Boosted Trees

Prof Wells

STA 295: Stat Learning

April 30th, 2024

Outline

- Discuss boosted trees as example of ensemble models
- Implement boosted trees in R

Section 1

Boosting

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

- Can one use it to build a strong classifier that has error close to 0?

Motivation

Suppose you have a model which, given a binary classification dataset, always returned a classifier with training error strictly lower than 50%.

- Can one use it to build a strong classifier that has error close to 0?

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

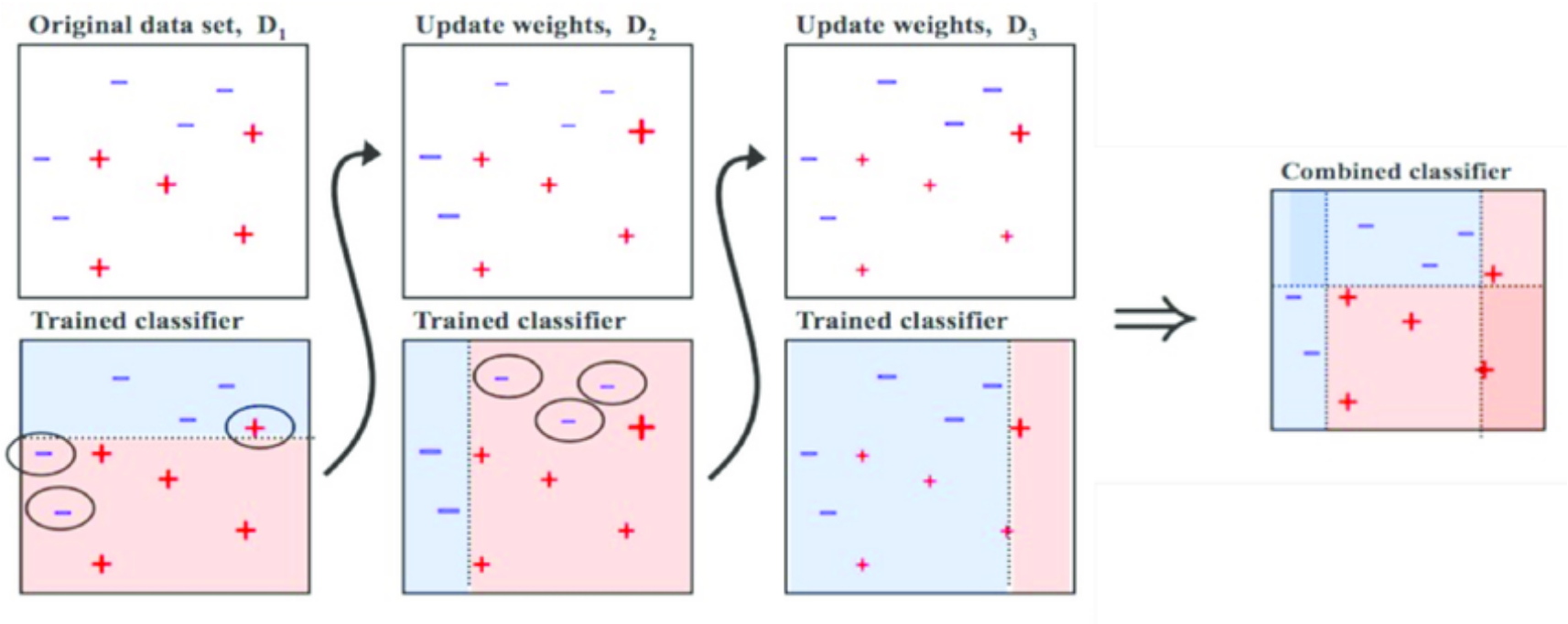
- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.
- The algorithm relies on using a sequence of **weak** learners (low variance, high bias)

AdaBoost

In the 1990s, Shapire and Freund developed algorithms to do just that.

- Their algorithm (AdaBoost) generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights.
  - Observations that are incorrectly classifed in the kth iteration recieve more weight in the $(k+1)$th iteration.
- The overall sequence of classifiers are combined into an ensemble which as high chance of classifying more accurately than any individaul model in the list.
- The algorithm relies on using a sequence of **weak** learners (low variance, high bias)
  - In the tree setting, we can create weak learners by restricting the depth of the tree.

