Code

Project 3 - Baseline

Yixin Zhang, Oded Loewenstein

This file is for baseline. The improved model and its code is demonstrated in another ipynb file.

Hide

```
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
if(!require("gbm")){
  install.packages("gbm")
}
library("EBImage")
library("gbm")
```

Step 0: specify directories.

Set the working directory to the image folder. Specify the training and the testing set. For data without an independent test/validation set, you need to create your own testing data by random subsampling. In order to obain reproducible results, set.seed() whenever randomization is used.

Hide

```
set.seed(2018)
setwd("../doc")
```

Provide directories for training images. Low-resolution (LR) image set and High-resolution (HR) image set will be in different subfolders.

Hide

```
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_LR_dir <- paste(train_dir, "LR/", sep="")
train_HR_dir <- paste(train_dir, "HR/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

Hide

```
run.cv=F # run cross-validation on the training set
K <- 3  # number of CV folds
run.feature.train=TRUE # process features for training set
run.test=TRUE # run evaluation on an independent test set
run.feature.test=TRUE # process features for test set
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this example, we use GBM with different `depth`. In the following chunk, we list, in a vector, setups (in this case, `depth`) corresponding to models that we will compare (depth = 1 and depth = 2). We also set shrinkage = 0.1 in our gbm fit.

Hide

```
model_values <- c(1,2)
model_labels = paste("GBM with depth =", model_values)
```

Step 2: construct features and responses

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later. + `feature.R` + Input: a path for low-resolution images. + Input: a path for high-resolution images. + Output: an RData file that contains extracted features and corresponding responses

Hide

```
source("../lib/feature.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(train_LR_dir, train_HR_dir))
  feat_train <- dat_train$feature
  label_train <- dat_train$label
  }
save(dat_train, file="../output/feature_train.RData")
```

Step 3: Train a classification model with training images

Call the train model and test model from library.

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps. + `train.R` + Input: a path that points to the training set features and responses. + Output: an RData file that contains trained classifiers in the forms of R objects: models/settings/links to external trained configurations. + `test.R` + Input: a path that points to the test set features. + Input: an R object that contains a trained classifier. + Output: an R object of response predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

Hide

```
source("../lib/train.R")
source("../lib/test.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in our model.

Hide

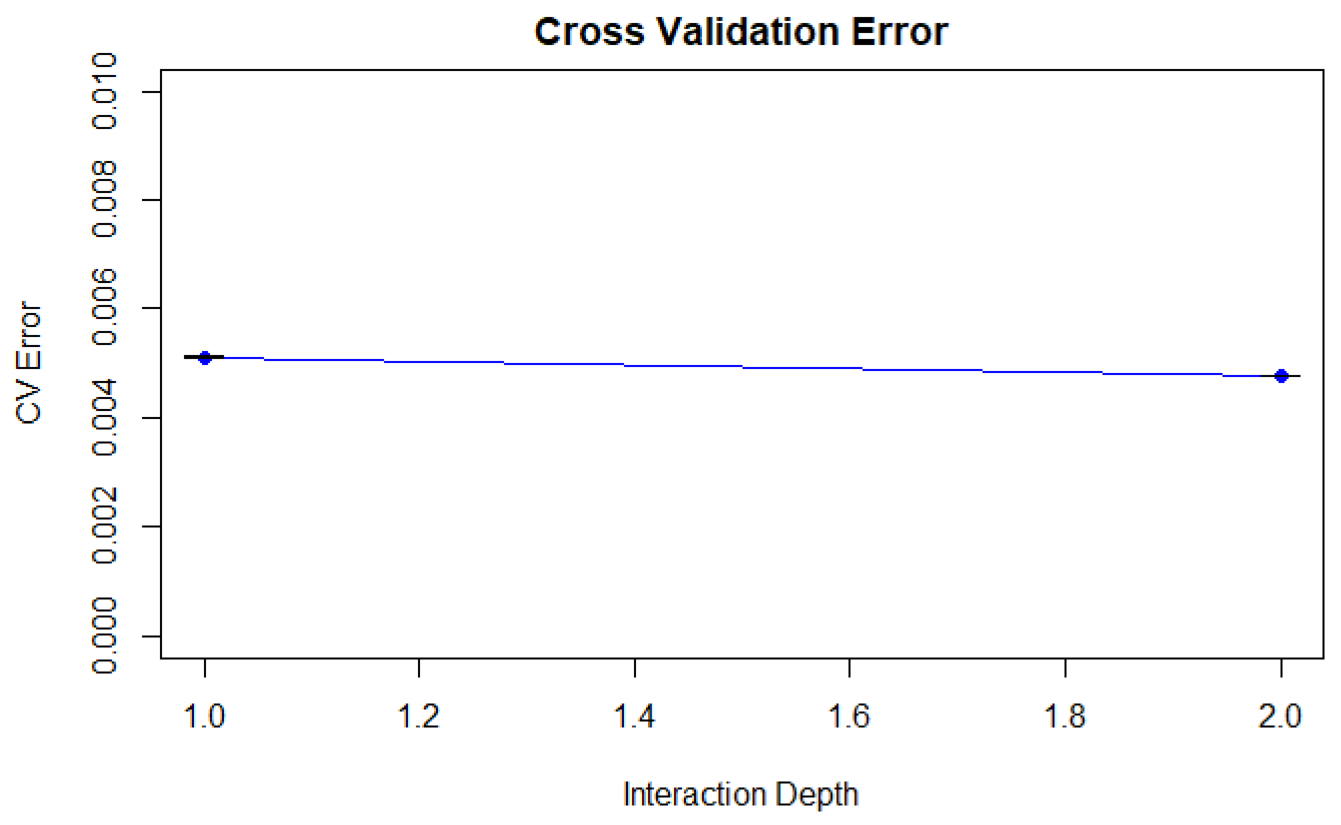
```
source("../lib/cross_validation.R")
if(run.cv){
  err_cv <- array(dim=c(length(model_values), 2))
  for(k in 1:length(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(feat_train, label_train, model_values[k], K)
  }
  save(err_cv, file="../output/err_cv.RData")
}
err_cv
```

```
            [,1]         [,2]
[1,] 0.005109547 1.123163e-05
[2,] 0.004760687 8.036916e-06
```

Visualize cross-validation results.

Hide

