Project 3 - Example Main Script

Chengliang Tang, Tian Zheng

In your final repo, there should be an R markdown file that organizes **all computational steps** for evaluating your proposed image classification framework.

This file is currently a template for running evaluation experiments of image analysis (or any predictive modeling). You should update it according to your codes but following precisely the same structure.

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

```
## Loading required package: EBImage
```

```
if(!require("gbm")){
  install.packages("gbm")
}
```

```
## Loading required package: gbm
```

```
## Loaded gbm 2.1.4
```

```
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.

```
set.seed(2018)
# setwd("./ads_fall2018_proj3")
# use relative path for reproducibility
```

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

```
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) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

```
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
```

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

```
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.

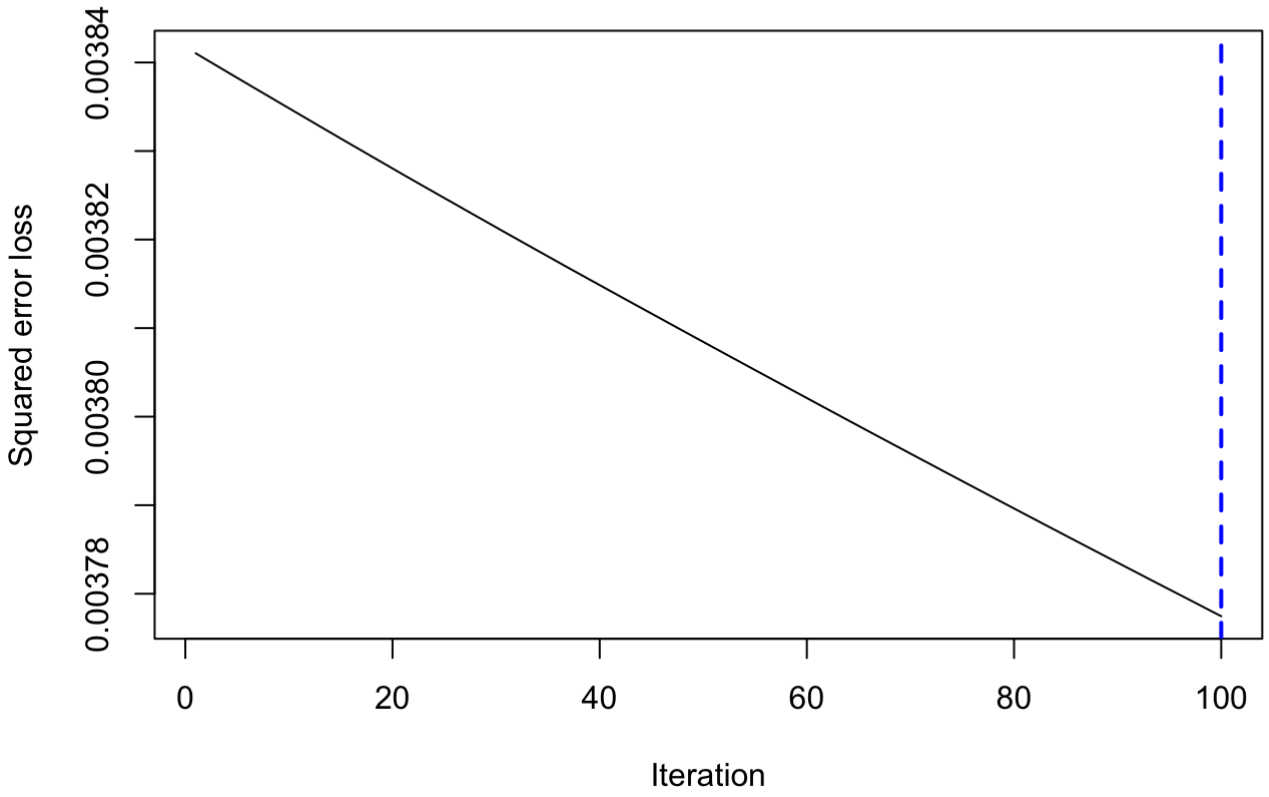
`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps. `+ train.R` + Input: a path that points to the training set features and responses. + Output: an RData file that contains trained classifiers in the forms of R objects: models/settings/links 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.

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

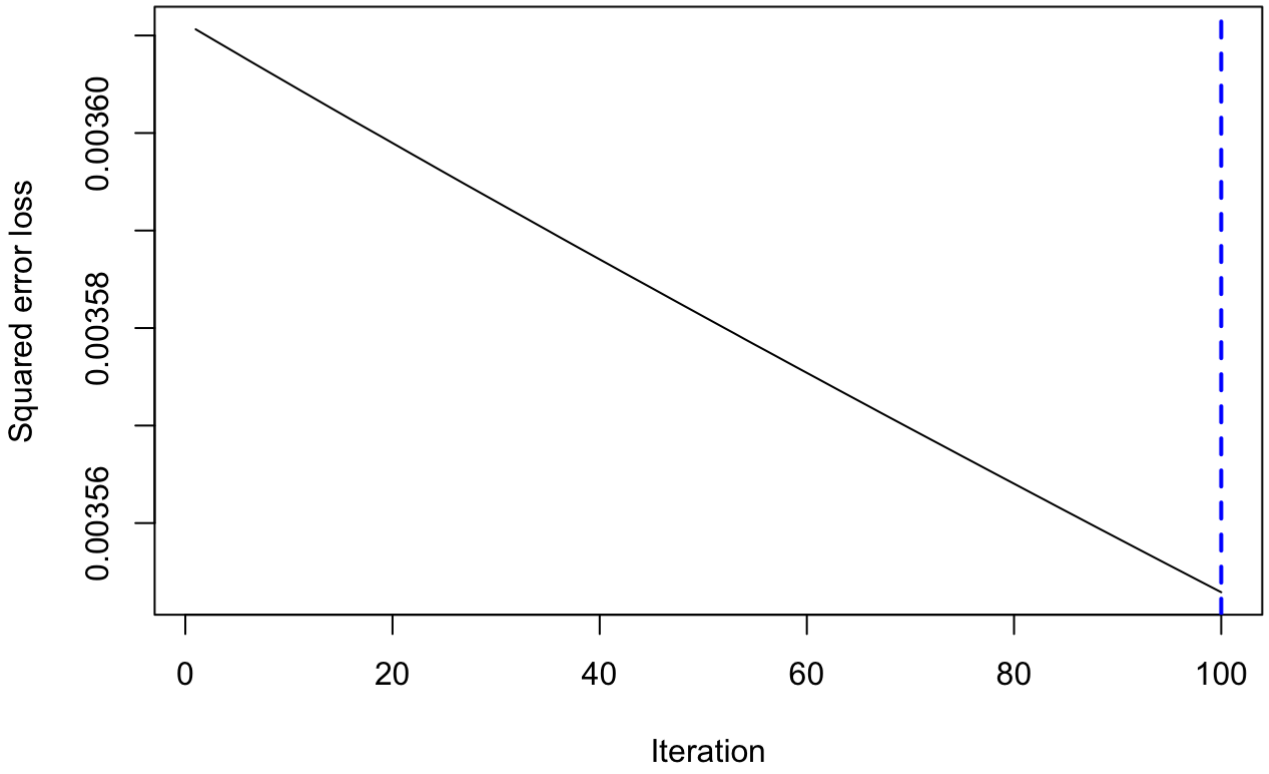
- Train the model with the entire training set using the selected model (model parameter)

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

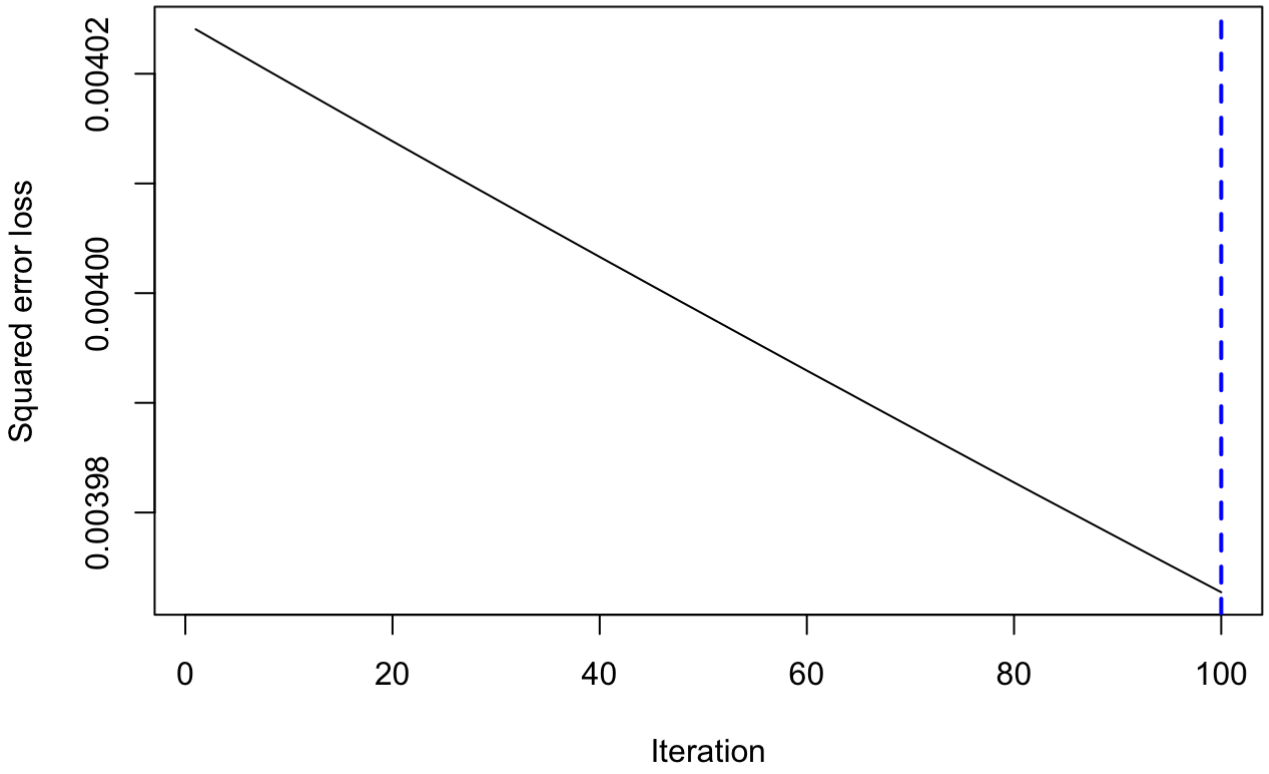
```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