AdaBoost Graphic



Boosting for regression

Boosting also works in the regression setting. The **gradient boosting machine** is a boosting algorithm that works as follows:

1. Select tree depth D and number of iterations K.
2. Compute the average response $\hat{y}$ and use this as the initial predicted value for each observation
3. Compute the residual for each observation.
4. Fit a regression tree of depth D, using the **residuals** as the response.
5. Predict each observation using the regression tree from the previous step.
6. Update the predicted value of each observation by adding the previous iteration's predicted value to the predicted value generated in the previous step.
7. Repeat at total of K times.

Brief Example

We return to the pdxTrees data a final time.

Compute the mean:

```
mu <- mean(my_pdxTrees_train$Carbon_Sequestration_lb)
mu
```

```
## [1] 34.49668
```

Brief Example

We return to the pdxTrees data a final time.

Compute the mean:

```
mu <- mean(my_pdxTrees_train$Carbon_Sequestration_lb)
mu
```

```
## [1] 34.49668
```

Compute residuals:

```
my_pdxTrees_train_boost <- my_pdxTrees_train %>%
  mutate(residuals1 = Carbon_Sequestration_lb - mu)
```

Brief Example

We return to the pdxTrees data a final time.

Compute the mean:

```
mu <- mean(my_pdxTrees_train$Carbon_Sequestration_lb)
mu
```

```
## [1] 34.49668
```

Compute residuals:

```
my_pdxTrees_train_boost <- my_pdxTrees_train %>%
  mutate(residuals1 = Carbon_Sequestration_lb - mu)
```

Fit a new tree

```
boost_tree_model<- rpart(residuals1 ~ Crown_Base_Height,
                   data = my_pdxTrees_train_boost,
                   control = rpart.control(maxdepth = 2))
```

Brief Example

We return to the pdxTrees data a final time.

Compute the mean:

```
mu <- mean(my_pdxTrees_train$Carbon_Sequestration_lb)
mu
```

```
## [1] 34.49668
```

Compute residuals:

```
my_pdxTrees_train_boost <- my_pdxTrees_train %>%
  mutate(residuals1 = Carbon_Sequestration_lb - mu)
```

Fit a new tree

```
boost_tree_model<- rpart(residuals1 ~ Crown_Base_Height,
                   data = my_pdxTrees_train_boost,
                   control = rpart.control(maxdepth = 2))
```

Predict

```
predictions<- predict(boost_tree_model, data = my_pdxTrees_test)+mu
```

Brief Example

We return to the pdxTrees data a final time.

Compute the mean:

```
mu <- mean(my_pdxTrees_train$Carbon_Sequestration_lb)
mu
```

```
## [1] 34.49668
```

Compute residuals:

```
my_pdxTrees_train_boost <- my_pdxTrees_train %>%
  mutate(residuals1 = Carbon_Sequestration_lb - mu)
```

Fit a new tree

```
boost_tree_model<- rpart(residuals1 ~ Crown_Base_Height,
                   data = my_pdxTrees_train_boost,
                   control = rpart.control(maxdepth = 2))
```

Predict

```
predictions<- predict(boost_tree_model, data = my_pdxTrees_test)+mu
```

And so on...

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)
  - Instead of adding the full value for a sample to the previous iteration's predicted value, only a fraction of the current predicted value is added.

Boosting Properties

Boosting is similar to random forests: the final prediction is sum of predictions from an ensemble of models.

- But in Random Forests, all trees are created independently, are of maximum depth, and contribute equally to the final model.
- In boosting, subsequent trees are are highly dependent on past trees, have minimal depth, and contribute unequally.

Unlike random forests, boosting is susceptible to over-fitting (since it uses a greedy algorithm to maximize gradient at each step).

