1 Introduction

XGBoost is an implementation of a machine learning technique known as gradient boosting. **In this blog post, we discuss what XGBoost is, and demonstrate a pipeline for working with it in R.** We won't go into too much theoretical detail. Rather, we'll focus on application.

2 XGBoost – An Implementation of Gradient Boosting

Gradient boosting is part of a class of machine learning techniques known as ensemble methods. An ensemble method leverages the output of many weak learners in order to make a prediction. Typically, these weak learners are implemented as decision trees. While each individual weak learner might not get the answer right, on average, their combined answers should be pretty decent. What makes gradient boosting different from another popular ensemble method – Random Forest, is that the construction of the weak learners depend on the previously constructed learners.

In gradient boosting, each weak learner is chosen iteratively in a greedy manner, so as to minimize the loss function. **XGBoost is a highly optimized implementation of gradient boosting.** The original paper describing XGBoost can be found here. Although XGBoost is written in C++, it can be interfaced from R using the `xgboost` package.

To install the package:

```
install.packages("xgboost")
```

In this tutorial we use the following packages:

```
library(Matrix)
library(xgboost)
library(ggplot2)
library(ggthemes)
library(readr)
library(dplyr)
library(tidyr)
library(stringr)

theme_set(theme_economist())

set.seed(1234) # For reproducibility.
```

3 Load And Explore The Data

We will use the Titanic Dataset which we get from Kaggle. Basically, we try predict whether a passenger survived or not (so this is a binary classification problem).

Let's load up the data:

```
# Replace directoryWhichContainsTrainingData and
directoryWhichContaintsTestData
# with your path to your training and test data, respectively.
# For example mine looks like this:
```

```
directoryWhichContainsTrainingData <-  "./xg_boost_data/train.csv"
directoryWhichContaintsTestData <- "./xg_boost_data//test.csv"

titanic_train <- read_csv(directoryWhichContainsTrainingData)
titanic_test <- read_csv(directoryWhichContaintsTestData)
```

For the sake of brevity (and my laziness), I'll only keep some of the features:

- Pclass: this refers to passenger class (1st, 2nd or 3rd)
- Sex
- Age
- Embarked: Port of Embarkation - C = Cherbourg, Q = Queenstown, S = Southampton

I'll use dplyr's `select()` to do this:

```
titanic_train <- titanic_train %>%
  select(Survived,
         Pclass,
         Sex,
         Age,
         Embarked)

titanic_test <- titanic_test %>%
  select(Pclass,
         Sex,
         Age,
         Embarked)
```

Let's have a look at our data after discarding a few features:

```
str(titanic_train, give.attr = FALSE)
## tibble [891 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Survived: num [1:891] 0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass  : num [1:891] 3 1 3 1 3 3 1 3 3 2 ...
##  $ Sex     : chr [1:891] "male" "female" "female" "female" ...
##  $ Age     : num [1:891] 22 38 26 35 35 NA 54 2 27 14 ...
##  $ Embarked: chr [1:891] "S" "C" "S" "S" ...
```

XGBoost will only take numeric data as input. Let's convert our character features to factors, and one-hot encode. Since we're one-hot encoding, we expect our matrix to be filled with lots of zeroes - in other words, we expect it to be sparse. We will use `sparse.model.matrix()` to create a sparse matrix which will be used as input for our model. XGBoost has been written to take advantage of sparse matrices, so we want to make sure that we're using this feature.

Unfortunately, in using R at least, sparse.model.matrix() will drop rows which contain NA's if the global option `options('na.action')` is set to `"na.omit"`. So we use a fix outlined here:

```
previous_na_action <- options('na.action') #store the current na.action
options(na.action='na.pass') #change the na.action

titanic_train$Sex <- as.factor(titanic_train$Sex)
titanic_train$Embarked <- as.factor(titanic_train$Embarked)
```

```
#create the sparse matrices
titanic_train_sparse <- sparse.model.matrix(Survived~., data =
titanic_train)[,-1]
titanic_test_sparse <- sparse.model.matrix(~., data =
titanic_test)[,-1]

options(na.action=previous_na_action$na.action) #reset the na.action
```

Alternatively, we could have just used a sentinel value to replace the NA's.