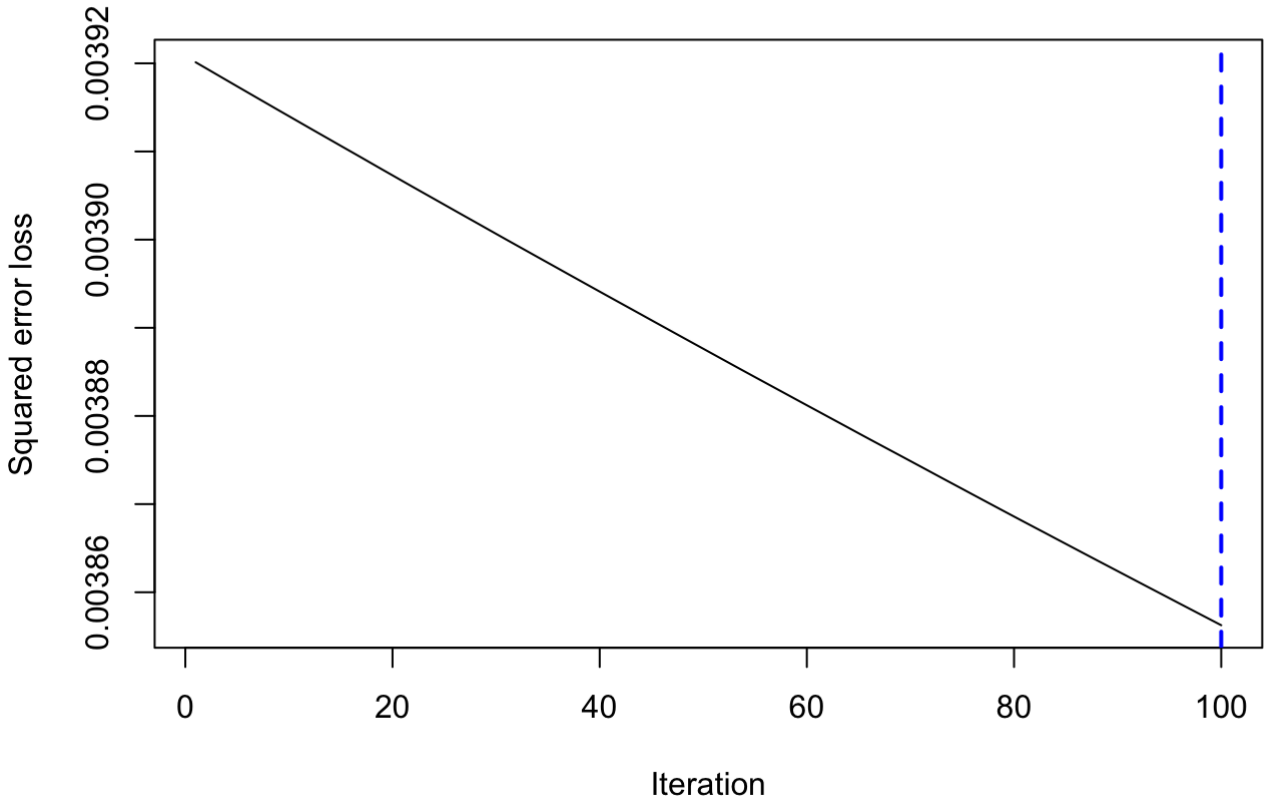

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



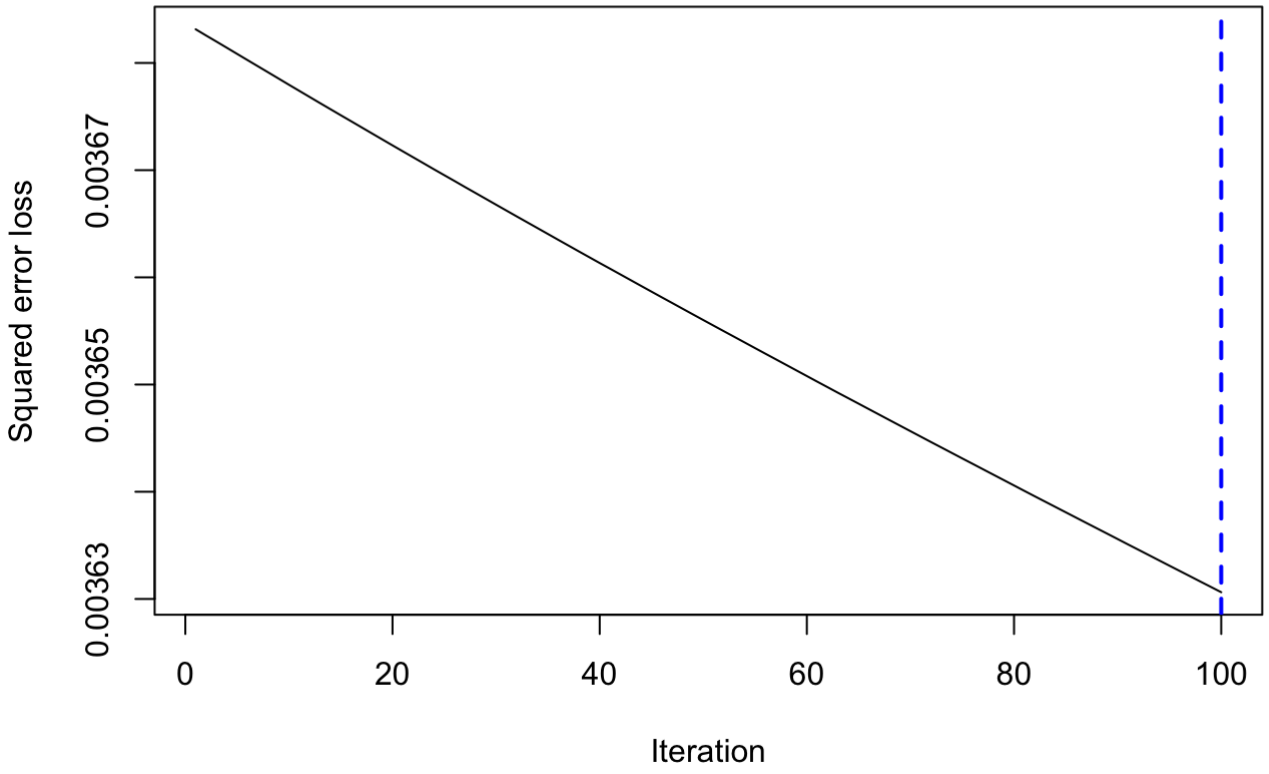

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



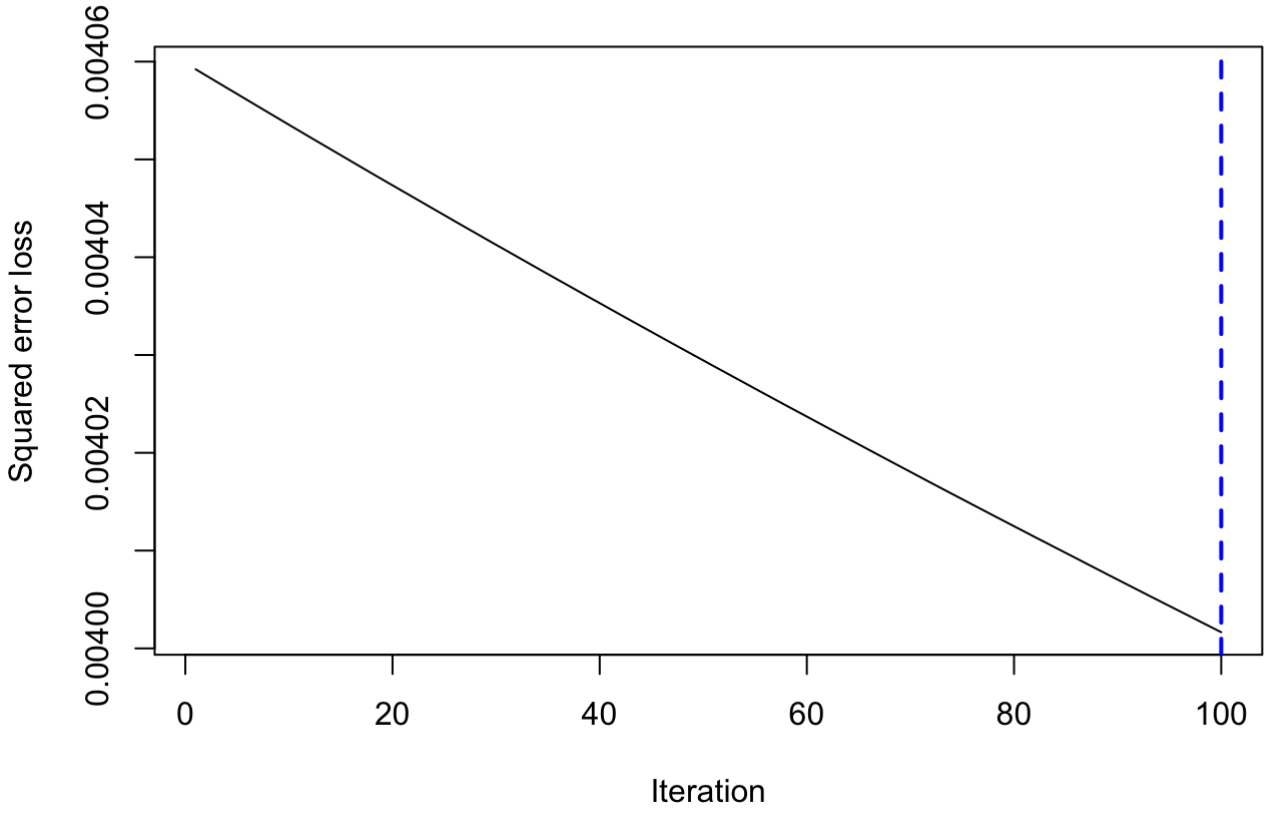

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



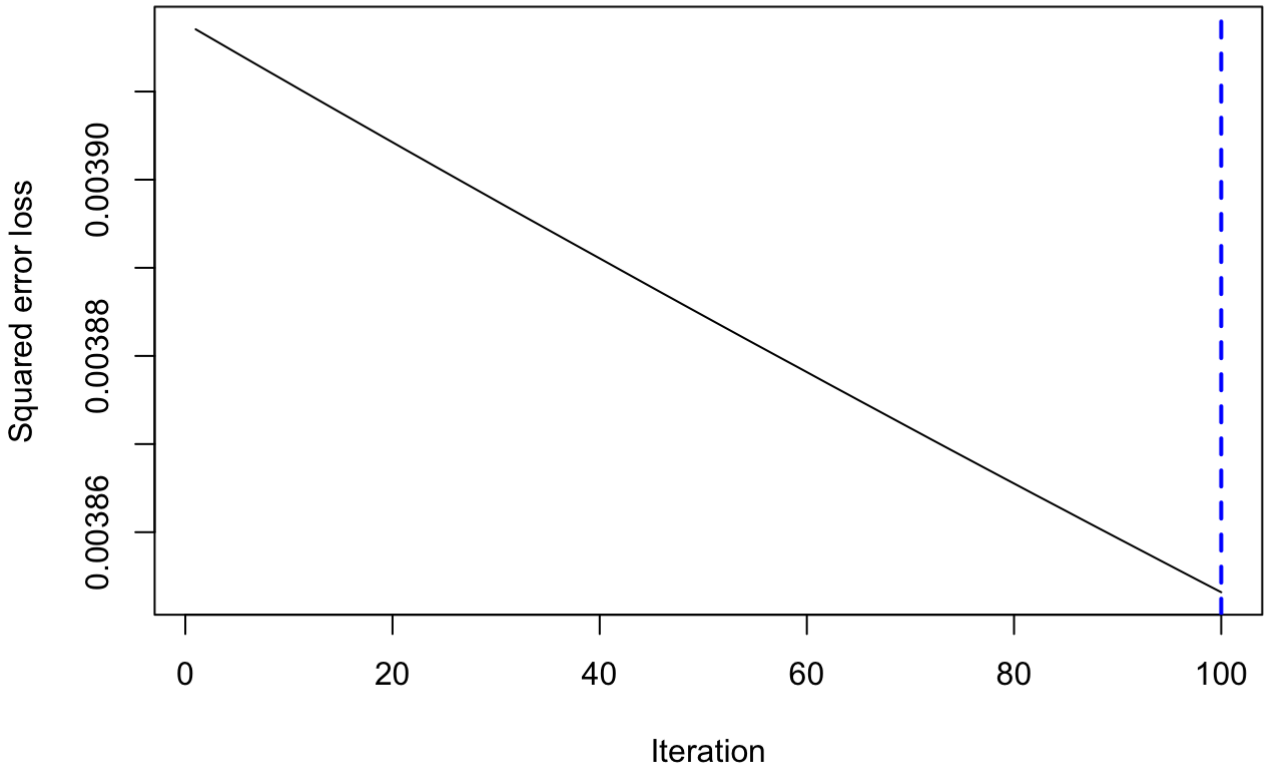