```
if(run.cv){
  load("../output/err_cv.RData")
  plot(model_values, err_cv[,1], xlab="Interaction Depth", ylab="CV Error",
       main="Cross Validation Error", type="n", ylim=c(0, 0.01))
  points(model_values, err_cv[,1], col="blue", pch=16)
  lines(model_values, err_cv[,1], col="blue")
  arrows(model_values, err_cv[,1]-err_cv[,2], model_values, err_cv[,1]+err_cv[,2],
        length=0.1, angle=90, code=3)
}
```



- Choose the “best”" parameter value We chose depth = 2 based on the MSE. (To avoid rerun cross-validation part, we set depth=2 manually.)

Hide

```
model_best=model_values[1]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1])]
}
par_best <- list(depth=model_best)
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

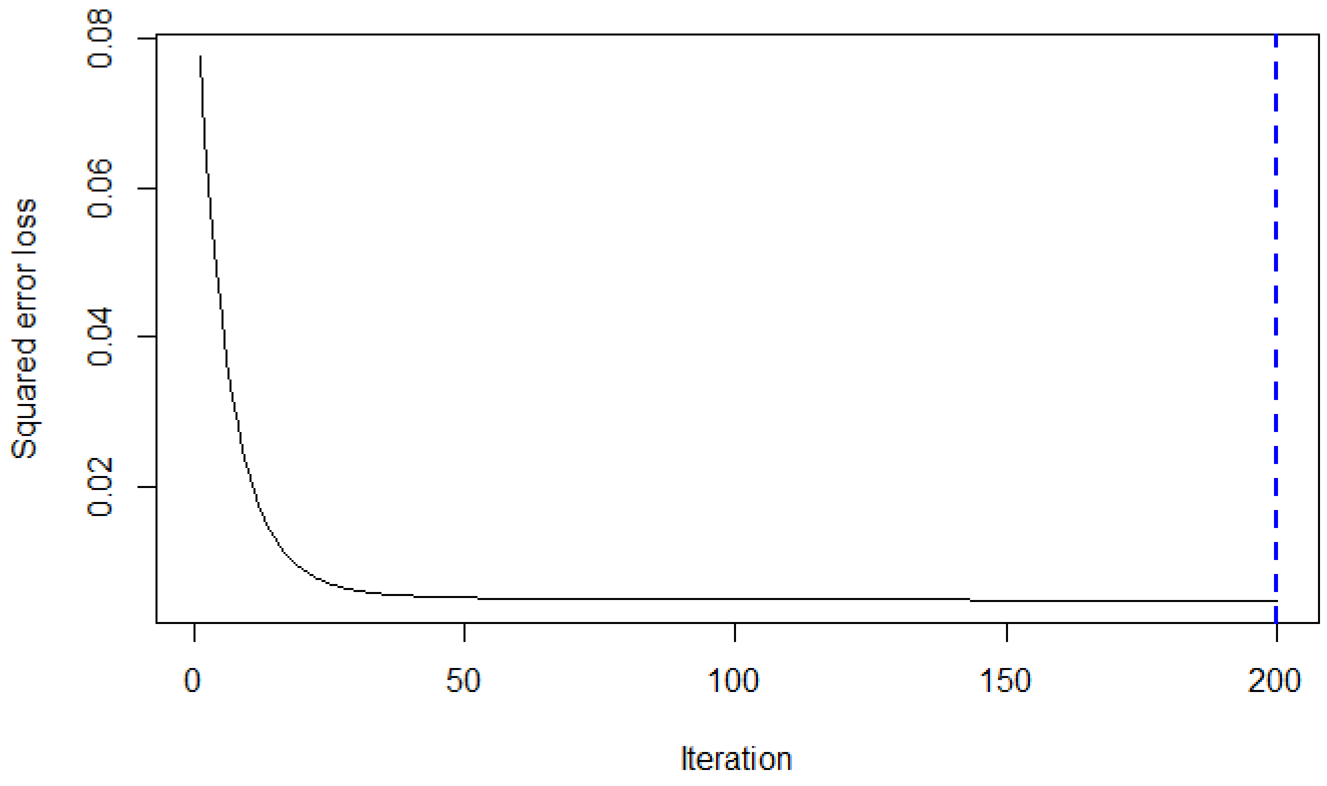
Hide

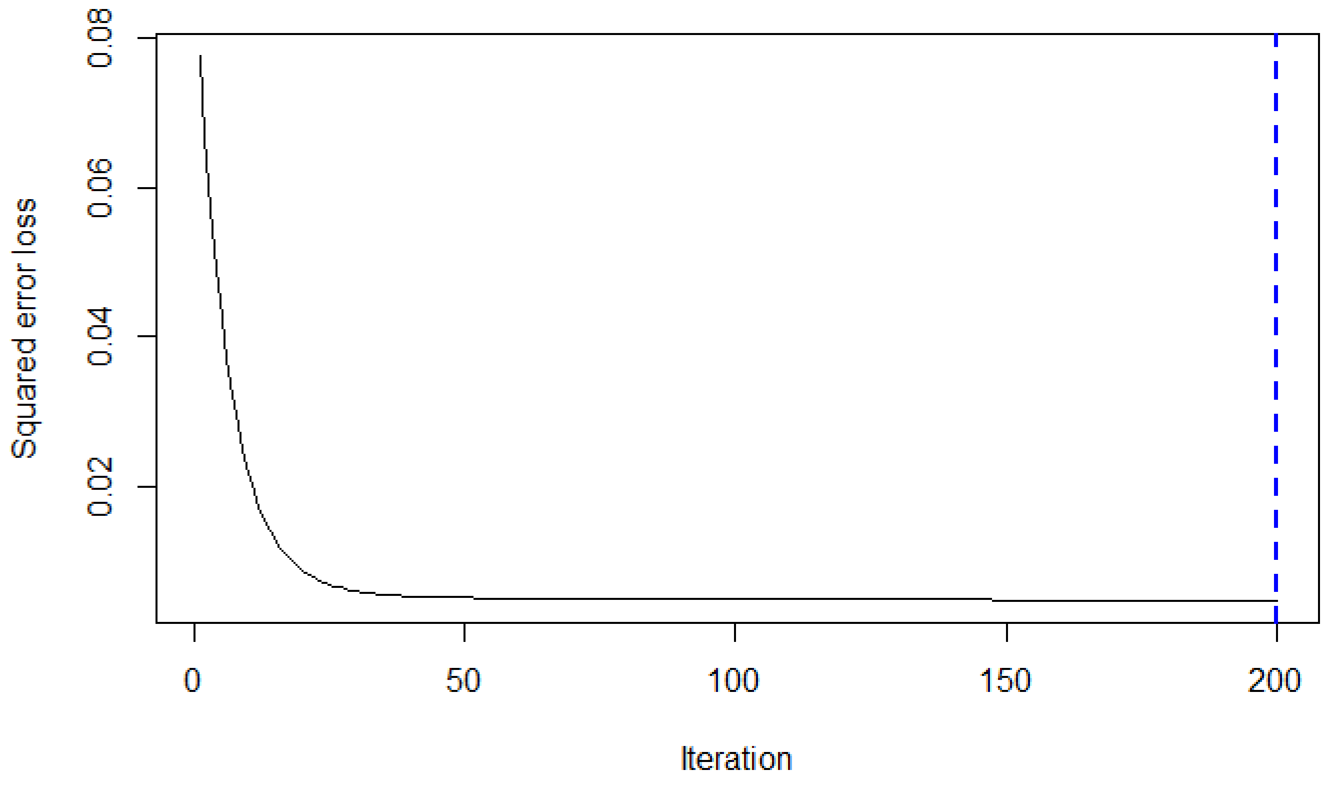
```
tm_train=NA
```

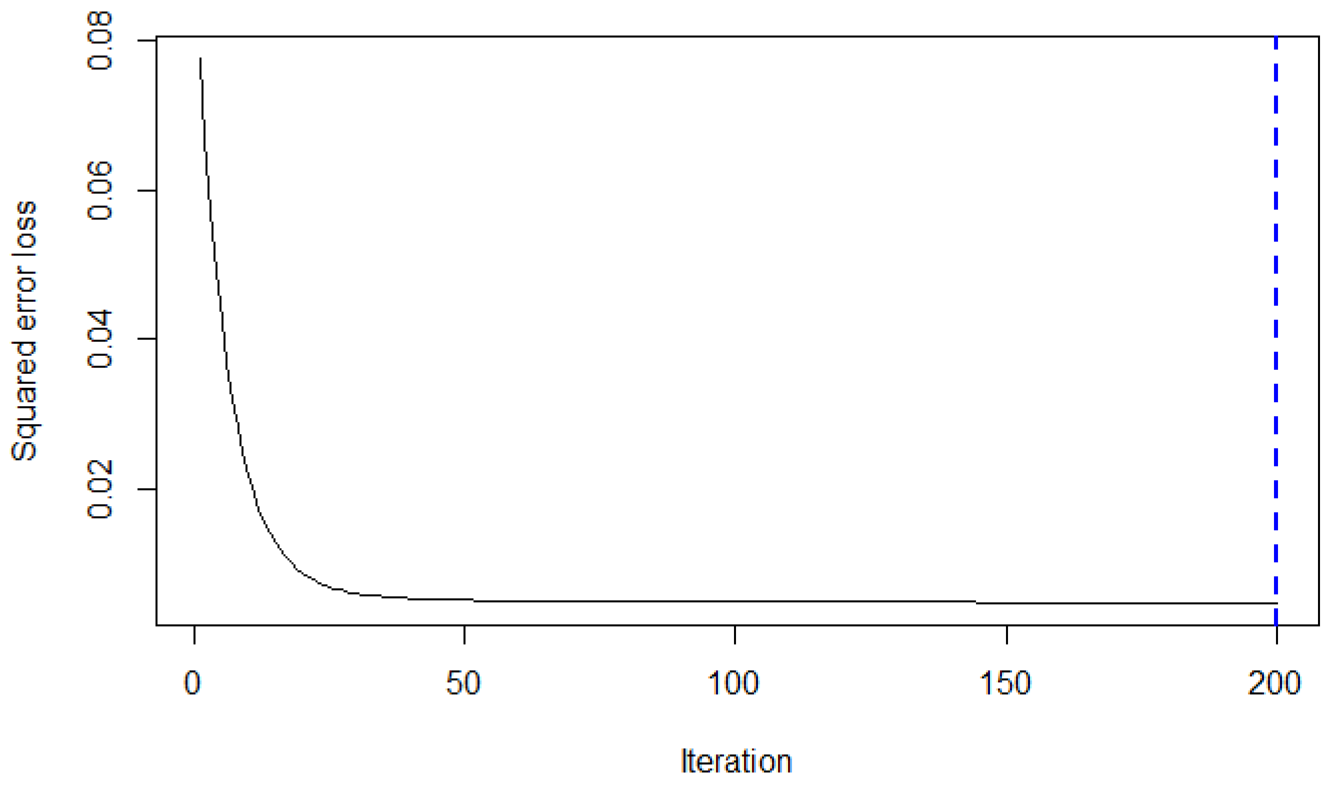
```
Warning messages:
1: In grDevices::png(f) :
  unable to open file 'C:\Users\yz322\AppData\Local\Temp\RtmpUtbW58\file2d1c49425036' for writing
2: In grDevices::png(f) : opening device failed
3: In grDevices::png(f) :
  unable to open file 'C:\Users\yz322\AppData\Local\Temp\RtmpUtbW58\file2d1c5cd01834' for writing
4: In grDevices::png(f) : opening device failed
5: In file(con, "w") :
  cannot open file 'C:\Users\yz322\AppData\Local\Temp\RtmpUtbW58\rmarkdown-str2d1c2df23141.html': No such
file or directory
```

