predicted probability of the test instance belonging to class "1" (there are only two classes: "1" and "0"), and the third column denotes the true label of the test instance. Using decision rule: a test instance is classified as "1" if its predicted probability belonging to class "1" is greater or equal to 0.5, and "0" otherwise.

| Index | Predicted probablitly belonging to 1 | True Label |
|---|---|---|
| 1 | 0.8 | 1 |
| 2 | 0.2 | 0 |
| 3 | 0.4 | 1 |
| 4 | 0.55 | 1 |
| 5 | 0.45 | 1 |
| 6 | 0.9 | 1 |
| 7 | 0.3 | 0 |
| 8 | 0.4 | 0 |
| 9 | 0.56 | 1 |
| 10 | 0.92 | 1 |

Please report the confusion matrix of the perceptron model [0.5 pts].

Augmenting the table:

| Index | Predicted probability belonging to 1 | True Label | Predicted Label | TP, FP, TN, FN? |
|---|---|---|---|---|
| 1 | 0.8 | 1 | 1 | TP |
| 2 | 0.2 | 0 | 0 | TN |
| 3 | 0.4 | 1 | 0 | FN |
| 4 | 0.55 | 1 | 1 | TP |
| 5 | 0.45 | 1 | 0 | FN |
| 6 | 0.9 | 1 | 1 | TP |
| 7 | 0.3 | 0 | 0 | TN |
| 8 | 0.4 | 0 | 0 | TN |
| 9 | 0.56 | 1 | 1 | TP |
| 10 | 0.92 | 1 | 1 | TP |

Confusion matrix

| | | True label | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted label** | Positive | TP = 5 | FP = 0 |
| | Negative | FN = 2 | TN = 3 |

Please report the classification accuracy [0.5 pt], True Positive Rate [0.5 pt], and False Positive Rates [0.5 pt] of the perceptron (assuming “1” is the positive).

$$Accuracy = \frac{5+3}{5+0+2+3} = 0.8$$

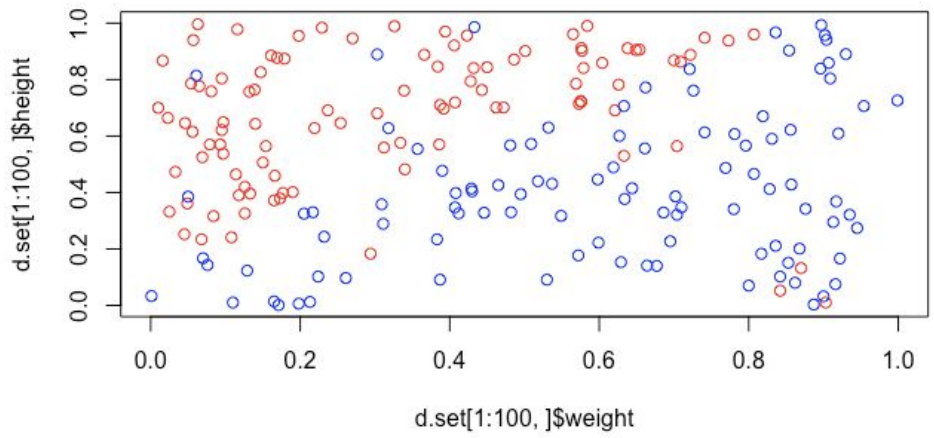
$$TPR = \frac{5}{5+2} = 0.714$$

$$FPR = \frac{0}{0+3} = 0$$

## Question 7 [3 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 Perceptron.R file in the Canvas to report the following results:

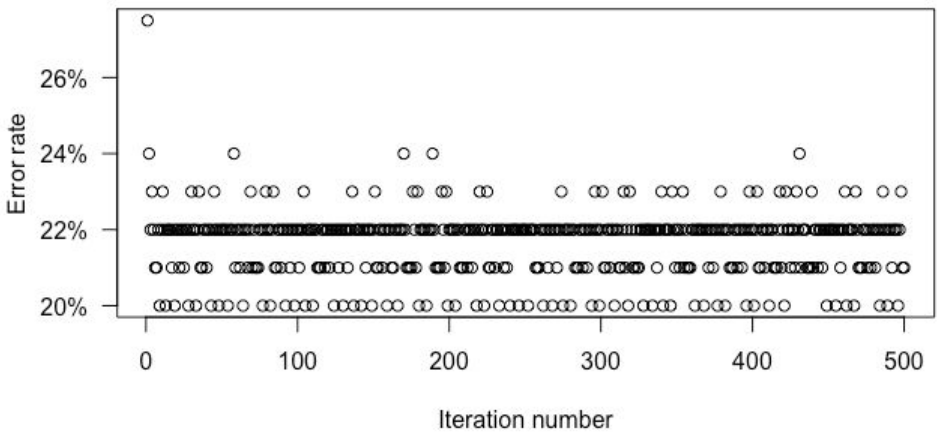
Report the scatter plot of the all 200 instances in the same plot, using different color to show instance in different class [0.5 pt]