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



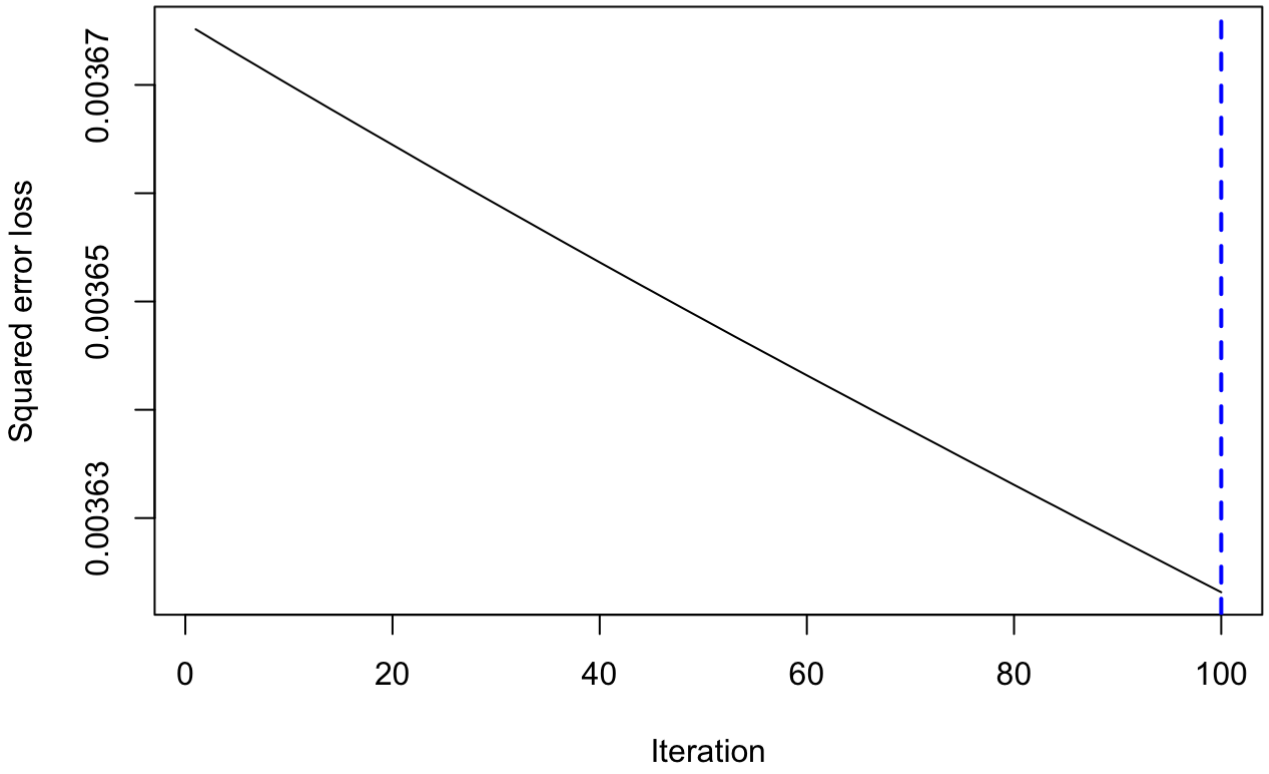

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



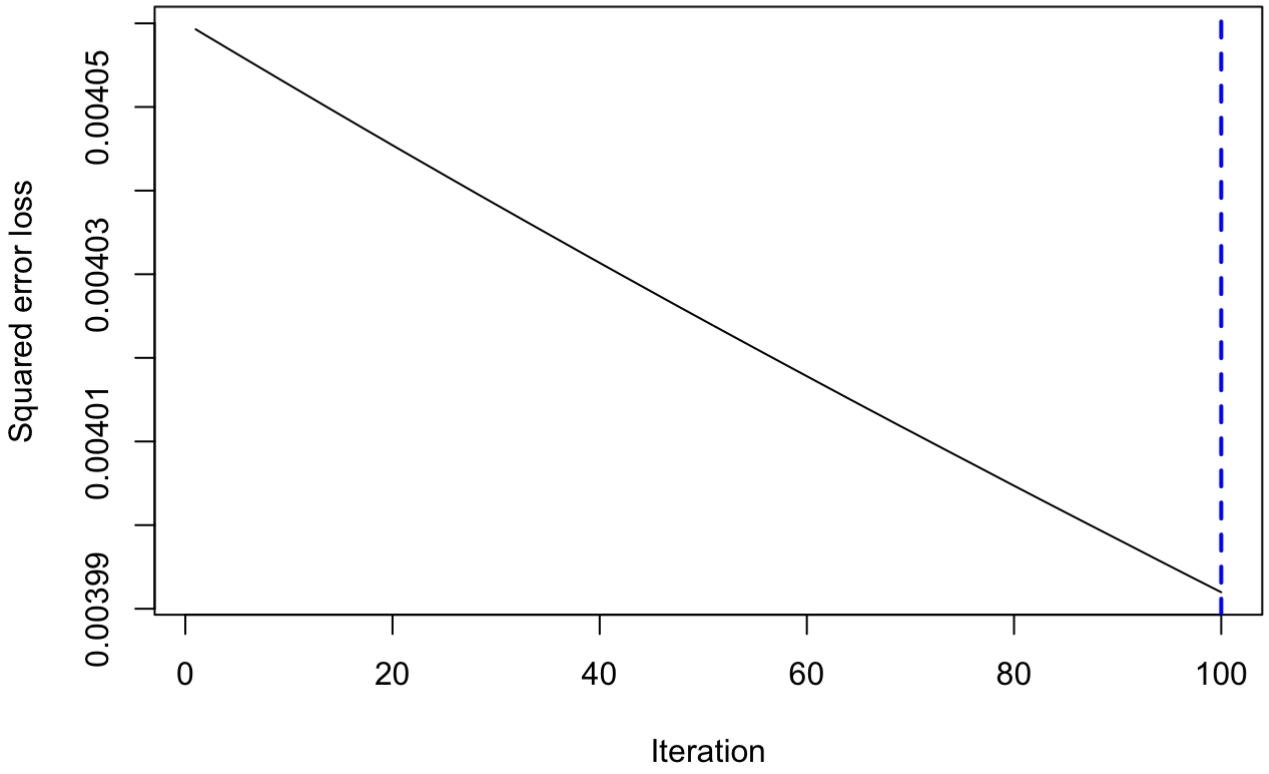

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



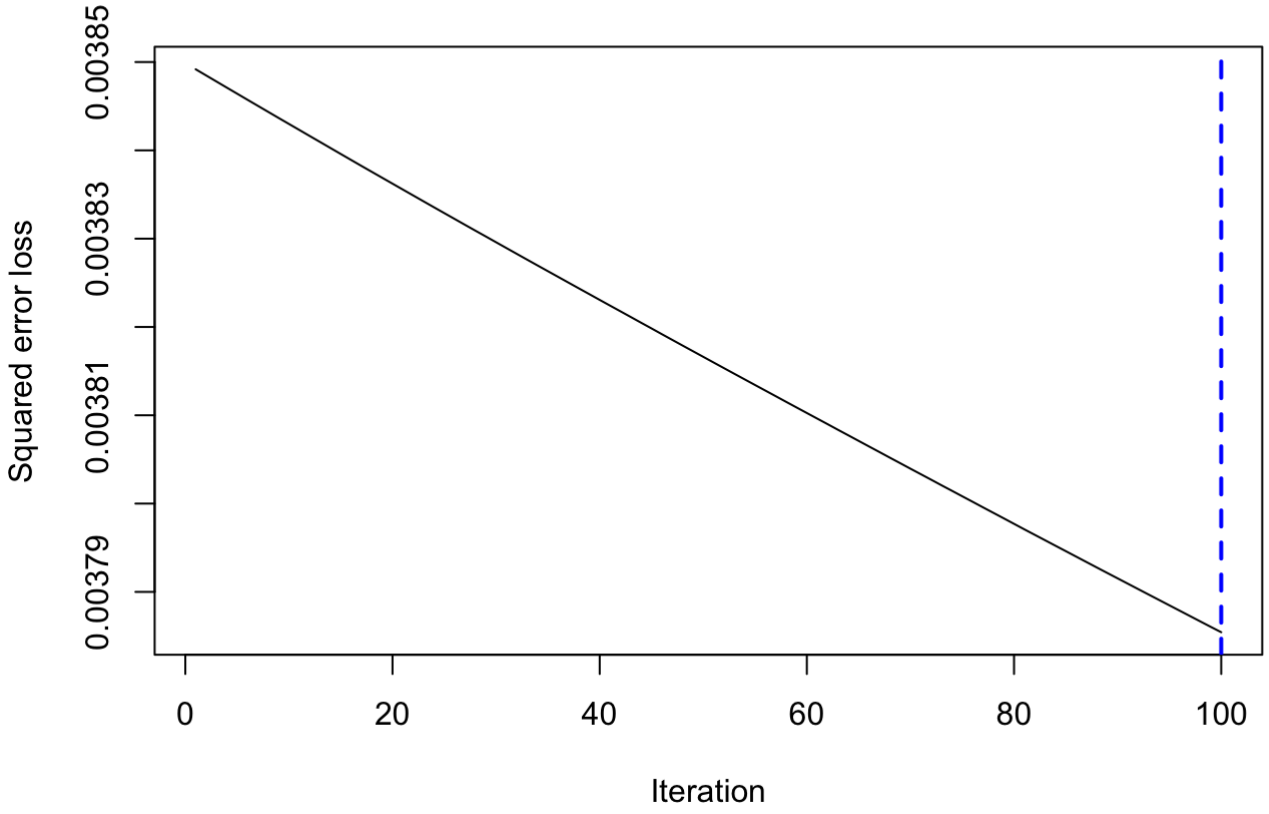