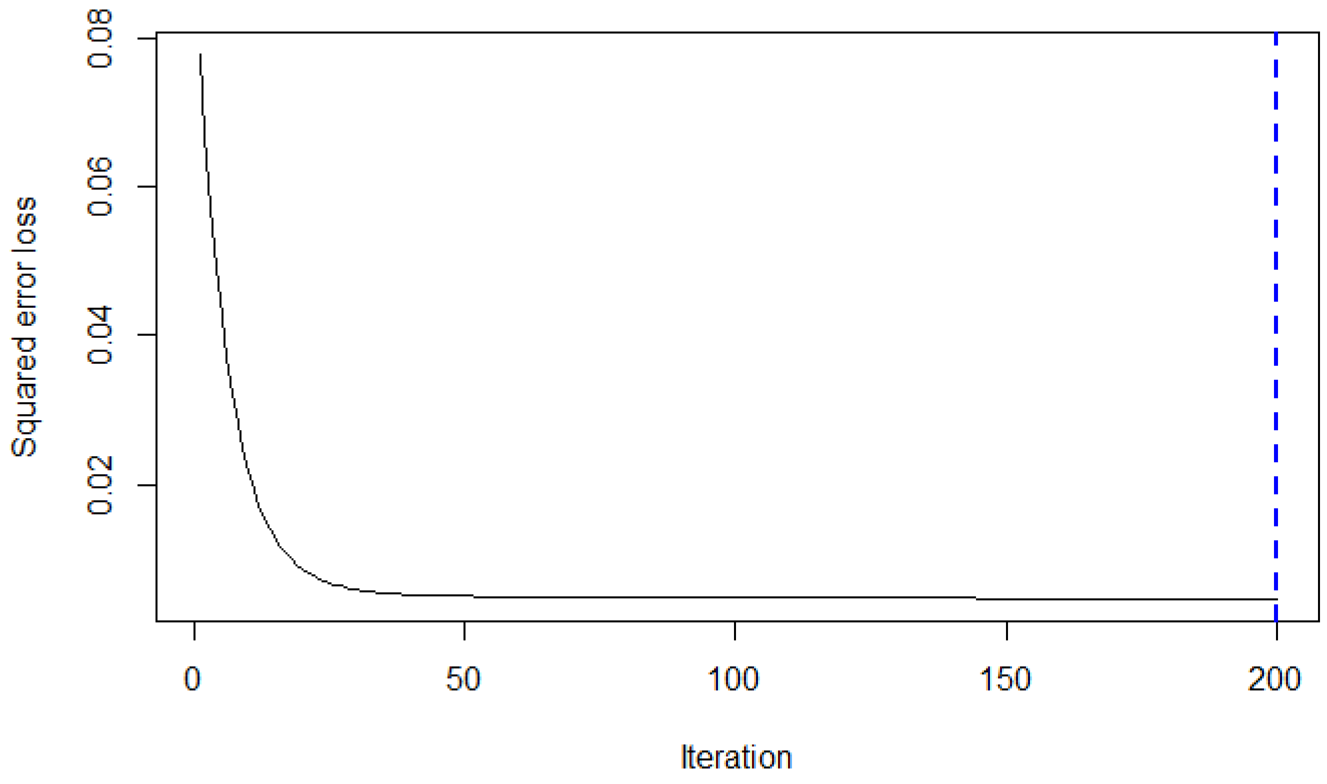
Hide

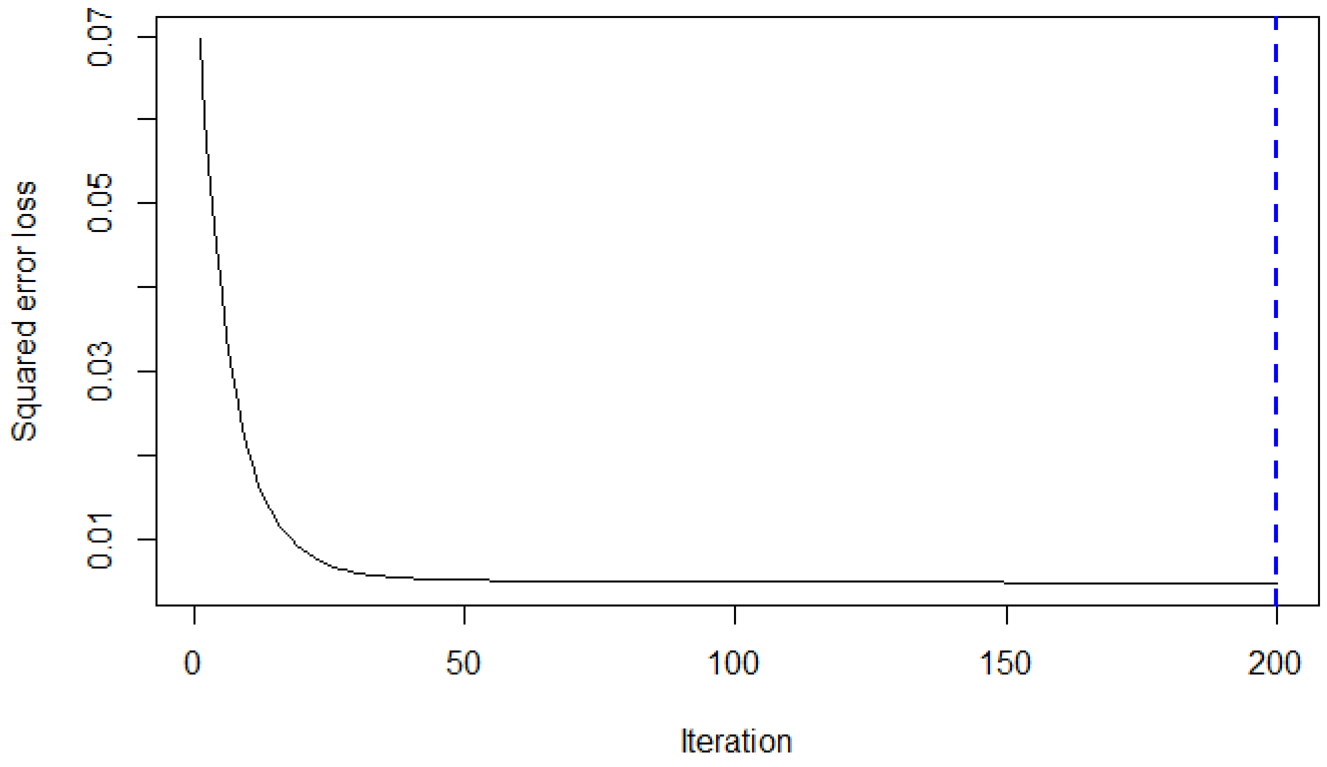
```
tm_train <- system.time(fit_train <- train(feat_train, label_train, par_best))
```

