Using Statistical Methods and Data Mining to Uncover Trends in a Dataset:

Gradient Boosting Machines with h2o

William Bachmai Foote
STATS 101C, Fall '21 Final Project
UCLA Department of Statistics
UID: 305134696
Professor Chenlu Shi

Introduction

Given a numeric dataset, different data mining techniques and statistical methods can be applied to analyze the relationship between the predictor variable(s) and the response. But, how do these methods with their various parameters hold up in different circumstances with different datasets – even as the new data comes from the same population as the original? Problems arise when given a training set with predictor and response variables, but a test set with only the predictor columns. If fitting too close to the training data overfitting may result. However, we could see how our predicted values compared to the response variable portion of the test dataset using 5 submission attempts per day on Kaggle to. We could score our models against the Kaggle test data, but not have individual values revealed on Kaggle. Even still, another partition of the test data which contained even more unknown response values would provide our final score once the Kaggle competition ended. The problem then is, how can we compare different statistical methods and data mining techniques with limited resources (5 submissions per day) and even how methods compared to each other with model parameters slightly tweaked? To address these issues I used K-fold cross validation to minimize bias of my models as I tuned model parameters. In selecting my final model, a Gradient Boosting Machine, I compared the different Root Mean Squared Error (RMSE) using the Kaggle assigned score under the assumption that the testing data on the “Public Leaderboard” should be for the most part representative of the “Private Leaderboard” test score.

Methods

I combined methods that we discussed in class as well as some various packages that build on the concepts used by the packages we talked about in lectures. For one, I relied heavily on K-Fold cross validation because of the nature of the competition. Because we did not have the Y-values of either of the test datasets, we could really only validate our findings 5 times per day based on the Kaggle submission rules. Should I have tried to validate only against the training data, I would have gotten great RMSE scores that would likely have been overfit when it came to the testing data. K-fold cross validation attempted to solve this issue by splitting the data into 10 (I used k=10) partitions and using a different one of the partitions as a quasi-test data set over 10 iterations of the model. Using the mean RMSE of the k-folds, I was able to get a less biased estimate of the RMSE, theoretically. When I was confident in my results, I submitted my predicted values to Kaggle and was given a test RMSE score that for the most part aligned with my results from the cross validation.

In the actual model making, I compared methods such as logistic regression, random forests, ridge regression, bagging, and support vector machines. Personally, I was most comfortable and had the most success with Gradient Boosting Machines which we talked about in lecture 8.1. While we identified how to use GBM’s using AdaBoost, the numeric/integer nature of the data required using GBM. AdaBoost only works for binary classification problems, while GBM can work for both regression and classification problems. I began using the **`gbm`** package that I used in the homework and that the textbook used in Chapter 8, but was unsatisfied by the lack of parameters that I could tune. Table 1 below compares the tools that were at my disposal with **`gbm`** and **`h2o.gbm`**. I won’t discuss the parameters one might tune in the package we used in class materials.

However, all of the three tuning parameters that were used in **h2o.gbm** exclusively, including Sample Rate, Column Sample Rate, and Column Sample Rate per Tree, likely helped reduce overfitting. The three parameters introduce a stochastic sampling aspect that can help generalization (i.e. less overfitting when applied to the testing data). Friedman (1999) explains that test accuracy improves when either the rows or columns are sampled. The idea is similar to k-fold CV in that by not looking at all aspects of the training data and instead looking at

partitions of it over multiple iterations, there is an added stochastic aspect that would mirror a testing dataset. By only fitting a model in a given iteration (note: making one tree is one iteration) to a certain number of rows or columns, the model theoretically should be better at generalizing to the test data. Specifically, I used both methods. sample_rate samples a proportion of the rows per tree, col_sample_rate a proportion of the columns, and col_sample_rate_per_tree a proportion of the columns per tree.

Finally, I chose `**h2o.gbm**` over `**gbm**` because of its hyperparameter grid-searching. In gbm, to tune all the listed parameters, we would construct a Cartesian grid of parameters using `**caret**` and then make a new model for each of the combinations. For four parameters with three different parameters each, there would be 3^4 = 81 combinations of models to make. Adding in k-fold cross validation makes this extremely computationally expensive. Ultimately, this is why I chose h2o.gbm which adds an easy to use method called Random Discrete Grid Searching. Instead of looking at each combination of models, h2o's grid searching randomly jumps from one combination of parameters to another given a grid of hyperparameters, stopping at a given criteria (i.e. performance, max number of models, and/or max number of seconds). I specified the stopping crieria to be a .001% improvement over the last 5 rounds, 100 models, and 600 seconds. As Boehmke notes "Although using a random discrete search path will likely not find the optimal model, it typically does a good job of finding a very good model." With the more efficient searching algorithm, I could also use k-fold cross validation because the computational cost of doing so was not as high as with `**caret**`.

Table 1: Comparing `gbm` and `h2o.gbm`		
Tool/Tunable Parameter	**`gbm`**	**`h2o.gbm`**
Cross Validation	✅	✅
Min no. of observations in the terminal nodes of trees	✅	✅
Learn Rate	✅	✅
Interaction Depth (Max) (i.e. the max level of variable interaction allowed)	✅	✅
Number of Trees	✅	✅
Row Sample Rate		✅
Column Sample Rate		✅
Column Sample Rate per Tree		✅
Easy Interface for Grid Searching		✅

Table 1: Comparing gbm and h2o.gbm. For my purposes, h2o.gbm had a few more parameters to tune and an easier interface to tune parameters.

