hw4

Qiuying Li UNI ql2280

3/28/2017

setwd("~/Desktop/2017 spring/GR 5241/HW/HW4")  
setwd("~/Desktop/2017 spring/GR 5241/HW/HW4")  
library("freestats")  
  
DScost <- function(theta, Xvec, Yvec, number, weights) {  
 # calculate the cost of given theta  
 # Xvec is only one dimension X data  
 classified <- rep(-1, number)  
 classified[Xvec > theta] = 1  
 c\_d <- sum(weights\*(classified != Yvec))  
 return(c\_d)  
}  
DScostN <- function(theta, Xvec, Yvec, number, weights) {  
 # calculate the cost of given theta  
 # Xvec is only one dimension X data  
 classified <- rep(1, number)  
 classified[Xvec > theta] = -1  
 c\_d <- sum(weights\*(classified != Yvec))  
 return(c\_d)  
}  
classify <- function(X, pars) {  
 # classify X use the parameters in pars  
 # pars is the triplet (j, theta, m)  
 j <- pars[1]  
 theta <- pars[2]  
 m <- pars[3]  
 n <- nrow(X)  
 X\_d <- X[,j]  
 classified <- rep(-m, n)  
 classified[X\_d > theta] = m  
 return(matrix(classified))  
}  
  
train\_pkg <- function(X, w, Y) {  
 # use the decisionStump function with weights w  
 # return [j, theta, m]  
 dc.out <- decisionStump(X, w, Y)  
 return(c(dc.out$j, dc.out$theta, dc.out$m))  
}  
  
train <- function(X, w, Y) {  
 # produce a decision stump classifier on X, Y with weights w  
 # m can be both 1 and -1  
 # return [j, theta, m]  
 n <- nrow(X)  
 p <- ncol(X)  
 min\_cs = rep(n, p)  
 min\_thetas = rep(-2, p)  
 min\_ms = rep(1, p)  
 # compute optimal theta for each dimension  
 for (d in seq(p)) {  
 X\_d <- X[,d]  
 unique\_X\_d <- unique(X\_d)  
 # add -2 to the list to make sure every possible solution is touched  
 unique\_X\_d <- c(unique\_X\_d, -2)  
 # get costs for every possible theta  
 # costs for both m = 1 and -1  
 c\_d <- apply(matrix(unique\_X\_d), 1, DScost, Xvec=X\_d, Yvec=Y, number=n, weights=w)  
 c\_d\_n <- apply(matrix(unique\_X\_d), 1, DScostN, Xvec=X\_d, Yvec=Y, number=n, weights=w)  
 # store the minimal costs with the respecting theta for this dimension  
 if (min(c\_d\_n) < min(c\_d)) {  
 ind <- which.min(c\_d\_n)  
 min\_ms[d] <- -1  
 min\_cs[d] <- c\_d\_n[ind]  
 } else {  
 ind <- which.min(c\_d)  
 min\_cs[d] <- c\_d[ind]  
 }  
 min\_thetas[d] <- unique\_X\_d[ind]  
 }  
 # return the minimal theta with the respecting dimension  
 min\_d <- which.min(min\_cs)  
 min\_theta <- min\_thetas[min\_d]  
 min\_c <- min\_cs[min\_d]  
 min\_m <- min\_ms[min\_d]  
 return(c(min\_d, min\_theta, min\_m))  
}  
  
agg\_class <- function(X, alpha, allPars) {  
 # evaluates the boosting classifier on X  
 # return the classified result with shape n x 1  
 n <- nrow(X)  
 B <- length(alpha)  
 agg\_labels <- matrix(0, n, 1)  
 if (B == 1) {  
 # deal with the case when there is one row, the matrix indexing is no longer applicable  
 allPars <- rbind(allPars, matrix(0,1,3))  
 }  
 for (i in seq(B)) {  
 a <- alpha[i]  
 agg\_labels <- agg\_labels + a \* classify(X, allPars[i,])   
 }  
 classified <- matrix(-1, n, 1)  
 classified[agg\_labels >= 0] <- 1  
 return(classified)  
}  
  