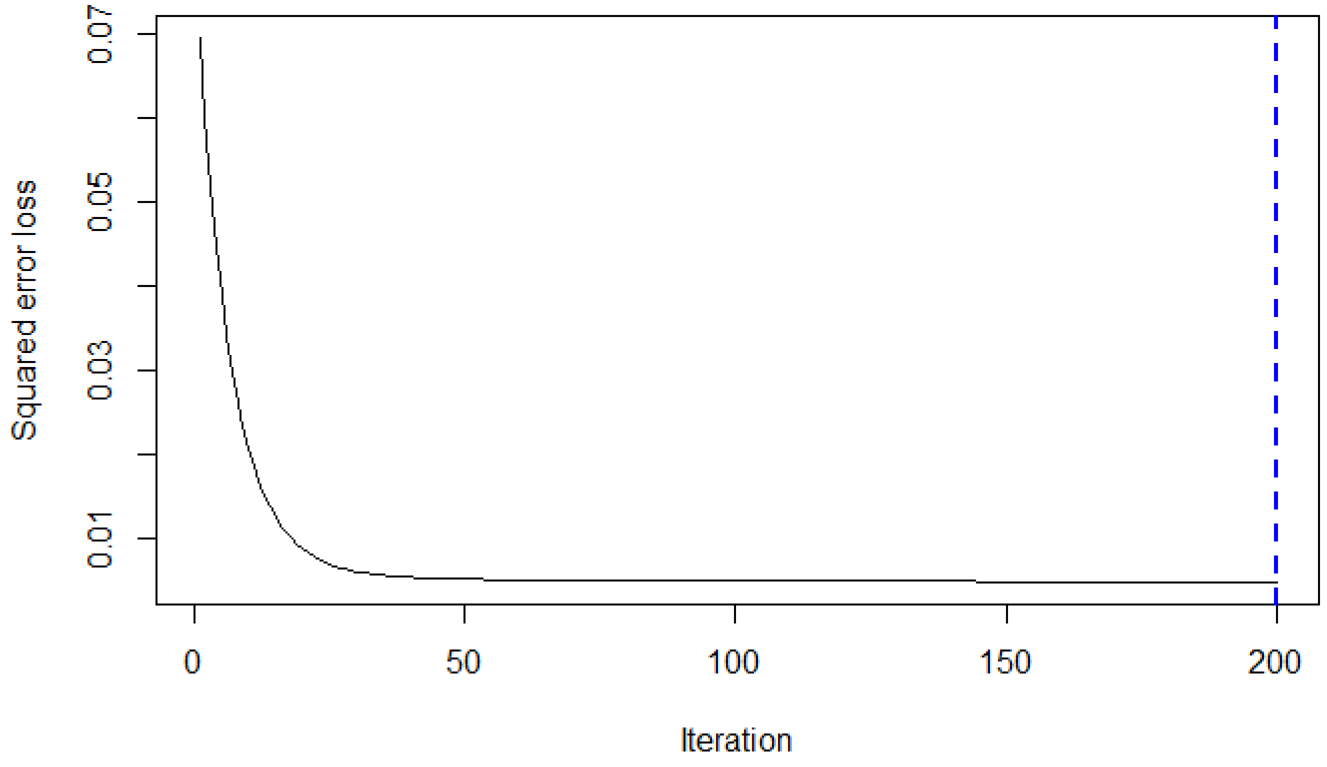


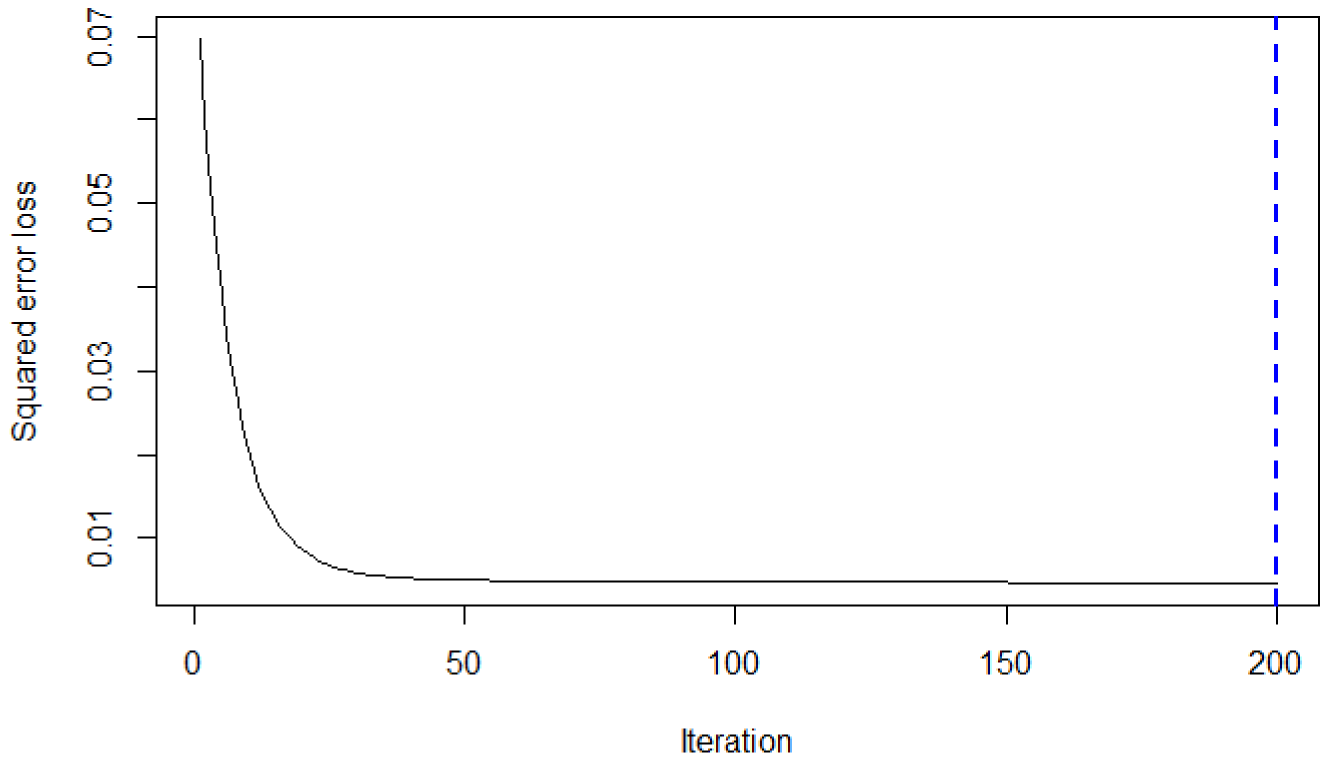


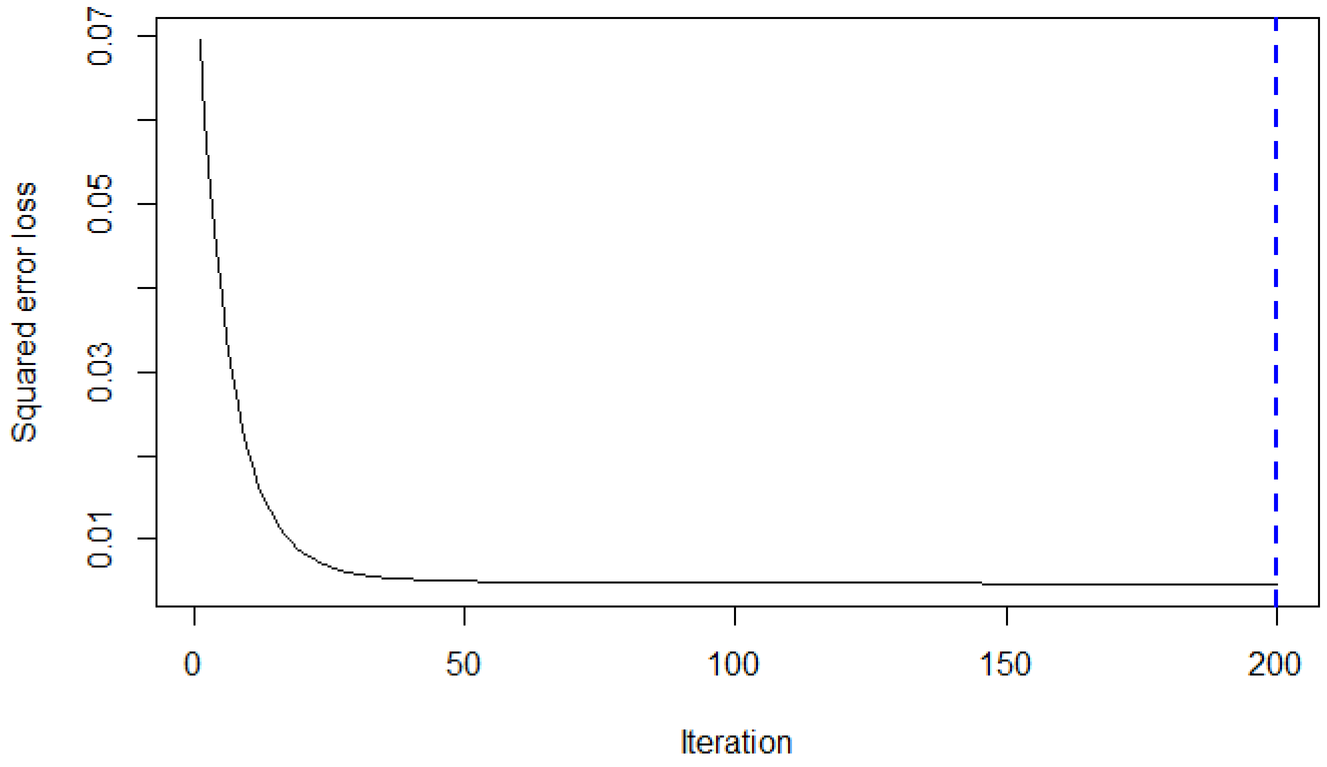

Squared error loss
0.02
0.04
0.06
0.08
0
50
100
150
200
Iteration

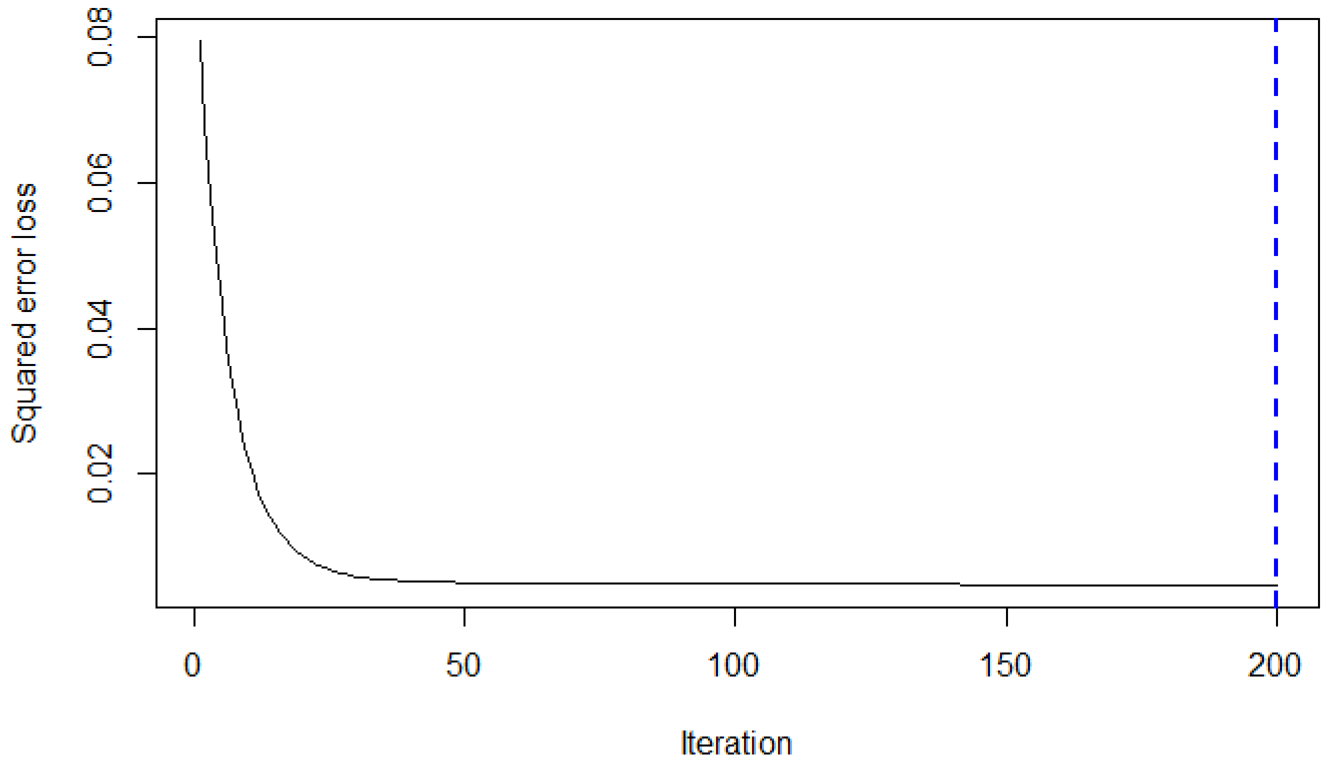

Squared error loss
0.02
0.04
0.06
0.08
0
50
100
150
200
Iteration

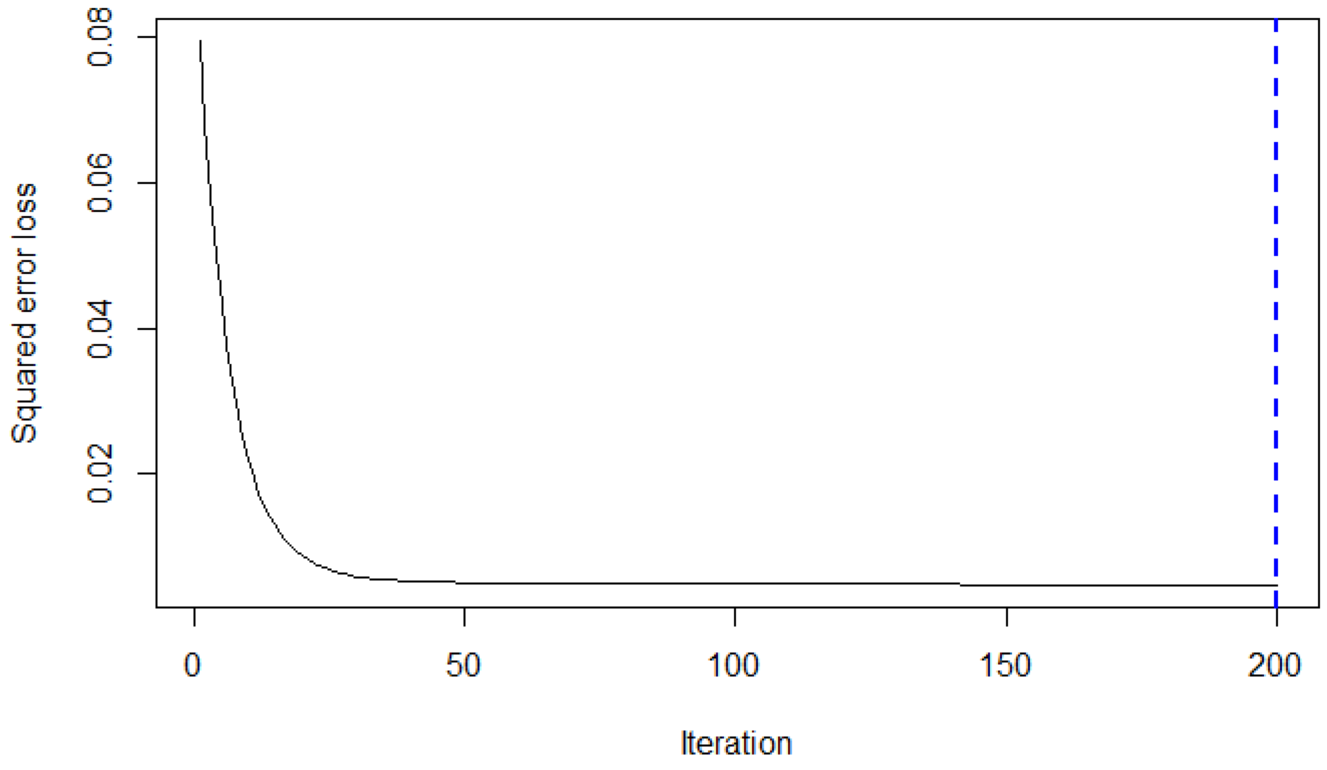

Squared error loss
0.07
0.05
0.03
0.01
0
50
100
150
200
Iteration

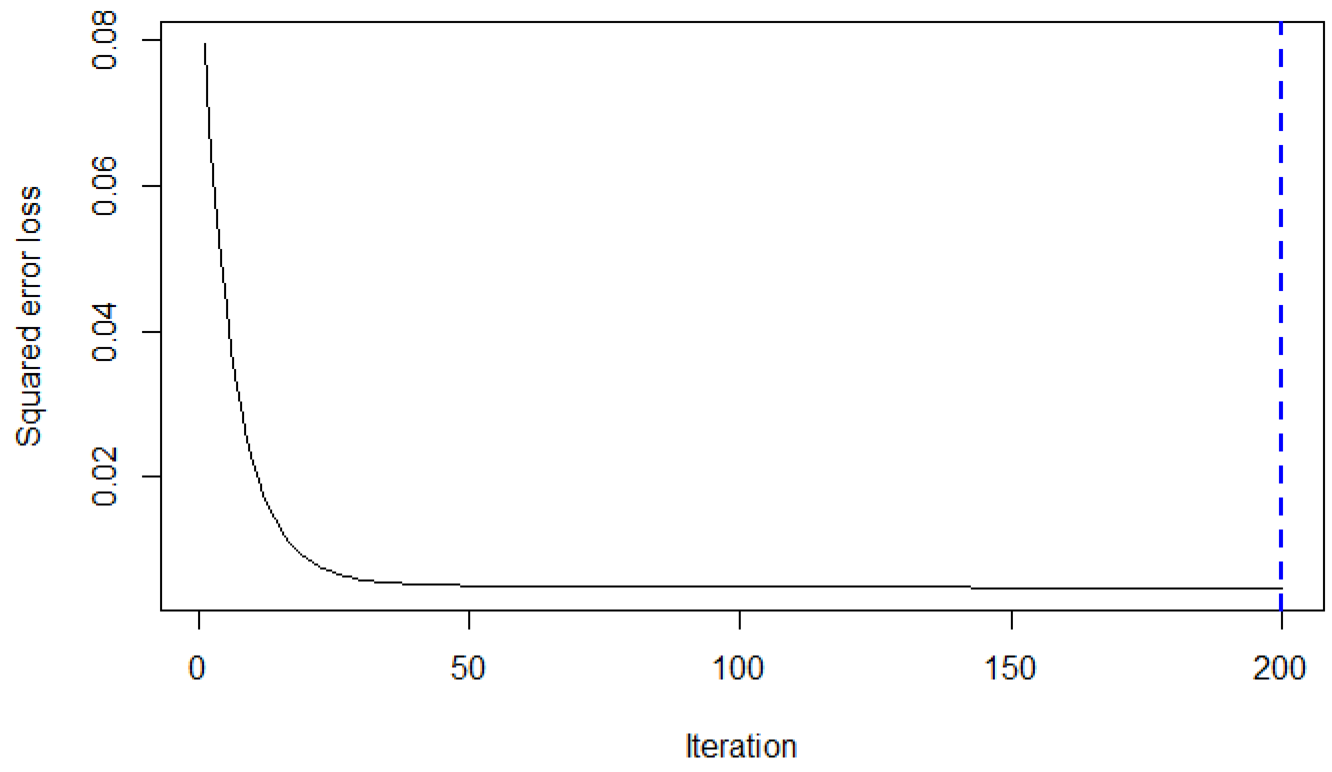

Squared error loss
0.07
0.05
0.03
0.01
0
50
100
150
200
Iteration

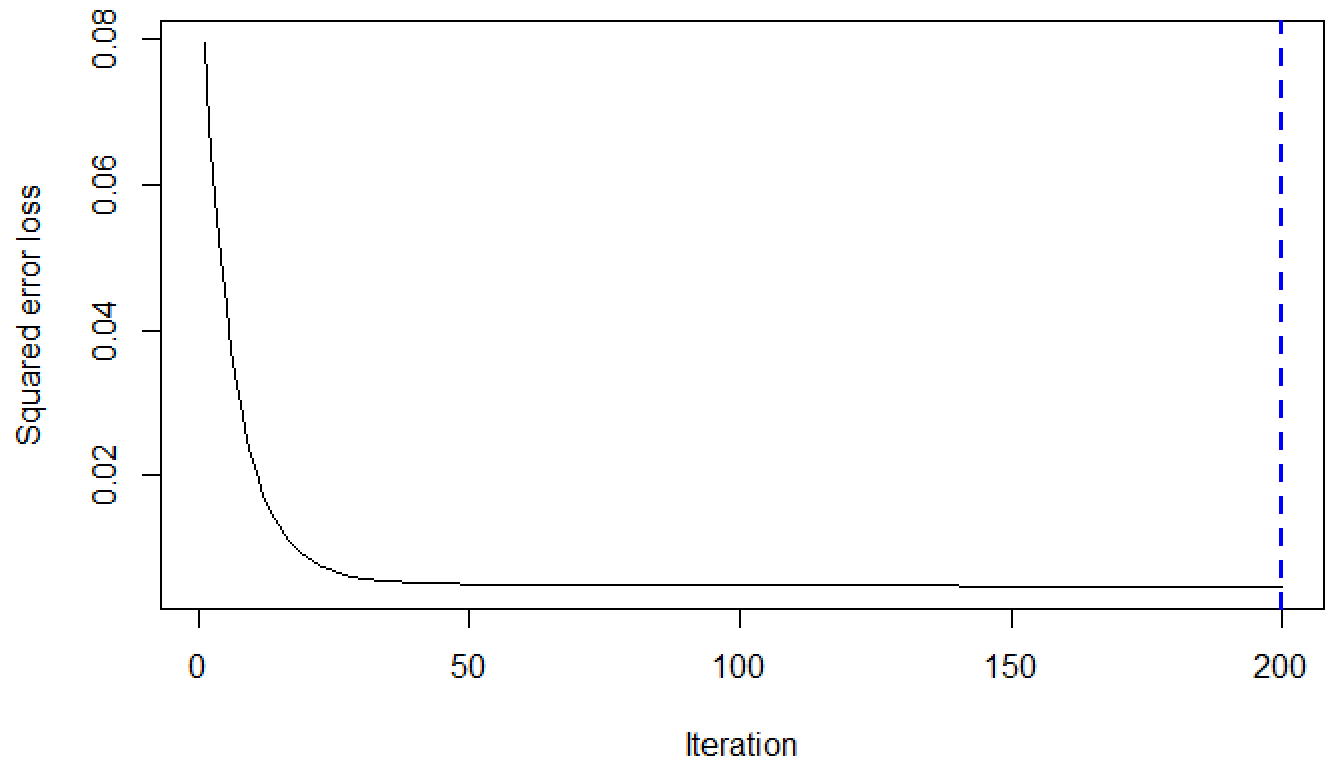

Squared error loss
0.07
0.05
0.03
0.01
0
50
100
150
200
Iteration