- To remedy, we introduce a shrinkage penalty (like in Ridge Regression/LASSO)
  - Instead of adding the full value for a sample to the previous iteration's predicted value, only a fraction of the current predicted value is added.
  - This fraction is called the *learning rate* λ, with $0 < \lambda < 1$. (Typical values range from 0.001 to 0.01)

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`
- The argument `n.trees` controls the number of iterations
- The argument `interaction.depth` controls the depth of each tree
- The argument `shrinkage` controlls the learning rate λ

Boosting in R

We use the `gbm` function in the `gmb` package to create Boosted Trees

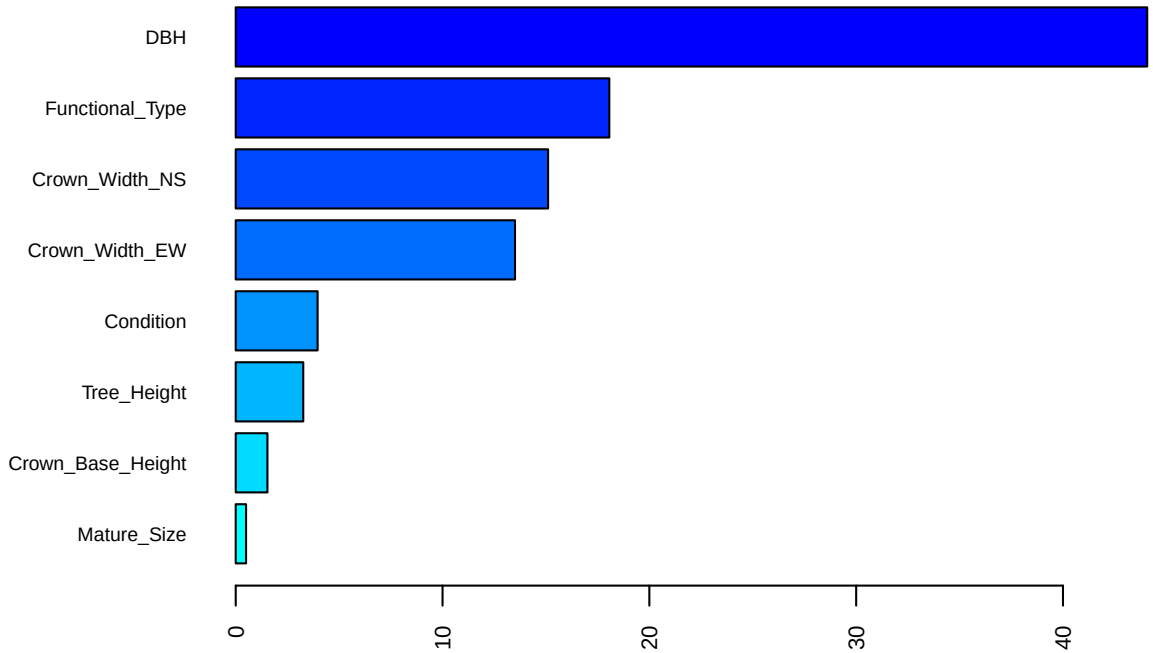
- For regression problems, we use the argument `distribution = "gaussian"` and for classification problems, we use `distribution = "bernoulli"`
- The argument `n.trees` controls the number of iterations
- The argument `interaction.depth` controls the depth of each tree
- The argument `shrinkage` controlls the learning rate λ

```
library(gbm)
set.seed(10101)
boosted_tree<-gbm(Carbon_Sequestration_lb ~., my_pdxTrees_train,
                  distribution = "gaussian",
                  n.trees=400,
                  interaction.depth = 3,
                  shrinkage = .1)
```

Summary Information

```
summary(boosted_tree )
```

```
##                                 var    rel.inf
## DBH                             DBH 44.0715885
## Functional_Type     Functional_Type 18.0639257
## Crown_Width_NS       Crown_Width_NS 15.1030328
## Crown_Width_EW       Crown_Width_EW 13.5036280
## Condition                 Condition  3.9588168
## Tree_Height             Tree_Height  3.2655545
## Crown_Base_Height Crown_Base_Height  1.5339425
## Mature_Size             Mature_Size  0.4995112
```



Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

- This behavior is typical. Boosted trees and Random Forests often have comparable performance, and both tend to be more accurate than other model types

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

- This behavior is typical. Boosted trees and Random Forests often have comparable performance, and both tend to be more accurate than other model types

- However, this performance comes at significant cost of interpretability.

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

- This behavior is typical. Boosted trees and Random Forests often have comparable performance, and both tend to be more accurate than other model types
- However, this performance comes at significant cost of interpretability.
- Note that boosted trees have a number of important parameters: `n.trees`, `interaction.depth`, `shrinkage`.