ff\_cv <- function(X, Y, allPars, alphas, iter) {  
 fold\_size <- n / 5  
 cv\_errors <- matrix(1,5,1)  
 for (cv in seq(5)) {  
 tr\_set\_feats <- X[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 tr\_set\_labels <- Y[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 tr\_w <- w[-(((cv-1)\*fold\_size+1):(cv\*fold\_size)),]  
 cv\_set\_feats <- X[((cv-1)\*fold\_size+1):(cv\*fold\_size),]  
 cv\_set\_labels <- Y[((cv-1)\*fold\_size+1):(cv\*fold\_size),]  
 # train weak learners  
 cv\_pars <- train(tr\_set\_feats, tr\_w, tr\_set\_labels) # [j, theta, m]  
 tr\_pred\_labels <- classify(tr\_set\_feats, cv\_pars)   
 tr\_error\_rate <- sum(tr\_w\*(tr\_pred\_labels != tr\_set\_labels)) / sum(tr\_w)  
 cv\_alpha <- log((1-tr\_error\_rate)/tr\_error\_rate)  
 # using the boosting classifier so far to classify the cv set  
 # print(paste("cross validation alpha:", cv\_alpha))  
 cv\_labels <- agg\_class(cv\_set\_feats, c(alphas[0:(iter-1),], cv\_alpha), rbind(allPars[0:(iter-1),], cv\_pars))  
 # compute cv error rate  
 cv\_error <- sum(cv\_labels != cv\_set\_labels) / fold\_size  
 cv\_errors[cv,] <- cv\_error  
 }  
 cv\_avg\_error <- mean(cv\_errors)  
 return(cv\_avg\_error)  
}  
  
####################################################  
train.3 <- as.matrix(read.table("train\_3.txt", header=FALSE, sep=","))  
train.8 <- as.matrix(read.table("train\_8.txt", header=FALSE, sep=","))  
xtrain <- rbind(train.3, train.8) # 1200 x 256  
ytrain.3 <- rep(c(1,-1), c(nrow(train.3), nrow(train.8))) #1200  
ytrain.3 <- matrix(ytrain.3) # 1200 x 1  
test <- as.matrix(read.table("zip\_test.txt",header = F))  
test <- test[test[,1]%in%c(3,8),] # 332 x 257  
xtest <- test[,-1]  
ytest <- test[,1]  
ytest[ytest == 3] <- 1  
ytest[ytest == 8] <- -1  
ytest <- matrix(ytest)  
  
# perform AdaBoost   
AdaBoost <- function(B, X, Y, testX, testY){  
 # X is a nxp matrix  
 # Y is a nx1 matrix  
 # return [alphas, pars, tr\_errors, cv\_errors, test\_errors]  
 n <- nrow(X)  
 test\_size <- nrow(testX)  
 w <- matrix(1/n, n) # n x 1  
 alphas <- matrix(0, B, 1)  
 allPars <- matrix(0, B, 3)  
 errors <- matrix(0, B, 3) # training, cv and test errors  
 itercount <- 0  
 while (itercount < B) {  
 itercount = itercount + 1  
 # get 5 fold cross validation error rate  
 cv\_error <- ff\_cv(X, Y, allPars, alphas, itercount)  
 # train weak learners  
 pars <- train(X, w, Y) # [j, theta, m]  
 # use the trained weak learner to classify  
 labels <- classify(X, pars) # n x 1  
 # compute training error rate  
 error\_rate <- sum(w\*(labels != Y)) / sum(w)  
 # compute voting weights  
 alpha <- log((1-error\_rate)/error\_rate)  
 # save the classifier  
 alphas[itercount,] <- alpha  
 allPars[itercount,] <- pars  
 # recompute weights w  
 w <- w \* exp(alpha \* (labels != Y)) # n x 1  
 # calcualte test error  
 test\_labels <- agg\_class(testX, alphas[1:itercount,], allPars[1:itercount,])  
 test\_error <- sum(test\_labels != testY) / test\_size  
 errors[itercount,1] <- error\_rate  
 errors[itercount,2] <- cv\_error  
 errors[itercount,3] <- test\_error  
 print(paste0("iter ", itercount, ": j=", pars[1], ", theta=", pars[2], ", m=", pars[3], ", alpha=", round(alpha, digits=4), ", tr\_err=", round(error\_rate, digits=4), ", cv\_err=", round(cv\_error, digits=4), ", test\_err=", round(test\_error, digits=4)))  
 }  
 return(cbind(alphas, allPars, errors))  
}  
n <- nrow(xtrain)  
w <- matrix(1/n, n) # initial weights   
out <- AdaBoost(100, xtrain, ytrain.3, xtest, ytest)

