Using Gradient Boosting for enhancer-promoter-class data predictions.

Shaurya Jauhari (Email: shauryajauhari@gzhmu.edu.cn)

2019-11-04

Contents

List of Figures

Introduction

In this module, we're going to explore the gradient boosting algorithm. *Gradient boosting could be construed as a unison of boosting and gradient descent techniques*, i.e. boosting carried out on the gradient descent of the cost function. The gradient descent approach, in loose terms, is the protocol to reach the minima of the function that maps the error terms. Our objective is to minimize the trade-off (error between the true values and the predicted values), thereby finding appropriate coefficients that fit the objective function's equation in a way that best approximations are engendered.

```r
myimages <- list.files("./Gradient_Descent", pattern = ".jpeg", full.names = TRUE)
knitr::include_graphics(myimages)
```

In the figures 2 and 3, we notice that the learning rate plays a key role in cornering the lowest difference point between the predicted and actual values. If we choose on a too high rate, despite being computationally fast, it is likely to overshoot and skip the target we desire. Contarily, a slow learning rate shall take it's *own sweet time* but somewhat guarantee to reach the minima.

Boosting is a way to derive random sub-samples of the same size as the original one.

This aspect of gradient boosting differentiates it from **Random Forests** that aim to aggregate outputs from multiple single-decision trees, implementing basically a consensus-based scheme. Gradient Boosting would contrarily penalize the leafs from the same decision tree until the gradient descent strategy's optimal parameters are met.

Gradient Boosting is also exclusive to Adaptive Boosting or **AdaBoost**

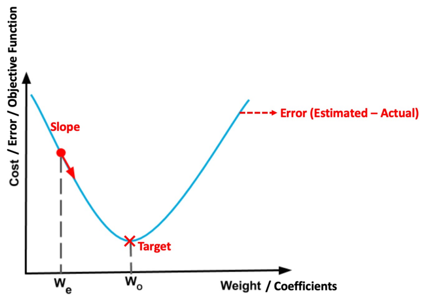


Figure 1: Gradient Descent overview

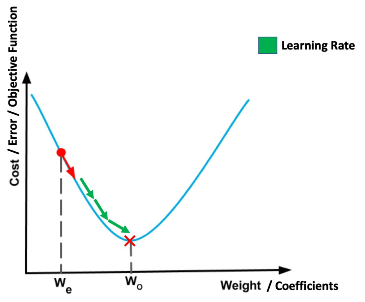


Figure 2: Gradient Descent overview

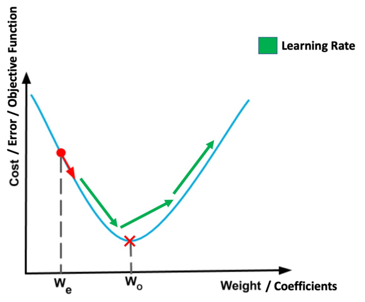


Figure 3: Gradient Descent overview

Package Installation

The package in question is “gbm”.

```r
install.packages("gbm", dependencies = TRUE, verbose = TRUE,
                 repos = "https://mirrors.tuna.tsinghua.edu.cn/CRAN/")
```

```
## Installing package into '/Users/mei/Library/R/3.6/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/hm/c3_fjypn62v5xh5b5ygv267m0000gn/T//RtmpdAuBfI/downloaded_packages
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.5
```

Dataset

We’ll be employing the enhancer prediction dataset to exemplify an application of gradient boosting algorithm. The details for the dataset could be reached here.

```r
epdata <- readRDS("../Machine_Learning_Deep_Learning/data/ep_data_sample.rds")
rownames(epdata) <- c()


set.seed(001)
data_partition <- sample(3, nrow(epdata), replace = TRUE, prob = c(0.64,0.16,0.2))
train <- epdata[data_partition==1,]
test <- epdata[data_partition==2,]
holdout <- epdata[data_partition==3,]

head(train)
```

```
##    peaks_h3k27ac peaks_h3k4me3 peaks_h3k4me2 peaks_h3k4me1 class
## 1       0.734144       0.00000      0.000000             0     1
## 2       0.000000       0.00000      0.000000             0     0
## 3       1.468290       1.13595      0.865892             0     1
## 5       0.000000       1.13595      0.000000             0     1
## 9       3.670720       0.00000      0.865892             0     1
## 10      0.000000       0.00000      0.000000             0     1
```