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

- This behavior is typical. Boosted trees and Random Forests often have comparable performance, and both tend to be more accurate than other model types
- However, this performance comes at significant cost of interpretability.
- Note that boosted trees have a number of important parameters: `n.trees`, `interaction.depth`, `shrinkage`.
  - How do we find the best values of these hyperparameters?

Boosted Tree Performance

- How does the boosted tree do vs Random Forest? A pruned tree? A linear model?

```
results %>% group_by(model) %>% rmse(truth = obs, estimate = preds) %>% arrange(.estimate)
```

```
## # A tibble: 4 x 4
##   model         .metric .estimator .estimate
##   <chr>         <chr>   <chr>          <dbl>
## 1 random_forest rmse    standard        10.8
## 2 boosted_tree  rmse    standard        11.5
## 3 pruned_tree   rmse    standard        13.7
## 4 linear_model  rmse    standard        17.7
```

- This behavior is typical. Boosted trees and Random Forests often have comparable performance, and both tend to be more accurate than other model types
- However, this performance comes at significant cost of interpretability.
- Note that boosted trees have a number of important parameters: `n.trees`, `interaction.depth`, `shrinkage`.
  - How do we find the best values of these hyperparameters?
  - Cross-validation!

Cross-Validating gbm

Warning! fitting a single gbm models can be time and computing intensive.

- Using cross-validation to compare multiple models can be VERY time and computing intensive
- Cross-validation for gbm models is NOT RECOMMENDED if using the RStudio Server

Cross-Validating gbm

Warning! fitting a single gbm models can be time and computing intensive.

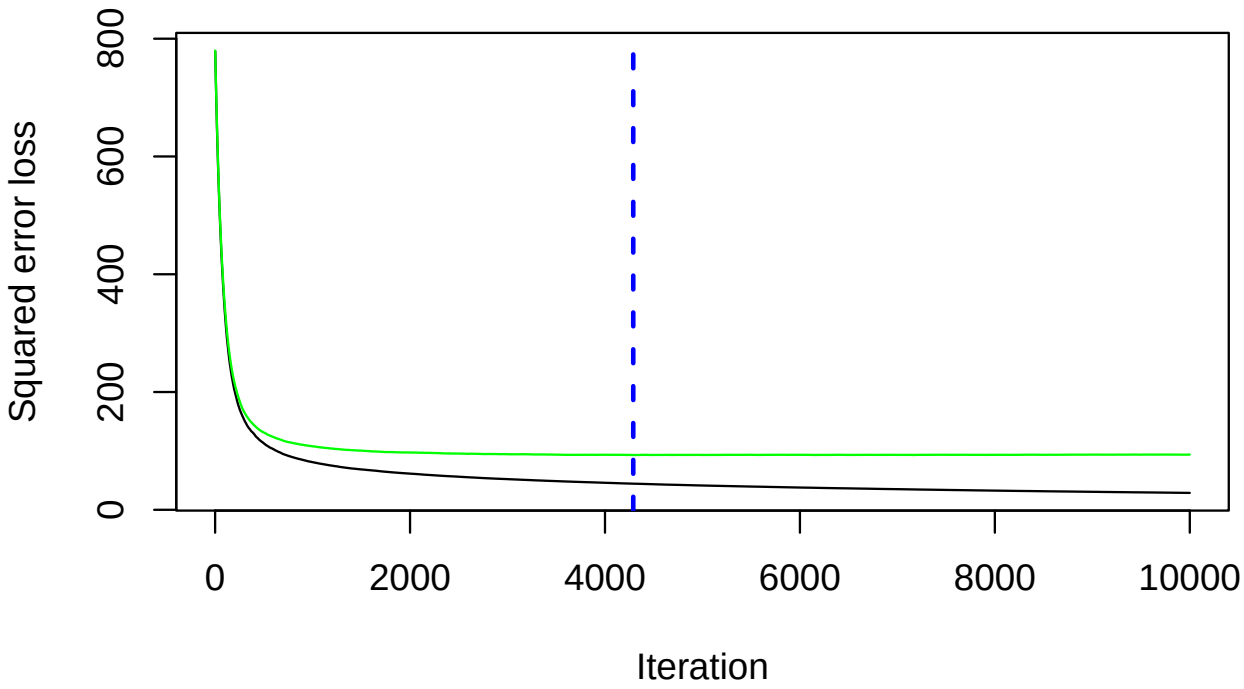
- Using cross-validation to compare multiple models can be VERY time and computing intensive
- Cross-validation for gbm models is NOT RECOMMENDED if using the RStudio Server
- We can include an additional cross-validation term in our boosted tree model.
  - It may be helpful to include a number of CPU cores as well. First verify your number of available cores using `parallel::decectCores()`

```
library(gbm)
set.seed(10101)
cv_boosted_tree<-gbm(Carbon_Sequestration_lb ~., my_pdxTrees_train,
                     distribution = "gaussian",
                     n.trees=10000,
                     interaction.depth = 3,
                     shrinkage = .01,
                     cv.folds = 10,
                     n.cores = 8)
```