## [1] "iter 1: j=167, theta=-1, m=-1, alpha=2.0163, tr\_err=0.1175, cv\_err=0.16, test\_err=0.1175"  
## [1] "iter 2: j=85, theta=-0.949, m=-1, alpha=1.5004, tr\_err=0.1824, cv\_err=0.16, test\_err=0.1175"  
## [1] "iter 3: j=173, theta=-0.98, m=1, alpha=1.5768, tr\_err=0.1713, cv\_err=0.1317, test\_err=0.0813"  
## [1] "iter 4: j=164, theta=-1, m=-1, alpha=1.0597, tr\_err=0.2574, cv\_err=0.135, test\_err=0.0813"  
## [1] "iter 5: j=136, theta=-0.484, m=-1, alpha=1.0899, tr\_err=0.2516, cv\_err=0.0942, test\_err=0.0813"  
## [1] "iter 6: j=70, theta=-0.978, m=-1, alpha=0.8583, tr\_err=0.2977, cv\_err=0.1017, test\_err=0.0843"  
## [1] "iter 7: j=41, theta=-0.348, m=1, alpha=0.9779, tr\_err=0.2733, cv\_err=0.0992, test\_err=0.0602"  
## [1] "iter 8: j=84, theta=-0.979, m=-1, alpha=0.7671, tr\_err=0.3171, cv\_err=0.0817, test\_err=0.0663"  
## [1] "iter 9: j=141, theta=-0.994, m=1, alpha=0.9207, tr\_err=0.2848, cv\_err=0.0733, test\_err=0.0663"  
## [1] "iter 10: j=204, theta=-0.501, m=1, alpha=0.7812, tr\_err=0.3141, cv\_err=0.0675, test\_err=0.0693"  
## [1] "iter 11: j=165, theta=-0.619, m=-1, alpha=0.7458, tr\_err=0.3217, cv\_err=0.0608, test\_err=0.0663"  
## [1] "iter 12: j=171, theta=0.941, m=1, alpha=0.8847, tr\_err=0.2922, cv\_err=0.06, test\_err=0.0633"  
## [1] "iter 13: j=183, theta=-1, m=-1, alpha=0.8408, tr\_err=0.3014, cv\_err=0.0508, test\_err=0.0602"  
## [1] "iter 14: j=69, theta=-1, m=-1, alpha=0.8283, tr\_err=0.304, cv\_err=0.0583, test\_err=0.0572"  
## [1] "iter 15: j=100, theta=-0.997, m=-1, alpha=0.7299, tr\_err=0.3252, cv\_err=0.0508, test\_err=0.0693"  
## [1] "iter 16: j=222, theta=-0.926, m=1, alpha=0.8573, tr\_err=0.2979, cv\_err=0.0458, test\_err=0.0602"  
## [1] "iter 17: j=218, theta=0.995, m=1, alpha=0.758, tr\_err=0.3191, cv\_err=0.0433, test\_err=0.0572"  
## [1] "iter 18: j=148, theta=-0.837, m=-1, alpha=0.771, tr\_err=0.3163, cv\_err=0.0392, test\_err=0.0542"  
## [1] "iter 19: j=153, theta=-0.992, m=-1, alpha=0.6859, tr\_err=0.3349, cv\_err=0.0425, test\_err=0.0572"  
## [1] "iter 20: j=87, theta=-0.18, m=-1, alpha=0.8428, tr\_err=0.3009, cv\_err=0.04, test\_err=0.0602"  
## [1] "iter 21: j=119, theta=0.355, m=-1, alpha=0.6886, tr\_err=0.3344, cv\_err=0.0425, test\_err=0.0572"  
## [1] "iter 22: j=188, theta=-0.794, m=1, alpha=0.6987, tr\_err=0.3321, cv\_err=0.0375, test\_err=0.0542"  
## [1] "iter 23: j=211, theta=-0.997, m=1, alpha=0.5996, tr\_err=0.3544, cv\_err=0.0325, test\_err=0.0542"  
## [1] "iter 24: j=51, theta=-0.955, m=-1, alpha=0.7876, tr\_err=0.3127, cv\_err=0.03, test\_err=0.0482"  
## [1] "iter 25: j=249, theta=-0.516, m=-1, alpha=0.7045, tr\_err=0.3308, cv\_err=0.03, test\_err=0.0482"  
## [1] "iter 26: j=168, theta=-0.402, m=-1, alpha=0.7446, tr\_err=0.322, cv\_err=0.0267, test\_err=0.0482"  
## [1] "iter 