Error Metric: Logarithmic Loss

```r
set.seed(002)
LogLossBinary = function(actual, predicted, eps = 1e-15) {
    predicted = pmin(pmax(predicted, eps), 1-eps)
    - (sum(actual * log(predicted) + (1 - actual) * log(1 - predicted))) / length(actual)

}
```

The logarithmic loss function is used to estimate the accuracy of a classifier by penalizing the misclassifications by it.For details check out this link.

Fitting Model

```
gbm_model <- gbm(formula = class ~ peaks_h3k27ac + peaks_h3k4me3 + peaks_h3k4me2 + peaks_h3k4me1,
               distribution = "bernoulli",
               data = train,
               n.trees = 2500,
               shrinkage = .01,
               n.minobsinnode = 20)
```

Testing the model on train data

Let's see how well the model performs on the training data.

```
gbm_train_pred <- predict(object = gbm_model,
                              newdata = train,
                              n.trees = 1500,
                              type = "response")
```

Let us view the first few results.

```
head(data.frame("Actual" = train$class,
                "Prediction Probability" = gbm_train_pred))
```

```
##   Actual Prediction.Probability
## 1      1              0.6210100
## 2      0              0.6313441
## 3      1              0.6320952
## 4      1              0.6450761
## 5      1              0.6408924
## 6      1              0.6313441
```

```
cat("The accuracy of the classifier is", (LogLossBinary(train$class, gbm_train_pred)*100),"%")
```

```
## The accuracy of the classifier is 65.29076 %
```

Testing the model on test data

It is pretty obvious that a model shall score high if the predictions are made on the same data it was trained on. How does it stack up with the test and holdout data.

```
# prediction

gbm_test_pred <- predict(object = gbm_model,
                              newdata = test,
                              n.trees = 1500,
                              type = "response") # binary classification (not regression)


#results

head(data.frame("Actual" = test$class,
                "Prediction Probability" = gbm_test_pred))
```

```
##   Actual Prediction.Probability
## 1      0              0.6263248
## 2      1              0.6437835
## 3      1              0.6313441
## 4      1              0.6199855
## 5      1              0.6313441
## 6      1              0.6348683
```

```
cat("The accuracy of the classifier is", (LogLossBinary(test$class, gbm_test_pred)*100),"%")
```

```
## The accuracy of the classifier is 66.85663 %
```

HoldOut dataset

When you have multiple machine learning candidate-models, the ideal way to make a selection for the best one is to follow the **train-test-holdout** data strategy; where the model is built on the training set, the prediction errors are calculated using the test set, and the holdout set is used to assess the generalization error of the final model.This is especially crucial while working with an ensemble of methods. In the present context, the idea is to choose the best-fitting decision tree out of several possible choices (of branching). P.S. the test data is also interchangeably referred as validation data.

Let us analyze how the model on test data fares on the holdout data.

```
# model

gbm_model_test <- gbm(formula = class ~ peaks_h3k27ac + peaks_h3k4me3 + peaks_h3k4me2 + peaks_h3k4me1,
               distribution = "bernoulli",
               data = test,
               n.trees = 1500,
               shrinkage = .01,
               n.minobsinnode = 50)

# prediction

gbm_ho_pred <- predict(object = gbm_model_test,
                              newdata = holdout,
                              n.trees = 1500,
                              type = "response") # binary classification (not regression)


#results

head(data.frame("Actual" = holdout$class,
                "Prediction Probability" = gbm_ho_pred))
```

```
##   Actual Prediction.Probability
## 1      1              0.6050356
## 2      1              0.6616224
## 3      1              0.5764757
## 4      1              0.7541860
## 5      0              0.6418048
## 6      1              0.6284930
```

```
cat("The accuracy of the classifier is", (LogLossBinary(holdout$class, gbm_ho_pred)*100),"%")
```

```
## The accuracy of the classifier is 66.63774 %
```