3.1 Interacting with the Sparse Matrix Object

The data are in the format of a **dgCMatrix** class - this is the Matrix package's implementation of sparse matrix:

```
str(titanic_train_sparse)
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:3080] 0 1 2 3 4 5 6 7 8 9 ...
##   ..@ p       : int [1:6] 0 891 1468 2359 2436 3080
##   ..@ Dim     : int [1:2] 891 5
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:891] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "Pclass" "Sexmale" "Age" "EmbarkedQ" ...
##   ..@ x       : num [1:3080] 3 1 3 1 3 3 1 3 3 2 ...
##   ..@ factors : list()
```

We can check the dimension of the matrix directly:

```
dim(titanic_train_sparse)
## [1] 891   5
```

The names are the features are given by `titanic_train_sparse@Dimnames[[2]]`:

```
head(titanic_train_sparse@Dimnames[[2]])
## [1] "Pclass"    "Sexmale"   "Age"       "EmbarkedQ" "EmbarkedS"
```

If needed, you can convert this data (back) into a data frame, thusly:

```
train_data_as_df <- as.data.frame(as.matrix(titanic_train_sparse))
```

4 Hyperparameters

Tuning hyperparameters is a vast topic. Without going into too much depth, I'll outline some of the more commonly used hyperparameters:

Full reference: https://xgboost.readthedocs.io/en/latest/parameter.html

Table 4.1: Parameters

Parameter	Explanation
eta	default = 0.3 learning rate / shrinkage. Scales the contribution of each try by a factor of 0 < eta < 1
gamma	default = 0 minimum loss reduction needed to make another partition in a given tree. larger the value, the more conservative the tree will be (as it will need to make a bigger reduction to split) So, conservative in the sense of willingness to split.

Parameter	Explanation
max_depth	default = 6 max depth of each tree…
subsample	1 (ie, no subsampling) fraction of training samples to use in each “boosting iteration”
colsample_bytree	default = 1 (ie, no sampling) Fraction of columns to be used when constructing each tree. This is an idea used in RandomForests
min_child_weight	default = 1 This is the minimum number of instances that have to been in a node. It’s a regularization parameter So, if it’s set to 10, each leaf has to have at least 10 instances assigned to it. The higher the value, the more conservative the tree will be.

Note that the above hyperparameters are in the case of the weak learner being a tree. It is possible to have linear models as your weak learners.

Let’s create the hyper-parameters list:

```
# booster = 'gbtree': Possible to also have linear boosters as your
weak learners.
params_booster <- list(booster = 'gbtree', eta = 1, gamma = 0,
max.depth = 2, subsample = 1, colsample_bytree = 1, min_child_weight =
1, objective = "binary:logistic")
```

One day, I shall write a blog post about various ways to tune your hyperparameters. But, today is not that day. If you are like me, and believe in serendipitous machine learning, then **try a random search of the hyperparameters** (within reason, of course. Don’t set `max.depth = 9999999999`. Or do, I’m not telling you how to live your life). You get surprisingly decent results.

5 Training The Model: Or, how I learned to stop overfitting and love the cross-validation

The `xgb.train()` and `xgboost()` functions are used to train the boosting model, and both return an object of class xgb.Booster. `xgboost()` is a simple wrapper for `xgb.train()`. `xgb.train()` is an advanced interface for training the xgboost model. We’re going to use `xgboost()` to train our model. Yay.