27: j=135, theta=0.259, m=-1, alpha=0.5972, tr\_err=0.355, cv\_err=0.0308, test\_err=0.0542"  
## [1] "iter 28: j=219, theta=0.959, m=1, alpha=0.5986, tr\_err=0.3547, cv\_err=0.0242, test\_err=0.0482"  
## [1] "iter 29: j=72, theta=0.096, m=-1, alpha=0.6309, tr\_err=0.3473, cv\_err=0.0242, test\_err=0.0512"  
## [1] "iter 30: j=53, theta=-0.998, m=-1, alpha=0.7285, tr\_err=0.3255, cv\_err=0.0225, test\_err=0.0542"  
## [1] "iter 31: j=156, theta=-0.516, m=1, alpha=0.663, tr\_err=0.3401, cv\_err=0.0225, test\_err=0.0602"  
## [1] "iter 32: j=87, theta=-0.884, m=-1, alpha=0.6381, tr\_err=0.3457, cv\_err=0.0183, test\_err=0.0512"  
## [1] "iter 33: j=165, theta=-0.186, m=-1, alpha=0.5997, tr\_err=0.3544, cv\_err=0.02, test\_err=0.0572"  
## [1] "iter 34: j=138, theta=-0.897, m=-1, alpha=0.6958, tr\_err=0.3327, cv\_err=0.0175, test\_err=0.0452"  
## [1] "iter 35: j=83, theta=-0.907, m=-1, alpha=0.7031, tr\_err=0.3311, cv\_err=0.0208, test\_err=0.0542"  
## [1] "iter 36: j=55, theta=0.997, m=1, alpha=0.7427, tr\_err=0.3224, cv\_err=0.0158, test\_err=0.0482"  
## [1] "iter 37: j=167, theta=-0.575, m=-1, alpha=0.6834, tr\_err=0.3355, cv\_err=0.015, test\_err=0.0572"  
## [1] "iter 38: j=44, theta=0.957, m=1, alpha=0.6155, tr\_err=0.3508, cv\_err=0.0192, test\_err=0.0512"  
## [1] "iter 39: j=163, theta=-0.073, m=-1, alpha=0.6234, tr\_err=0.349, cv\_err=0.0167, test\_err=0.0542"  
## [1] "iter 40: j=221, theta=-0.206, m=1, alpha=0.6933, tr\_err=0.3333, cv\_err=0.0142, test\_err=0.0512"  
## [1] "iter 41: j=216, theta=0.199, m=1, alpha=0.5756, tr\_err=0.36, cv\_err=0.0158, test\_err=0.0452"  
## [1] "iter 42: j=184, theta=0.162, m=-1, alpha=0.7221, tr\_err=0.3269, cv\_err=0.0125, test\_err=0.0482"  
## [1] "iter 43: j=74, theta=0.928, m=1, alpha=0.5931, tr\_err=0.3559, cv\_err=0.0142, test\_err=0.0392"  
## [1] "iter 44: j=165, theta=-0.186, m=-1, alpha=0.6523, tr\_err=0.3425, cv\_err=0.0058, test\_err=0.0482"  
## [1] "iter 45: j=54, theta=-1, m=-1, alpha=0.5999, tr\_err=0.3544, cv\_err=0.0083, test\_err=0.0392"  
## [1] "iter 46: j=172, theta=-0.038, m=1, alpha=0.657, tr\_err=0.3414, cv\_err=0.0067, test\_err=0.0482"  
## [1] "iter 47: j=101, theta=-0.669, m=-1, alpha=0.6335, tr\_err=0.3467, cv\_err=0.005, test\_err=0.0422"  
## [1] "iter 48: j=158, theta=-0.716, m=1, alpha=0.7225, tr\_err=0.3268, cv\_err=0.0033, test\_err=0.0452"  
## [1] "iter 49: j=182, theta=-0.732, m=-1, alpha=0.6039, tr\_err=0.3535, cv\_err=0.005, test\_err=0.0392"  
## [1] "iter 50: j=51, theta=-0.886, m=-1, alpha=0.6648, tr\_err=0.3397, cv\_err=0.0067, test\_err=0.0422"  
## [1] "iter 51: j=26, theta=0.999, m=1, alpha=0.5946, tr\_err=0.3556, cv\_err=0.0067, test\_err=0.0392"  
## [1] "iter 52: j=55, theta=-0.788, m=-1, alpha=0.5115, tr\_err=0.3749, cv\_err=0.005, test\_err=0.0452"  
## [1] "iter 