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```



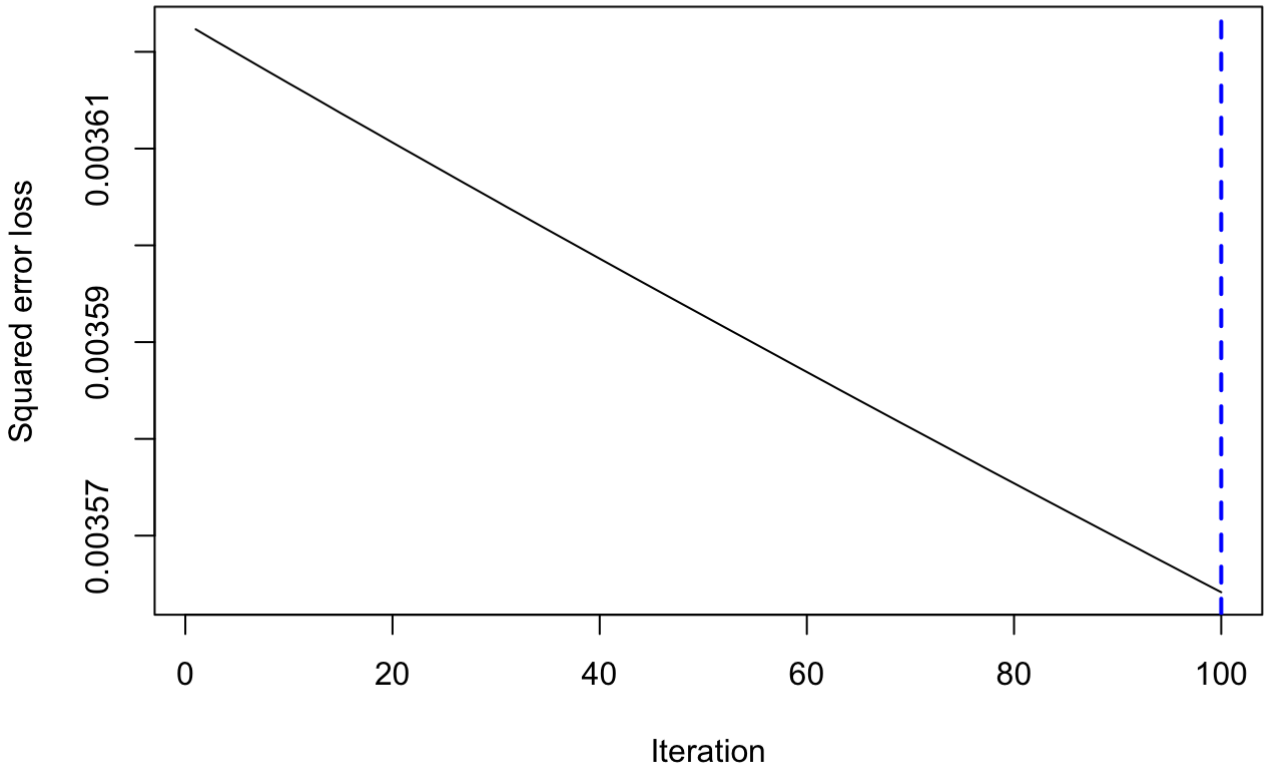

```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```

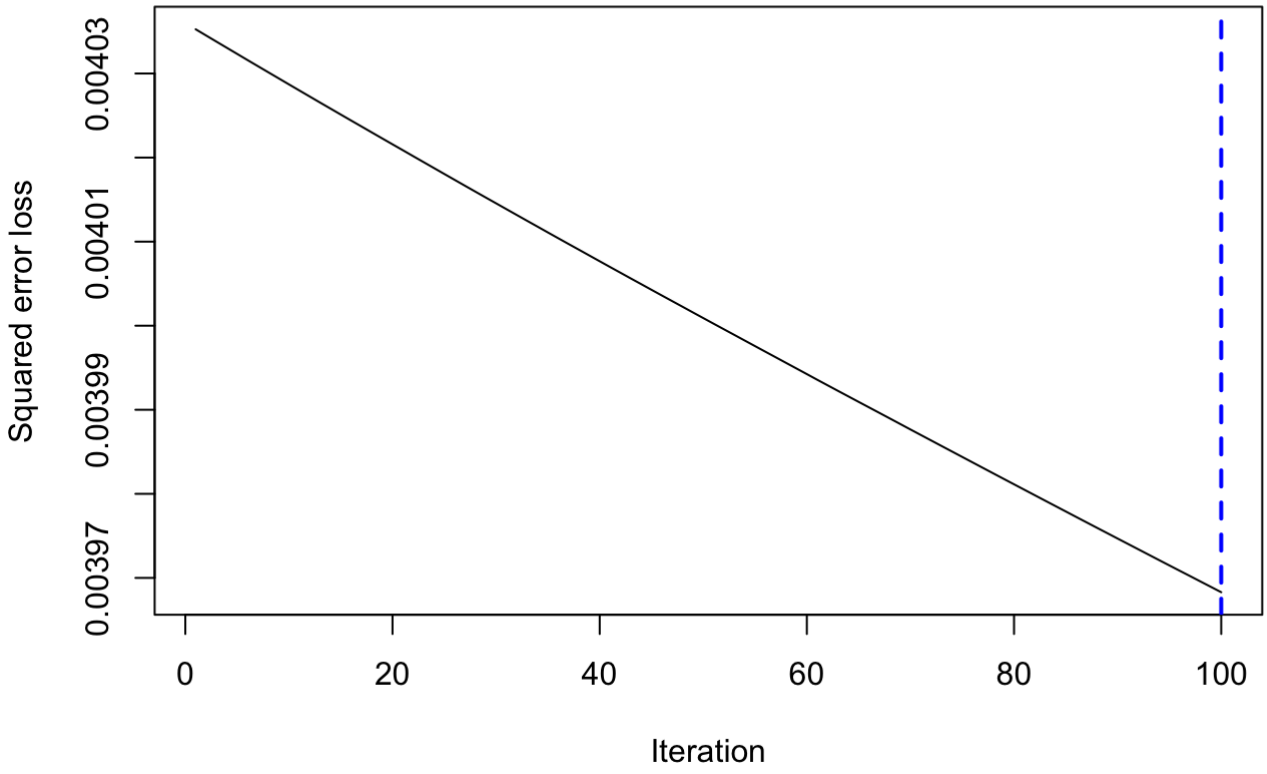



```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```




```
## OOB generally underestimates the optimal number of iterations although predictive per
formance is reasonably competitive. Using cv_folds>1 when calling gbm usually results in
improved predictive performance.
```


Squared error loss
0.00357
0.00359
0.00361
0
20
40
60
80
100
Iteration


Squared error loss
0.00397
0.00399
0.00401
0.00403
0
20
40
60
80
100
Iteration

```
#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.

```
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(MSE<-superResolution(test_LR_dir, test_HR_dir, fit_train))
}
PSNR <- 20*log(1,10)-10*log(MSE,10)
```

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.

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

```
## Time for constructing training features= 75.285 s
```

```
#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= 719.106 s
```

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

```
## Time for super-resolution= 2209.282 s
```

```
cat('MSE is',MSE,'\n')
```

```
## MSE is 0.003858508
```

```
cat('PSNR is',PSNR)
```

```
## PSNR is 24.13581
```