One of the parameters we set in the `xgboost()` function is `nrounds` - the maximum number of boosting iterations. So, how many weak learners get added to our ensemble. If we set this parameter too low, we won’t be able to model the complexity of our dataset very well. If we set it too high, we run the risk of overfitting. **We always need to be wary of overfitting our model to our training data.**

This leads us to the xgb.cv() function. **Let’s use xgb.cv() to determine how many rounds we should use for training.** Important to note that xgb.cv() returns an object of type xgb.cv.synchronous, not xgb.Booster. So you won’t be able to call functions like xgb.importance() on it, as xgb.importance() takes object of class xgb.Booster **not** xgb.cv.synchronous.

```
# NB: keep in mind xgb.cv() is used to select the correct hyperparams.
# Here I'm only looking for a decent value for nrounds; We won't do
full hyperparameter tuning.
# Once you have them, train using xgb.train() or xgboost() to get the
final model.
```

```
bst.cv <- xgb.cv(data = titanic_train_sparse,
                 label = titanic_train$Survived,
                 params = params_booster,
                 nrounds = 300,
                 nfold = 5,
                 print_every_n = 20,
                 verbose = 2)

# Note, we can also implement early-stopping: early_stopping_rounds =
3, so that if there has been no improvement in test accuracy for a
specified number of rounds, the algorithm stops.
```

Using the resuts of xgb.cv(), let's plot our validation error and training error against the number of round:

```
res_df <- data.frame(TRAINING_ERROR = bst.cv$evaluation_log$train_
error_mean,
                     VALIDATION_ERROR = bst.cv$evaluation_log$test_
error_mean, # Don't confuse this with the test data set.
                     ITERATION = bst.cv$evaluation_log$iter) %>%
  mutate(MIN = VALIDATION_ERROR == min(VALIDATION_ERROR))

best_nrounds <- res_df %>%
  filter(MIN) %>%
  pull(ITERATION)

res_df_longer <- pivot_longer(data = res_df,
                              cols = c(TRAINING_ERROR,
VALIDATION_ERROR),
                              names_to = "ERROR_TYPE",
                              values_to = "ERROR")

g <- ggplot(res_df_longer, aes(x = ITERATION)) +        # Look @ it
overfit.
  geom_line(aes(y = ERROR, group = ERROR_TYPE, colour = ERROR_TYPE)) +
  geom_vline(xintercept = best_nrounds, colour = "green") +
  geom_label(aes(label = str_interp("${best_nrounds} iterations gives
minimum validation error"), y = 0.2, x = best_nrounds, hjust = 0.1)) +
  labs(
    x = "nrounds",
    y = "Error",
    title = "Test & Train Errors",
    subtitle = str_interp("Note how the training error keeps decreasing
after ${best_nrounds} iterations, but the validation error starts
\ncreeping up. This is a sign of overfitting.")
  ) +
  scale_colour_discrete("Error Type: ")

g
```

This leads us to believe that we should use 9 as the value for `nrounds`. Let's train our XGBoost model:

```
bstSparse <- xgboost(data = titanic_train_sparse, label =
titanic_train$Survived, nrounds = best_nrounds, params =
params_booster)
## [1]  train-error:0.207632
## [2]  train-error:0.209877
## [3]  train-error:0.189675
## [4]  train-error:0.173962
## [5]  train-error:0.163861
## [6]  train-error:0.166105
## [7]  train-error:0.166105
## [8]  train-error:0.162739
## [9]  train-error:0.156004
```

6 Making Predictions

Now that we have our model, we can use it to make predictions:

```
titanic_test <- read_csv(directoryWhichContaintsTestData)

predictions <- predict(bstSparse, titanic_test_sparse)
titanic_test$Survived = predictions

titanic_test <- titanic_test %>% select(PassengerId, Survived)
titanic_test$Survived = as.numeric(titanic_test$Survived > 0.5)


# write_csv(titanic_test, "./xg_boost_data/sb.csv")
# I submitted the above to Kaggle, and got a score of 0.77751 (this is
the categorization accuracy)
# No discussion of AUC, precision / recall. One day, I'll blog about
this as well.... Maybe
```

7 Conclusion

In this blog post we saw how to put together a pipeline to implement XGBoost. We saw how to create a sparse matrix, do cross validation, create the actual model, and finally, make predictions.

There's a lot here that I didn't cover. Things like feature importance, for example. The plan is to write a Part II, Some Day, which goes into more detail. Really, I just intend this to be a handy reference for future me.
If you see any errors in this post, please let me know :). I'll try fix them.