Squared error loss
0.07
0.05
0.03
0.01
0
50
100
150
200
Iteration


Squared error loss
0.02
0.04
0.06
0.08
0
50
100
150
200
Iteration


Squared error loss
0.02
0.04
0.06
0.08
0
50
100
150
200
Iteration





Hide

```
save(fit_train, file="../output/fit_train.RData")
```

Step 4: Super-resolution for test images

Feed the final training model with the completely holdout testing data. + `superResolution.R` + Input: a path that points to the folder of low-resolution test images. + Input: a path that points to the folder (empty) of high-resolution test images. + Input: an R object that contains tuned predictors. + Output: construct high-resolution versions for each low-resolution test image.

Hide

```
source("../lib/superResolution.R")
test_dir <- "../data/test_set/" # This will be modified for different data sets.
test_LR_dir <- paste(test_dir, "LR/", sep="")
test_HR_dir <- paste(test_dir, "HR/", sep="")
tm_test=NA
if(run.test){
  # load(file="../output/fit_train.RData")
  tm_test <- system.time(superResolution(test_LR_dir, test_HR_dir, fit_train))
}
```

Step 5: PSNR for test HR images and origin HR images

We write a img function to get a list of psnr of all the images. Then calculate the mean to get PSNR = 24.67765.

Hide

```
source("../lib/psnr.R")
psnr <- mean(calculate_psnr(test_HR_dir,train_HR_dir))
psnr
```

```
[1] 24.67765
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

Hide

```
cat("Time for constructing training features=", tm_feature_train[1], "s \n")
```

```
Time for constructing training features= 32.91 s
```

Hide

```
# cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training model=", tm_train[1], "s \n")
```

```
Time for training model= 3178.64 s
```

Hide

```
cat("Time for super-resolution=", tm_test[1], "s \n")
```

```
Time for super-resolution= 3331.08 s
```

Time for super-resolution is based on a 1500-image test set.