Module 2 - Perceptron & ADALINE: learning rules

Pawel Chilinski

November 4, 2014

The exercises have been implemented in R and document generated using sweave.

Exercise 1

Implement a single-unit perceptron together with its learning algorithm
Function to compute classification error:

```
>   classficationError<-function(fun,w,data){
+     sum(apply(data,1,function(row){
+             input<-row[-length(row)]
+             target<-row[length(row)]
+             target != fun(w,input)
+           }))/nrow(data)
+   }
```

```
> #Perceptron as a function of extended weight vector and extended input vector
> # w - extended weight vector
> # x - extended input vector
> perceptron<-function(w,x){
+   #return signum of the dot product between extended input vector and extended weight vector
+   return(sign(sum(w*x)))
+ }
> #Function training perceptron
> # trainData - matrix with train example per row (as extended input) and appended label as last column
> trainPerceptron<-function(trainData,maxEpochs=-1,lerarningRateFun=function(epoch){0.01}){
+   #random weights
+   set.seed(1023)
+   startW=runif(ncol(trainData)-1,-0.5,0.5)
+   w=startW
+   classErrHist<-classficationError(perceptron,w,trainData)
+   #while we don't want to stop yet
+   misclassified<-T
+   epoch<-1
+   while(misclassified & (epoch<maxEpochs | maxEpochs<0)){
+     misclassified<-F
+     #permute training examples for this epoch
+           permuation<-sample(nrow(trainData))
+           for(i in permuation){
+               #get input and its label for selected example
+             input<-trainData[i,-ncol(trainData)]
+             desiredOutput<-trainData[i,ncol(trainData)]
+               #when misclassified
+             if(perceptron(w,input) != desiredOutput){
+                 #update weights acording to perceptron rule.
+                      w=w+lerarningRateFun()*desiredOutput*input
+                 misclassified<-T
+               }
+     }
+     classErrHist<-c(classErrHist,classficationError(perceptron,w,trainData))
+     #print(paste("After epoch:",epoch,"w=",paste(w)))
+     epoch<-epoch+1
+   }
+   names(w)<-c("w0","w1","w2")
+   return(list(w=w,startW=startW,classErrHist=classErrHist))
+ }
```

Exercise 2

Implement a single ADALINE unit together with its Delta learning algorithm.

```
> #Ramp activation function for adeline classification
> ramp<-function(x){
+   if(x>-1 & x<1){
+     return(x)
+   }else{
+     return(sign(x))
+   }
+ }
> #ADALINE as a function of extended weight vector and extended input vector
> # w - extended weight vector
> # x - extended input vector
> adaline<-function(w,x){
+   #return signum of the dot product between extended input vector and extended weight vector
+   return(ramp(sum(w*x)))
+ }
> #computed RMSE for adaline model.
> # trainData - matrix with train example per row (as extended input) and target in the last column
> # w extended weights of the model
> rmseAdeline<-function(trainData,w){
+                   sqrt(sum(apply(trainData,1,function(row){
+                           input<-row[-length(row)]
+                       target<-row[length(row)]
+                 (target - adaline(w,input))^2
+                 }))/nrow(trainData))
+ }
> #Function training ADALINE using delta rule.
> # trainData - matrix with train example per row (as extended input) and target in the last column
> trainAdaline<-function(trainData,maxRmse,lerarningRateFun,maxEpochs=-1){
+   #random weights
+   set.seed(1024)
+   w=runif(ncol(trainData)-1,-0.5,0.5)
+   rmseHist<-c()
+   classErrHist<-classficationError(function(w,x){sign(adaline(w,x))},w,trainData)
+   startW=w
+   #while we don't want to stop yet
+   epoch<-1
+   while(T){
+     #permute training examples for this epoch
+     permuation<-sample(nrow(trainData))
+     for(i in permuation){
+       #get input and its target for selected example
+       input<-trainData[i,-ncol(trainData)]
+       desiredOutput<-trainData[i,ncol(trainData)]
+       #compute epsilon
+       eps<-desiredOutput-adaline(w,input)
+       #delta rule
+       w<-w+lerarningRateFun(epoch)*eps*input
+     }
+     #check of RMSE decreased sufficiently
+     rmse <- rmseAdeline(trainData,w)
+     rmseHist<-c(rmseHist,rmse)
+     classErrHist<-c(classErrHist,classficationError(function(w,x){sign(adaline(w,x))},w,trainData))
+     #print(paste("rmse: ",currentRmse))
+     if(rmse<maxRmse || (maxEpochs >0 && epoch >= maxEpochs)){
+       names(w)<-c("w0","w1","w2")
+       return(list(w=w,startW=startW,rmseHist=rmseHist,classErrHist=classErrHist))
+     }
+     epoch<-epoch+1
+   }
+ }
```

Exercise 3

Perform a training of the perceptron on the OR-type function approximation (linearly separable).