CV Results

- We can plot cross-validated performance using `gbm.perf()`

```
gbm.perf(cv_boosted_tree, method = "cv")
```

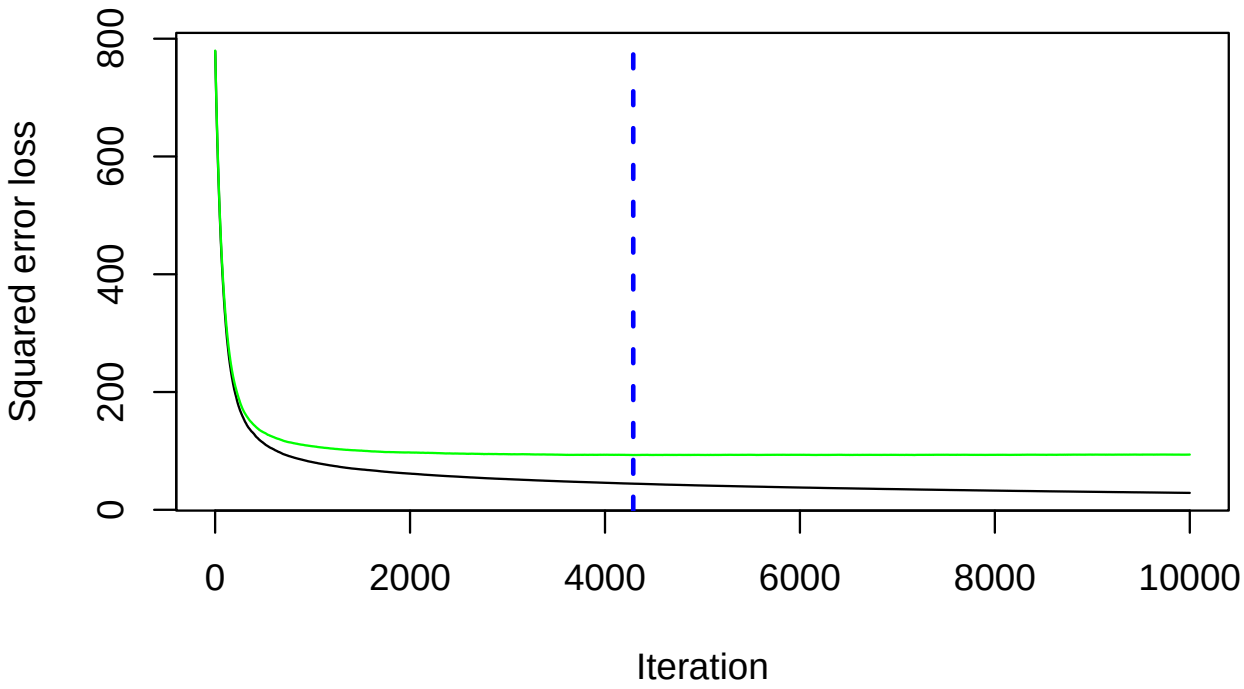



```
## [1] 4290
```

CV Results

- We can plot cross-validated performance using `gbm.perf()`

```
gbm.perf(cv_boosted_tree, method = "cv")
```