53: j=84, theta=-0.953, m=-1, alpha=0.5663, tr\_err=0.3621, cv\_err=0.0058, test\_err=0.0452"  
## [1] "iter 54: j=40, theta=0.973, m=1, alpha=0.6395, tr\_err=0.3454, cv\_err=0.0025, test\_err=0.0452"  
## [1] "iter 55: j=246, theta=-0.584, m=1, alpha=0.6751, tr\_err=0.3374, cv\_err=0.0033, test\_err=0.0482"  
## [1] "iter 56: j=37, theta=0.123, m=-1, alpha=0.5822, tr\_err=0.3584, cv\_err=0.0025, test\_err=0.0482"  
## [1] "iter 57: j=216, theta=-0.375, m=1, alpha=0.5536, tr\_err=0.365, cv\_err=8e-04, test\_err=0.0482"  
## [1] "iter 58: j=183, theta=-0.997, m=-1, alpha=0.6505, tr\_err=0.3429, cv\_err=8e-04, test\_err=0.0452"  
## [1] "iter 59: j=118, theta=-0.295, m=-1, alpha=0.5687, tr\_err=0.3615, cv\_err=8e-04, test\_err=0.0452"  
## [1] "iter 60: j=171, theta=0.986, m=1, alpha=0.498, tr\_err=0.378, cv\_err=8e-04, test\_err=0.0482"  
## [1] "iter 61: j=152, theta=-0.333, m=-1, alpha=0.5787, tr\_err=0.3592, cv\_err=8e-04, test\_err=0.0452"  
## [1] "iter 62: j=88, theta=-0.597, m=-1, alpha=0.5926, tr\_err=0.356, cv\_err=8e-04, test\_err=0.0452"  
## [1] "iter 63: j=135, theta=0.259, m=-1, alpha=0.5649, tr\_err=0.3624, cv\_err=8e-04, test\_err=0.0422"  
## [1] "iter 64: j=10, theta=0.125, m=1, alpha=0.583, tr\_err=0.3582, cv\_err=0, test\_err=0.0542"  
## [1] "iter 65: j=9, theta=0.387, m=-1, alpha=0.571, tr\_err=0.361, cv\_err=8e-04, test\_err=0.0422"  
## [1] "iter 66: j=71, theta=-0.616, m=-1, alpha=0.5463, tr\_err=0.3667, cv\_err=0, test\_err=0.0512"  
## [1] "iter 67: j=84, theta=-0.931, m=-1, alpha=0.5414, tr\_err=0.3679, cv\_err=0, test\_err=0.0452"  
## [1] "iter 68: j=140, theta=0.691, m=1, alpha=0.6468, tr\_err=0.3437, cv\_err=0, test\_err=0.0452"  
## [1] "iter 69: j=148, theta=-0.519, m=-1, alpha=0.5604, tr\_err=0.3635, cv\_err=0, test\_err=0.0422"  
## [1] "iter 70: j=248, theta=-0.727, m=-1, alpha=0.604, tr\_err=0.3534, cv\_err=0, test\_err=0.0422"  
## [1] "iter 71: j=9, theta=-0.101, m=1, alpha=0.5936, tr\_err=0.3558, cv\_err=0, test\_err=0.0452"  
## [1] "iter 72: j=107, theta=0.417, m=1, alpha=0.538, tr\_err=0.3687, cv\_err=0, test\_err=0.0482"  
## [1] "iter 73: j=106, theta=-0.825, m=-1, alpha=0.5381, tr\_err=0.3686, cv\_err=0, test\_err=0.0482"  
## [1] "iter 74: j=153, theta=0.655, m=-1, alpha=0.6144, tr\_err=0.3511, cv\_err=0, test\_err=0.0452"  
## [1] "iter 75: j=58, theta=-0.972, m=1, alpha=0.5264, tr\_err=0.3714, cv\_err=0, test\_err=0.0452"  
## [1] "iter 76: j=61, theta=0.066, m=1, alpha=0.5453, tr\_err=0.3669, cv\_err=0, test\_err=0.0482"  
## [1] "iter 77: j=165, theta=-0.186, m=-1, alpha=0.5411, tr\_err=0.3679, cv\_err=0, test\_err=0.0392"  
## [1] "iter 78: j=75, theta=0.823, m=1, alpha=0.5899, tr\_err=0.3567, cv\_err=0, test\_err=0.0422"  
## [1] "iter 79: j=101, theta=-0.59, m=-1, alpha=0.5298, tr\_err=0.3706, cv\_err=0, test\_err=0.0422"  