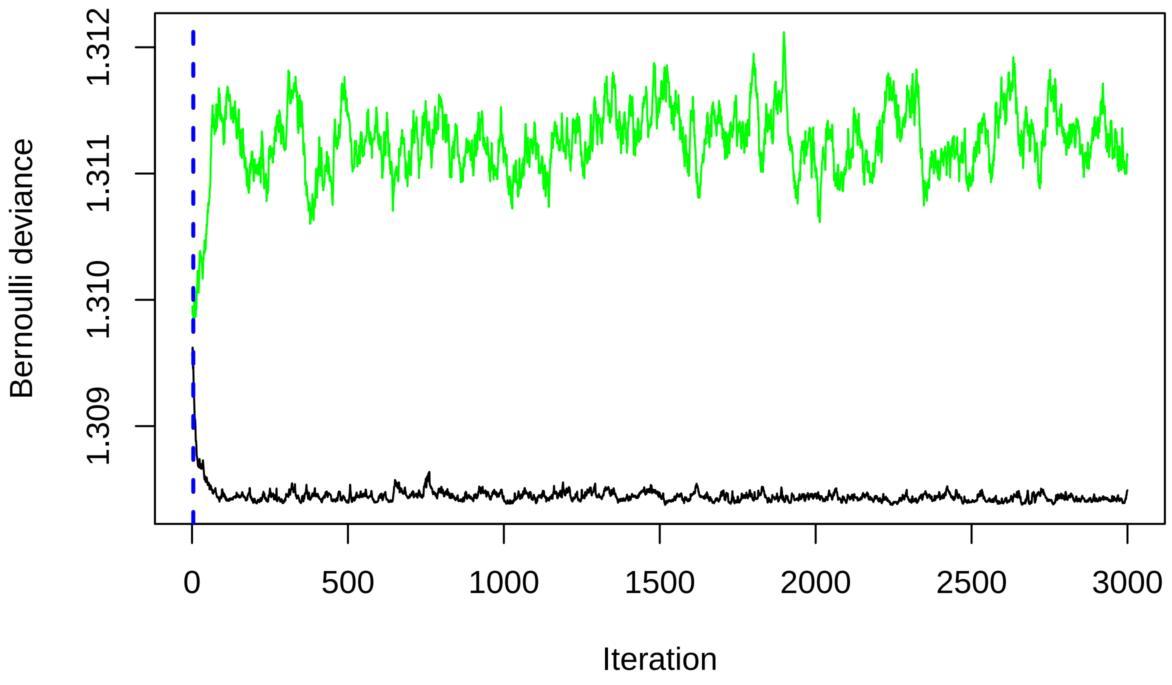
This is merely a case for illustration. The accuracy of the classifier is aggregately the same when tested for test and holdout data, for models from train and test data respectively. Although, holdout data is best preserved to test the general efficacy of the chosen model out of multiple tuned with the train data and validated with test data.

Using Cross-Validation Data

Let's fit a GBM with 7 fold cross validation and use the cross validation procedure to find the best number of trees for prediction.

```r
gbm_model_cv <- gbm(formula = class ~ .,
                            distribution = "bernoulli",
                            data = train,
                            n.trees = 3000,
                            shrinkage = .1,
                            n.minobsinnode = 200,
                            cv.folds = 7,
                            n.cores = 1)

the_chosen_one <- gbm.perf(gbm_model_cv)
```



The black line is the training bernoulli deviance and the green line is the testing bernoulli deviance. The tree selected for prediction, indicated by the vertical blue line, is the tree that minimizes the testing error on the cross-validation folds; surely it is the one with the least deviance.