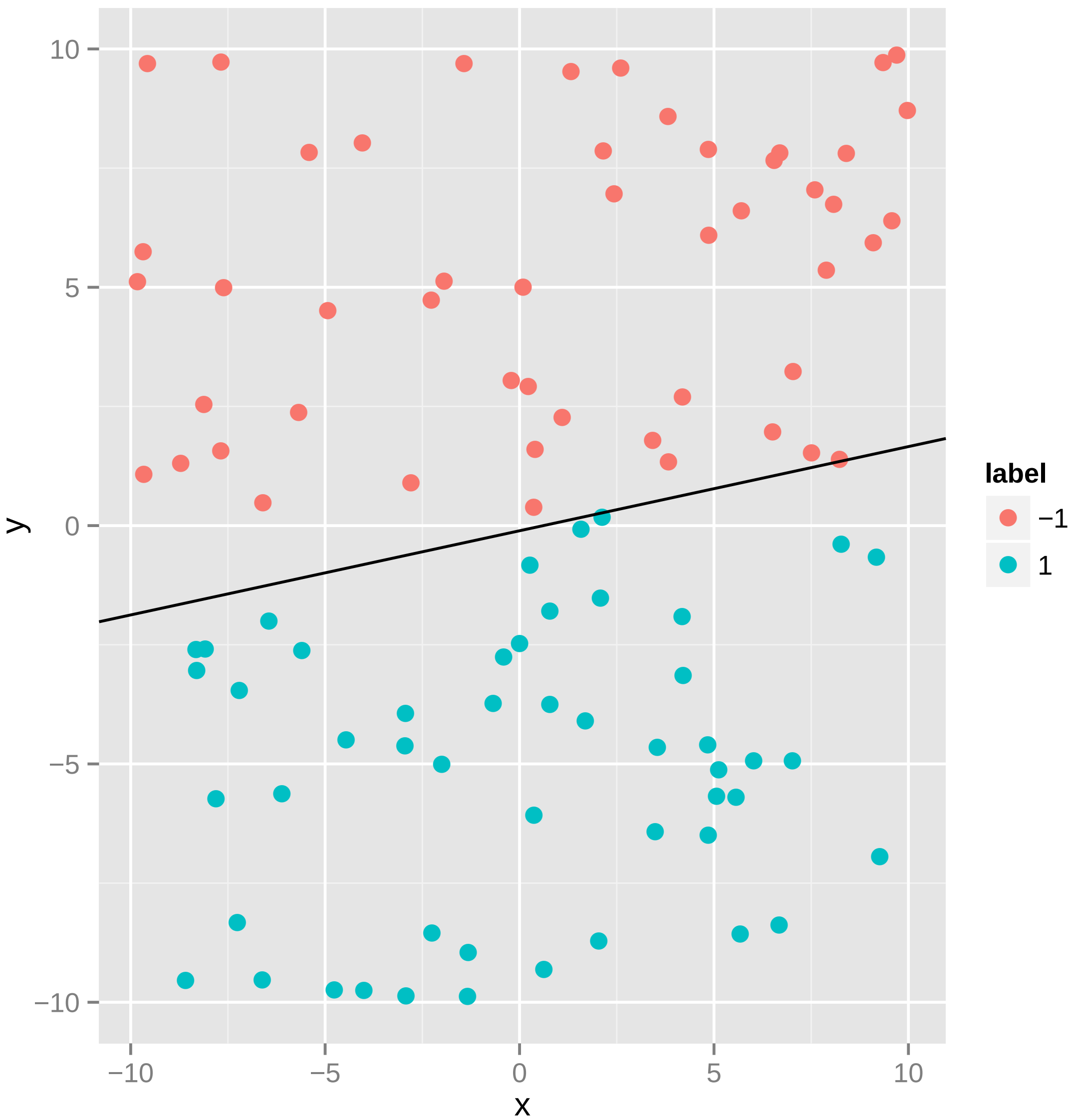
First define a function that allows to create points partitioned into 2 separable classes:

```
> #generates 2D data with 2 classes linearly separable. 1st column is the extension of input,
> #2nd column is x coordinate, 3rd column is y coordinate, 4th column is class
> #points - number of points
> orFunctionGenerateData<-function(points){
+   #generate matrix with data
+   data<-matrix(c(rep(1,points),runif(2*points,-10,10),rep(1,points)),ncol = 4)
+   #generate line separating 2 set of points
+   set.seed(1025)
+   w0<-runif(1,-5,-5)
+   w1<-runif(1,-100,100)
+   w2<-runif(1,-100,100)
+   data<-t(apply(data,1,function(row){
+       val <- w0*row[1]+w1*row[2]+w2*row[3]
+       if(val>=0){
+               c(row[1:3],1)
+       }else{
+         c(row[1:3],-1)
+       }
+     }))
+   colnames(data)<-c("ex","x","y","label")
+   return(list(data=data,w=c(w0,w1,w2)))
+ }
>
```

Generate points:

```
> separableTrainingData<-orFunctionGenerateData(100)
```

and show them on a plot with the true separating line:



generated weights:

```
> separableTrainingData$w
```

```
[1]  -5.000000   8.140375 -46.098459
```

Train the perceptron:

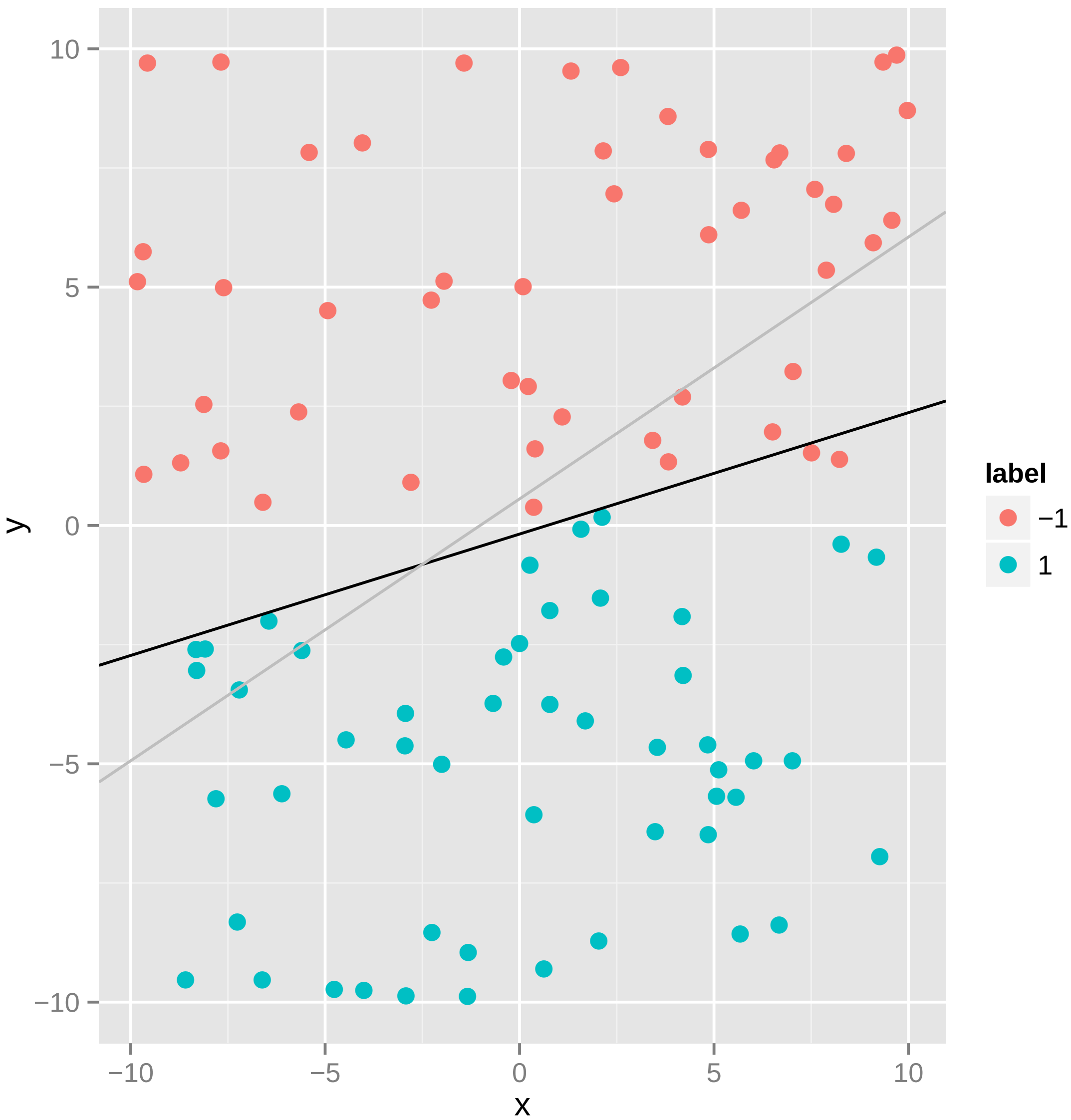
```
>         trainedPerceptron <- trainPerceptron(separableTrainingData$data,maxEpochs = 10)
>         trainedW <- trainedPerceptron$w
```

Show trained weights:

```
> trainedW
```

```
         w0          w1          w2
-0.05064198  0.07150082 -0.28075035
```

Random initialisation of algorithm and trained weights as a line:



The perceptron model fitted the data perfectly, of course the final solution is different from the data generating model (original line separating classes).

Exercise 4

Perform a training of the ADALINE on the OR-type function approximation (linearly separable). Try 3 different learning rates. Is it worth to implement exponentially decreasing learning rate in this case?

Reusing data generated for the previous exercise to train ADALINE:

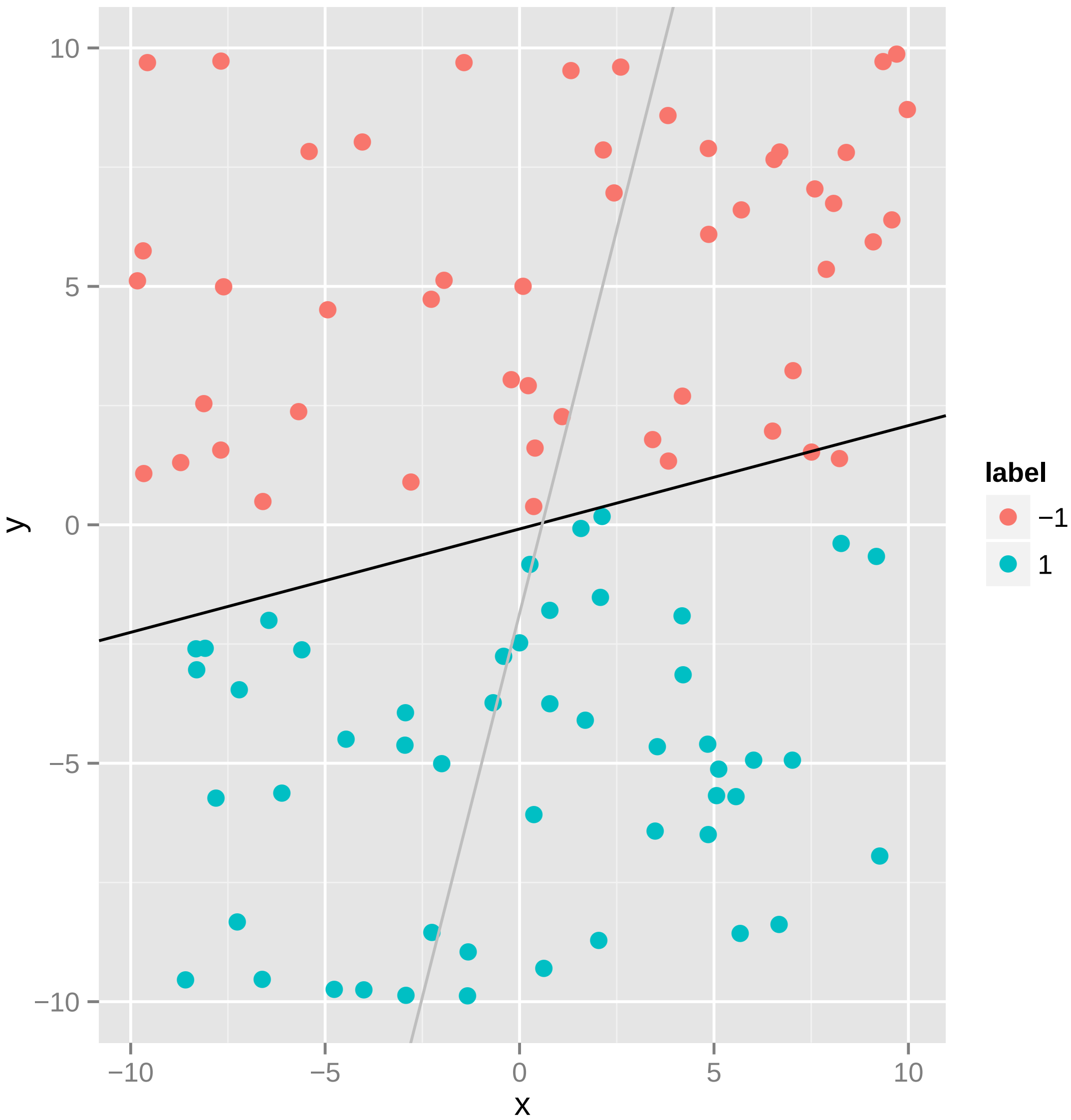
```
>         trainedAdaline<-trainAdaline(separableTrainingData$data,1,function(epoch){0.1})
>           trainedWAdaline<-trainedAdaline$w
```

Show trained weights:

```
> trainedWAdaline

        w0         w1         w2
-0.2328629  0.5799952 -2.6754121
```

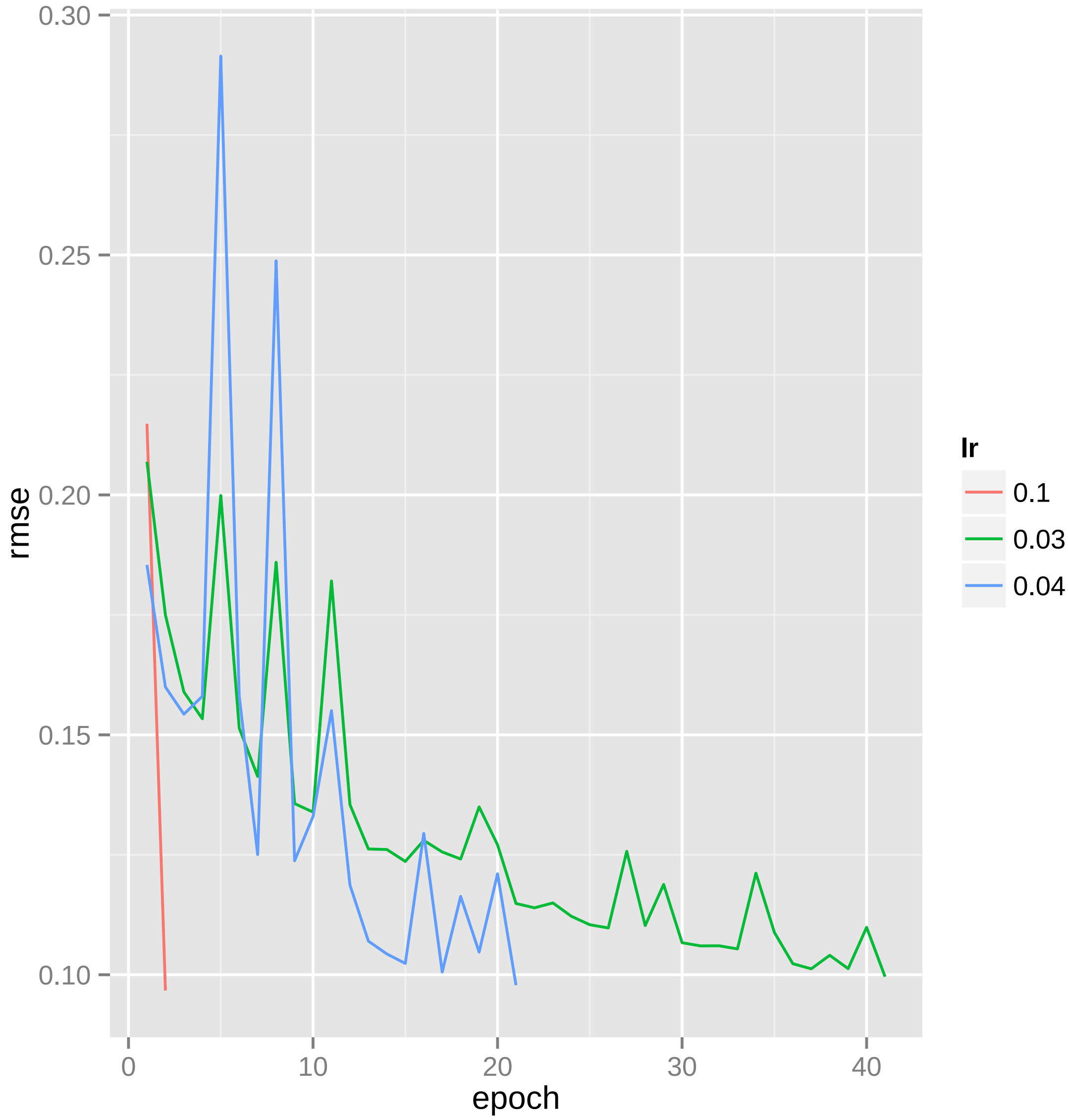
Random initialisation of algorithm and trained weights as a line:



Trying training ADALINE with different weights:

```
>  trainedAdalineLR1<-trainAdaline(separableTrainingData$data,0.1,function(epoch){0.1})
>  trainedAdalineLR2<-trainAdaline(separableTrainingData$data,0.1,function(epoch){0.03})
>  trainedAdalineLR3<-trainAdaline(separableTrainingData$data,0.1,function(epoch){0.04})
```

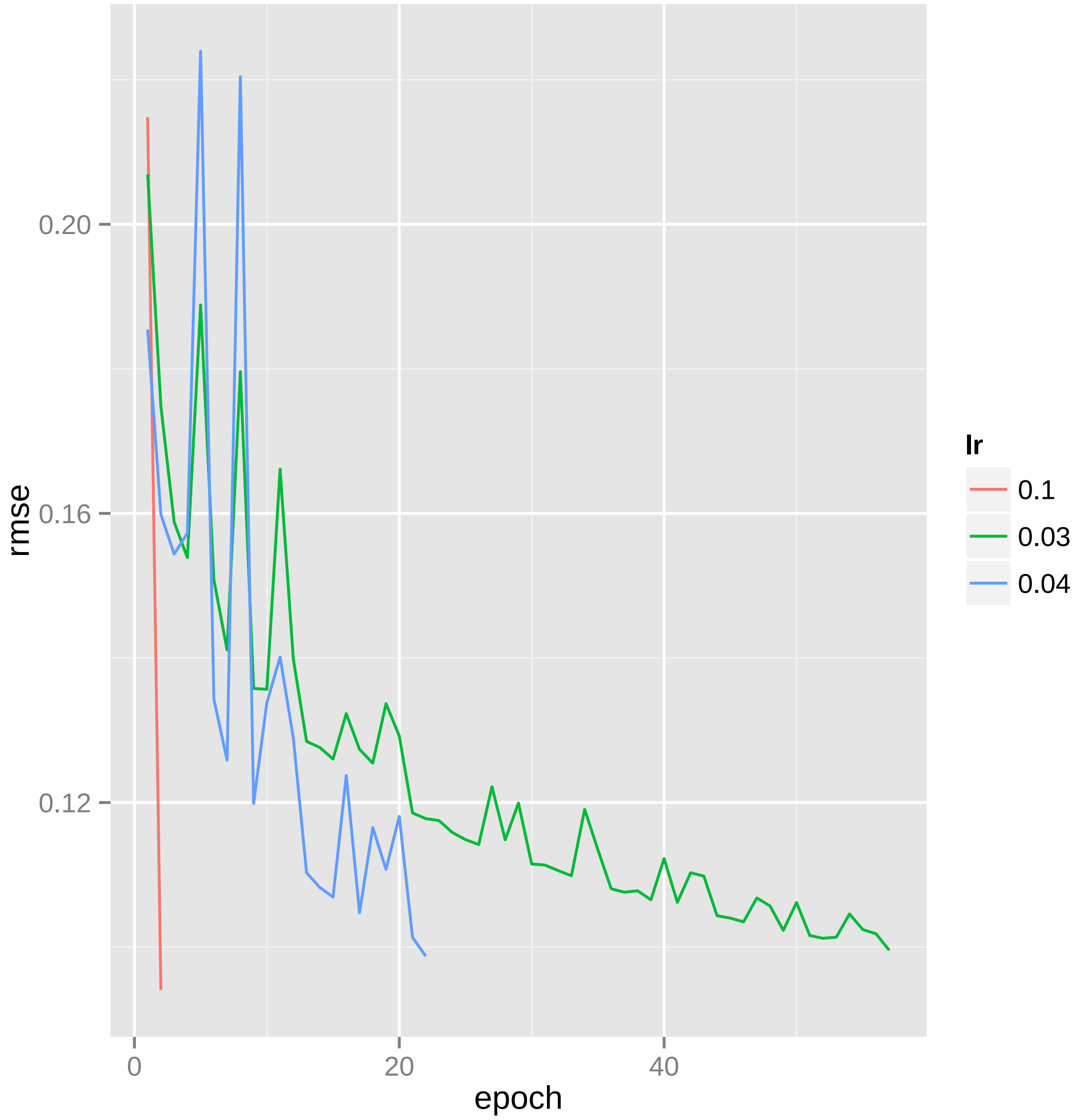
Showing history of RMSE during learning for different learning rates:



Trying those learning rates with exponential decay:

```
> trainedAdalineLR1ExpDecr<-trainAdaline(separableTrainingData$data,0.1,function(epoch){(0.1)*exp(-0.1*(epoc
> trainedAdalineLR2ExpDecr<-trainAdaline(separableTrainingData$data,0.1,function(epoch){(0.03)*exp(-0.01*(ep
> trainedAdalineLR3ExpDecr<-trainAdaline(separableTrainingData$data,0.1,function(epoch){(0.04)*exp(-0.01*(ep
```

Showing history of RMSE during learning for different learning rates with exponential decay:



It looks that one has to be careful with hyperparameters that setup exponential decay because they can make learning longer. Looks that in case of lineary separatable data there is no point in using exponential decay.

Exercise 5

Perform comparative tests of the perceptron and ADALINE on the XOR-type function approximation (not linearly separable). Stop criterion: after a reasonable number of epochs.

Function to generate XOR-type data:

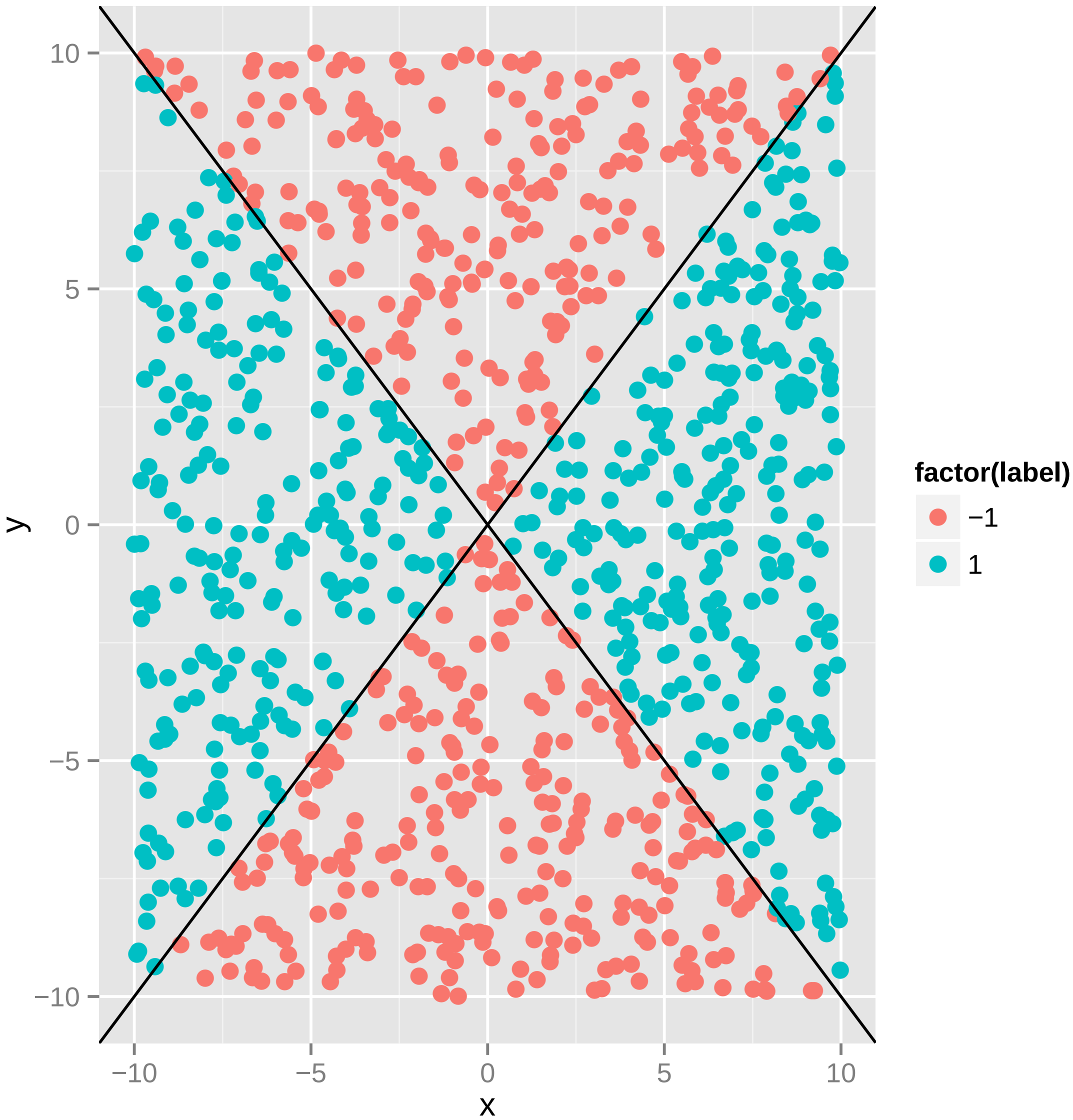
```
> #generates 2D data with 2 classes not linearly separable. 1st column is the extension of input,
> #2nd column is x coordinate, 3rd column is y coordinate, 4th column is class
> #points - number of points
> xorFunctionGenerateData<-function(points){
+   #generate matrix with data
+   data<-matrix(c(rep(1,points),runif(2*points,-10,10),rep(1,points)),ncol = 4)
+   #generate 2 lines separating 2 set of points
+   #set.seed(1025)
+   l1w0<-0
+     #runif(1,-5,-5)
+   l1w1<-1
+     #runif(1,-100,100)
+   l1w2<-1
+     #runif(1,-100,100)
+
+   l2w0<-0
+     #runif(1,-5,-5)
+   l2w1<--1
+     #runif(1,-100,100)
+   l2w2<-1
+     #runif(1,-100,100)
```

```
+
+   data<-t(apply(data,1,function(row){
+       val1 <- l1w0*row[1]+l1w1*row[2]+l1w2*row[3]
+       val2 <- l2w0*row[1]+l2w1*row[2]+l2w2*row[3]
+       if((val1>=0 && val2 <= 0) || (val1 <= 0 && val2>=0)){
+               c(row[1:3],1)
+       }else{
+         c(row[1:3],-1)
+       }
+     }))
+   colnames(data)<-c("ex","x","y","label")
+   return(list(data=data,l1w=c(l1w0,l1w1,l1w2),l2w=c(l2w0,l2w1,l2w2)))
+ }
>
```