## [1] "iter 80: j=54, theta=-1, m=-1, alpha=0.6733, tr\_err=0.3378, cv\_err=0, test\_err=0.0452"  
## [1] "iter 81: j=183, theta=-0.935, m=-1, alpha=0.5966, tr\_err=0.3551, cv\_err=0, test\_err=0.0512"  
## [1] "iter 82: j=216, theta=0.199, m=1, alpha=0.6147, tr\_err=0.351, cv\_err=0, test\_err=0.0482"  
## [1] "iter 83: j=163, theta=-0.073, m=-1, alpha=0.61, tr\_err=0.3521, cv\_err=0, test\_err=0.0422"  
## [1] "iter 84: j=236, theta=0.981, m=1, alpha=0.6899, tr\_err=0.3341, cv\_err=0, test\_err=0.0452"  
## [1] "iter 85: j=72, theta=0.302, m=-1, alpha=0.5587, tr\_err=0.3639, cv\_err=0, test\_err=0.0452"  
## [1] "iter 86: j=156, theta=-0.402, m=1, alpha=0.5204, tr\_err=0.3727, cv\_err=0, test\_err=0.0452"  
## [1] "iter 87: j=166, theta=-0.838, m=-1, alpha=0.5547, tr\_err=0.3648, cv\_err=0, test\_err=0.0452"  
## [1] "iter 88: j=249, theta=-0.043, m=-1, alpha=0.6019, tr\_err=0.3539, cv\_err=0, test\_err=0.0482"  
## [1] "iter 89: j=83, theta=-0.84, m=-1, alpha=0.6809, tr\_err=0.3361, cv\_err=0, test\_err=0.0512"  
## [1] "iter 90: j=246, theta=0.141, m=1, alpha=0.5988, tr\_err=0.3546, cv\_err=0, test\_err=0.0482"  
## [1] "iter 91: j=85, theta=-0.622, m=-1, alpha=0.5182, tr\_err=0.3733, cv\_err=0, test\_err=0.0482"  
## [1] "iter 92: j=251, theta=-0.83, m=1, alpha=0.5607, tr\_err=0.3634, cv\_err=0, test\_err=0.0422"  
## [1] "iter 93: j=22, theta=-0.594, m=-1, alpha=0.5984, tr\_err=0.3547, cv\_err=0, test\_err=0.0452"  
## [1] "iter 94: j=229, theta=-0.985, m=1, alpha=0.5805, tr\_err=0.3588, cv\_err=0, test\_err=0.0422"  
## [1] "iter 95: j=215, theta=0.881, m=1, alpha=0.5943, tr\_err=0.3556, cv\_err=0, test\_err=0.0422"  
## [1] "iter 96: j=164, theta=-1, m=-1, alpha=0.594, tr\_err=0.3557, cv\_err=0, test\_err=0.0452"  
## [1] "iter 97: j=141, theta=-0.994, m=1, alpha=0.5846, tr\_err=0.3579, cv\_err=0, test\_err=0.0482"  
## [1] "iter 98: j=87, theta=-0.989, m=-1, alpha=0.5427, tr\_err=0.3676, cv\_err=0, test\_err=0.0452"  
## [1] "iter 99: j=204, theta=-0.71, m=1, alpha=0.6198, tr\_err=0.3498, cv\_err=0, test\_err=0.0482"  
## [1] "iter 100: j=136, theta=-0.641, m=-1, alpha=0.573, tr\_err=0.3605, cv\_err=0, test\_err=0.0482"

pdf("1.pdf",width=6.5,height=4)  
par(mfrow=c(1,1),oma=c(2,2,2,1), pty='m',mar=c(1,3,0,1) ,mgp=c(1.5,0.25,0), lwd=0.5,tck=-0.01, cex.axis=0.5, cex.lab=0.6, cex.main=0.6)  
plot(seq(100), out[,7], type='n', ylim=c(0, 0.2), xlab="", ylab=" ", main=" ",axes=F)  
lines(seq(100),out[,6], col="green")  
lines(seq(100),out[,7], col="red")  
axis(1,lwd=0.5)  
axis(2,lwd=0.5)  
mtext(1,text="iteration",cex=0.6,line = 1)  
mtext(2,text="error rate",cex=0.6,line = 1)  
  
box(lwd=0.5)  
legend("topright", c("training error", "testing error"), lty=c(1,1), col=c("green", "red"),cex=0.7)  
dev.off()

## quartz\_off\_screen   
## 2