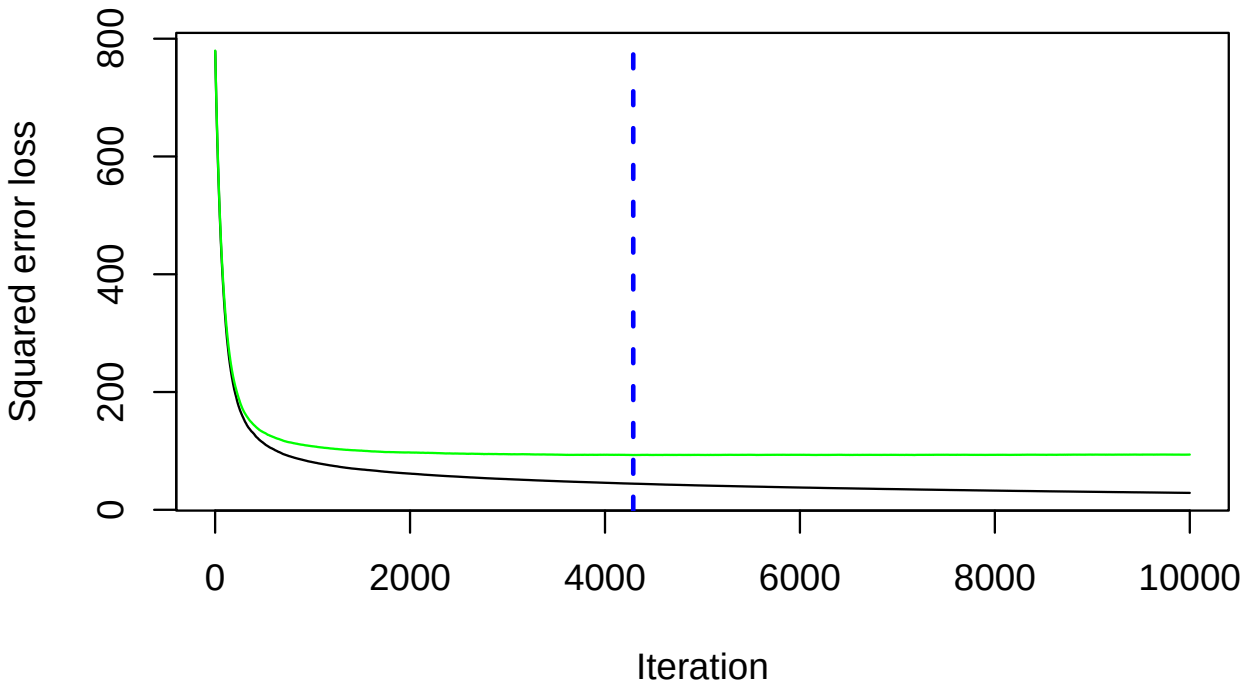

```
## [1] 4290
```

- The green curve is the cross-validated error, while the black curve is the training error.

CV Results

- We can plot cross-validated performance using `gbm.perf()`

```
gbm.perf(cv_boosted_tree, method = "cv")
```




```
## [1] 4290
```

- The green curve is the cross-validated error, while the black curve is the training error.
- The blue vertical line is the optimal value of the cross-validated error

Recording CV Error

- The `gbm` object also stores the values of the cross-validated errors for each number of trees used, accessible via `$cv.errors`

Recording CV Error

- The `gbm` object also stores the values of the cross-validated errors for each number of trees used, accessible via `$cv.errors`

```
my_errors <- cv_boosted_tree$cv.error
best_n <- which.min(cv_boosted_tree$cv.error)
data.frame(best_n, cv_error = my_errors[best_n])
```

```
##   best_n cv_error
## 1   4290 93.01164
```

Recording CV Error

- The `gbm` object also stores the values of the cross-validated errors for each number of trees used, accessible via `$cv.errors`

```
my_errors <- cv_boosted_tree$cv.error
best_n <- which.min(cv_boosted_tree$cv.error)
data.frame(best_n, cv_error = my_errors[best_n])
```

```
##   best_n cv_error
## 1   4290 93.01164
```

- This is particularly useful if we want to record the error for a model with certain parameters

General Strategy for finding best Parameters

1. Choose a relatively high initial learning rate. A rate of 0.1 is a reasonable starting point.
2. Determine the optimal number of trees for this learning rate using cross-validation.
3. Fix other tree-specific parameters and tune the learning rate, assessed by computation speed and model accuracy.
4. Tune tree-specific parameters for fixed learning rate.
5. Once tree-specific parameters have been found, lower learning rate and increase number of trees to assess improvements in accuracy.

Warning! This search can take considerable time (minutes to hours), depending on computing power, number of variables in model, and number of observations. DO NOT ATTEMPT ON RSTUDIO SERVER!!