Generate points:

```
> notSeparableTrainingData<-xorFunctionGenerateData(1000)
```

and show them on a plot with the true separating lines:



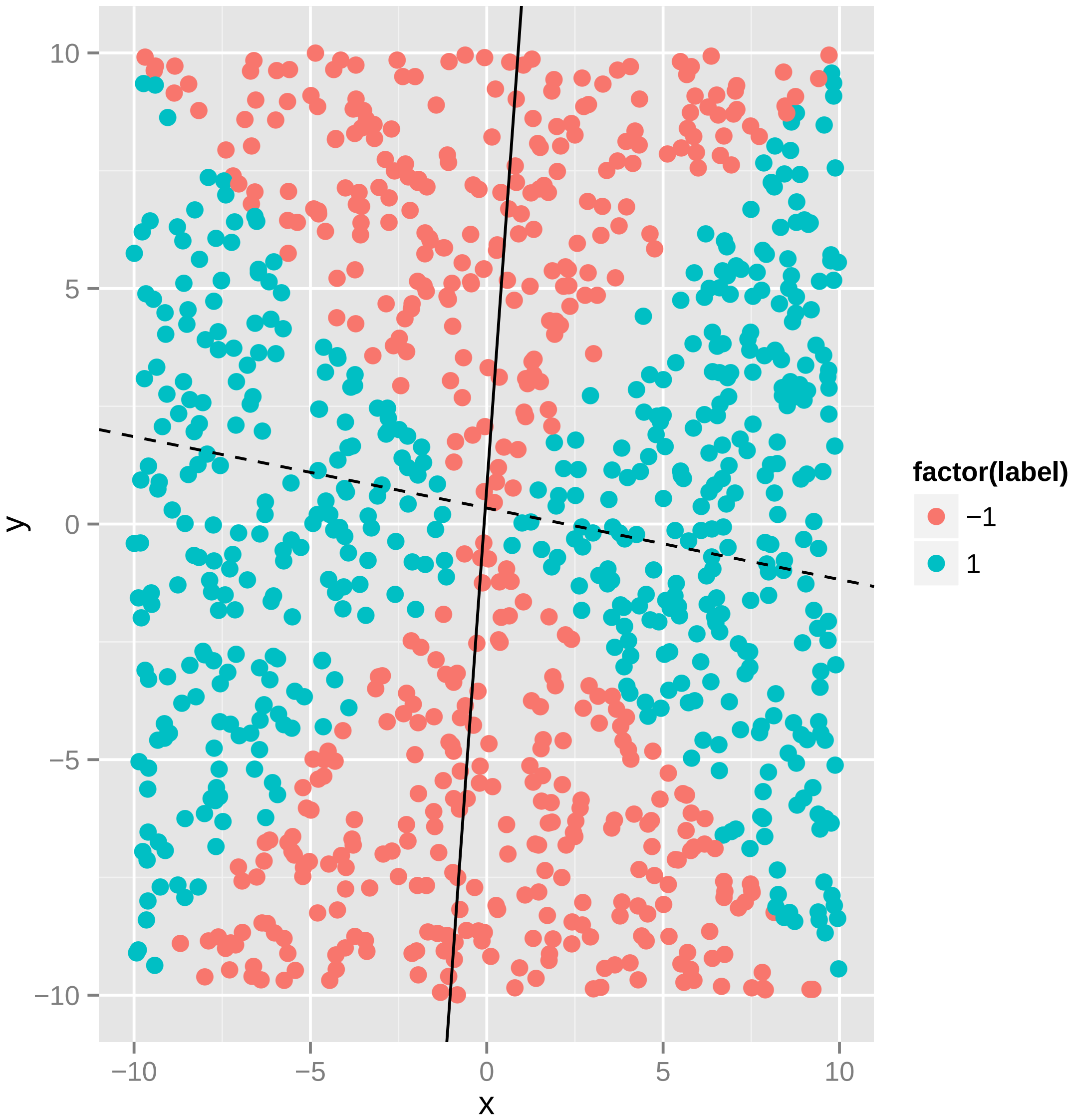
Run perceptron

```
>   trainedPerNotSep<-
+     trainPerceptron(notSeparableTrainingData$data,lerarningRateFun = function(epoch){0.01},
+       maxEpochs = 1000)
```

and adaline learning:

```
>   trainedAdalineNotSep<-
+     trainAdaline(notSeparableTrainingData$data,maxRmse =  0.1,lerarningRateFun = function(epoch){0.01},
+       maxEpochs = 1000)
```