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. [1 pt]

Note: since the question asked for the error rate (as opposed to count), the following R code was used to plot the graph:

```
error_rate = weight.err$v2 / 200 # 200 = # of instances in training data
plot(1:iterations, error_rate, xlab="Iteration number", ylab="Error rate", yaxt="n")
axis(2, at=pretty(error_rate), lab=paste0(pretty(error_rate) * 100, "%"), las=TRUE)
```

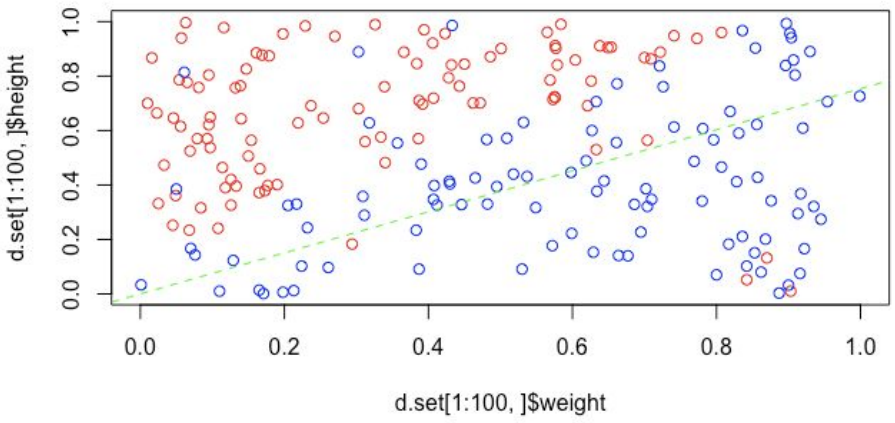



Report final weight values and the slope and y-intercept of the decision surface [1 pt]

```
> print(weight.err[1]) # weight values
$v1
[1]  0.0000 -0.3156  0.4188

> print(slope)
[1] 0.7535817
> print(intercept)
[1] 0
```

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



## Question 8 [3 pts]

For datasets (Class1.txt and Class2.txt) in question 6, please write a Gradient Descent Learning algorithm to find the decision surface to separate the two classes.

Please submit your final Gradient Descent Learning program (as a separated R file) [0.5 pt]

Please see attached file cap5615-homework3-question8.R.

**IMPORTANT:** Please note that that the learning rate 0.05 didn't work. The network error didn't converge with that value (probably overshot the minimum). To make it work I reduced the learning rate to 0.005.

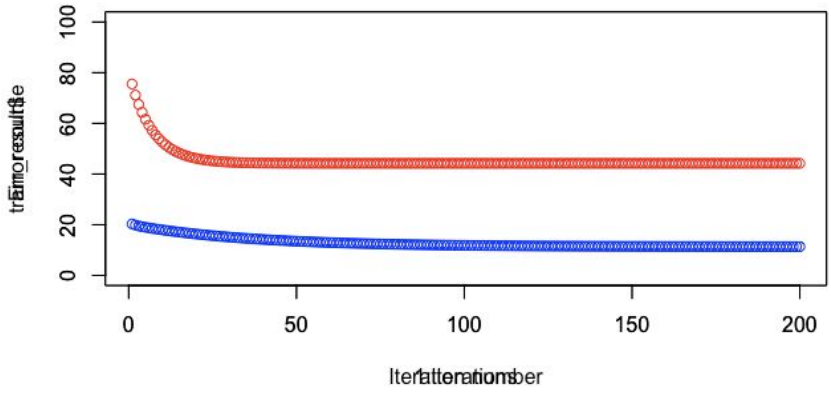
Please randomly select 80% instances from class1.txt and 80% instances from class2.txt to train a perceptron classifier (using gradient descent learning rule), and use the classifier to classify remaining 20% instances in class1.txt and class2.txt. Please report the classification accuracy of the perceptron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

The accuracy is 0.925. The relevant part of the code is:

```
ctable <- table(output_test, labels_test) # confusion table
ctable
TN <- ctable[1, 1]
FP <- ctable[1, 2]
FN <- ctable[2, 1]
TP <- ctable[2, 2]
accuracy <- (TN + TP) / (TN + FP + FN + TP)
```

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]

Training error is red, test error is in blue.



Report the final decision surface on the same scatter plot which shows the 200 instances [1 pt]