Trained weights as a line (where solid line is adaline and dashed line is perceptron):



Classification errors for adaline and perceptron:

```
>          classficationError(function(w,x){sign(adaline(w,x))},trainedAdalineNotSep$w,notSeparableTrainingData
```

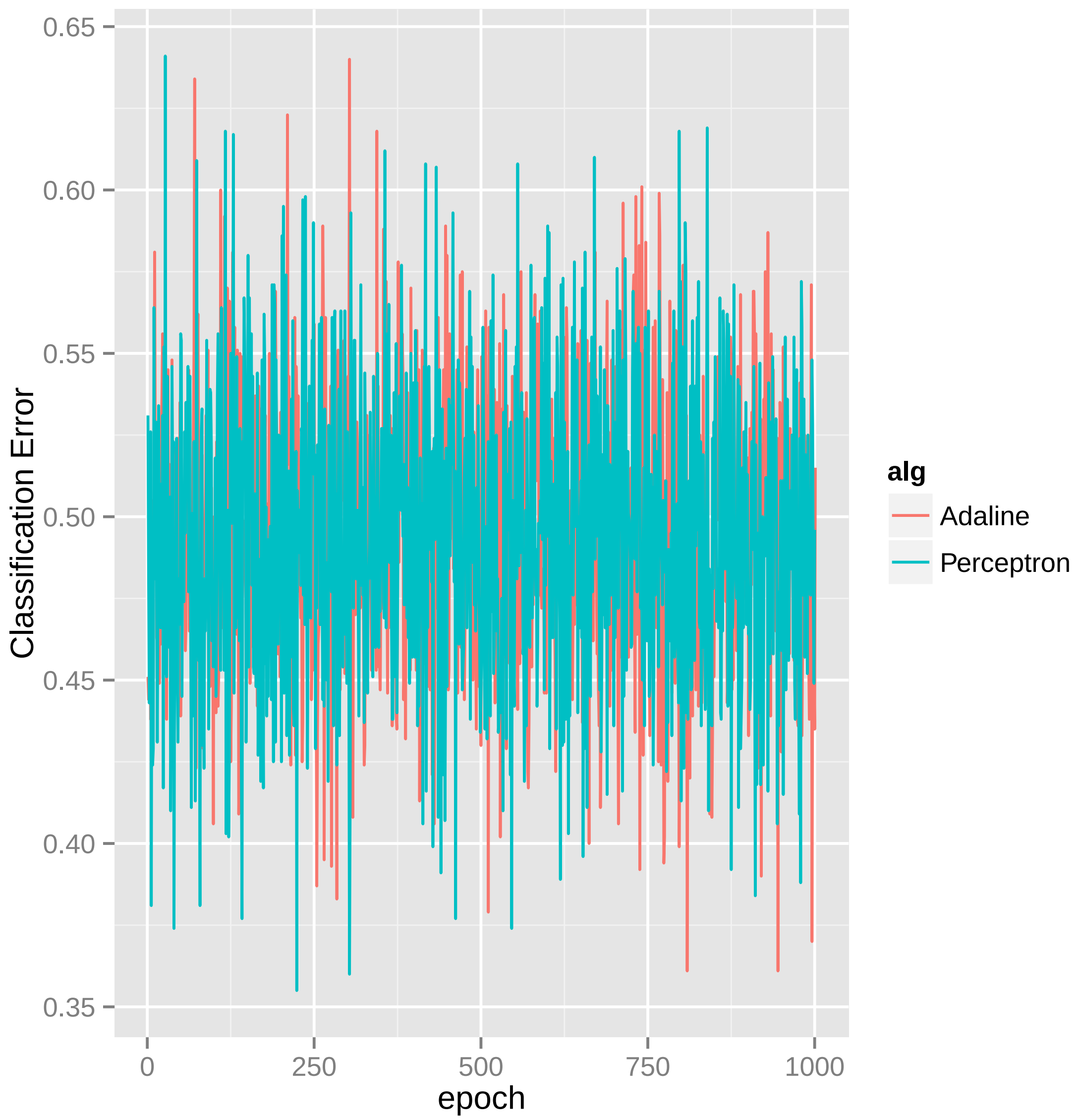
```
[1] 0.515
```

```
>            classficationError(perceptron,trainedAdalineNotSep$w,notSeparableTrainingData$data)
```

```
[1] 0.515
```

We see that two models selected different solution (possibly because of the different starting vectors but ended-up with the same classification error)

Showing history of classification error during learning for both algorithms:



The convergence also looks similar.
Here I am not checking how well models can genralize but only how well they can fit the training data.