Results

After completing this hyperparameter grid search, I settled on a set of model parameters that compared favorably with other models that the grid search created by comparing k-fold cross validation RMSE, and one that compared favorably with other models based on the Kaggle-assigned RMSE performance score on the public partition of the testing data. My final choice of h2o.gbm over other methods was a result of how well it did on the Public Score.

Using the h2o grid search, 105 GBM models were looked at with 10-fold cross validation for each model. The best model from the grid search had a k-fold cross validation RMSE of

1.238419, which was actually a pretty close estimate of the 1.23721 value given by the Private Score RMSE from Kaggle.

Table 2: Final h2o.gbm Model	
Tool/Tunable Parameter	**`h2o.gbm`**
Min observations in leaves	10
Learn Rate	0.00644
Interaction Depth (Max)	17
Number of Trees	1330
Row Sample Rate	.6
Column Sample Rate	.85
Column Sample Rate per Tree	.4
10-Fold Cross Validation RMSE	1.23842

Table 2: The parameters for the final h2o.gbm model.

Discussion/Conclusion

Overall, the model that was determined to be used for the final grading score was sufficient, but not the best it could be. Using k-fold cross validation and random discrete grid searching in h2o's interface of Gradient Boosting Machines, a model that performed well was created. However, bias was an issue as the private score for this iteration of the h2o GBM that I trained had a much higher Private Score RMSE than Public Score RMSE. For some models like bagging, these scores were practically the same. Even for other iterations of the h2o GBM that I scored but didn't select as my final model had smaller differences between Public and Private scores and lower Private Scores altogether (than either or both of the respective public score and the final model's Private Score).

To alleviate this issue, I want to investigate changing the number of folds looked at in k-fold cross validation and potentially increase the max run time. If I decrease the k-folds from 10 to 5 for example, I can look at significantly more models in the same amount of time. Because I only tapped into a portion of the models (as opposed to if I had done a Cartesian grid search), it could be help to look at a larger portion or all of the models with less folds and then increase the number of folds when I have a better picture of what the parameters should be. For example, if I decide that of the 10 sample rates I look for in the first iteration using k=5, 3 sample rates show up routinely in the best models, I might do another iteration with k=10 and only look at the 3 sample rates.

If given unlimited time and computational power, I would use more k-folds and do more hyperparameter grid searches where each grid search is dependent on and informed by the last. I would also use the same amount of folds for each different method so that I could easily compare RMSE's to use a less biased estimate of Private RMSE. Selecting my final model compared to models of the same or different method using solely Public RMSE was a flaw in my method because Public RMSE is not necessarily indicative of Private RMSE. Uniform k-fold cross validation would likely give me a better perspective on how to compare methods and models.

Appendix

```
# I have 16 gb of RAM and was instructed to let h2o only use a max of a portion of that.
# h2o.init initializes the h2o interface with runs using Java

h2o.init(max_mem_size = "10G")

# Recipe allows me to get the data in a cleaner format if necessary
blueprint <- recipe(Y ~ ., data = train) %>%
  step_other(all_nominal(), threshold = 0.005)


# Create training & test sets for h2o
train_h2o <- prep(blueprint, training = train, retain = TRUE) %>%
  juice() %>%
  as.h2o()
test_h2o <- prep(blueprint, training = train) %>%
  bake(new_data = test) %>%
  as.h2o()

# h2o needs the names of the variables in a specific format, which I do below
Y <- "Y"
X <- setdiff(names(train), c("Y", "Id"))

# Construct a large Cartesian hyper-parameter space
ntrees_opts = c(10000)      # early stopping will stop earlier
max_depth_opts = seq(1,20)
min_rows_opts = c(1,5,10,20,50,100)
learn_rate_opts = seq(0.001,0.05, length.out = 10)
sample_rate_opts = seq(0.3,1,0.05)
col_sample_rate_opts = seq(0.3,1,0.05)
col_sample_rate_per_tree_opts = seq(0.3,1,0.05)

# Make this into a list
hyper_params = list(ntrees = ntrees_opts,
                    max_depth = max_depth_opts,
                    min_rows = min_rows_opts,
                    learn_rate = learn_rate_opts,
                    sample_rate = sample_rate_opts,
                    col_sample_rate = col_sample_rate_opts,
                    col_sample_rate_per_tree = col_sample_rate_per_tree_opts
)

# Search a random subset of these hyper-parmameters. Max runtime
# and max models are enforced, and the search will stop after we
# don't improve much over the best 5 random models.
search_criteria = list(strategy = "RandomDiscrete",
                       max_runtime_secs = 600,
                       max_models = 100,
                       stopping_metric = "AUTO",
```

```
                  stopping_tolerance = 0.00001,
                  stopping_rounds = 5,
                  seed = 123456)

gbm_grid <- h2o.grid("gbm",
                  grid_id = "mygrid",
                  x = X,
                  y = Y,
                  # faster to use a 80/20 split
                  # alternatively, use N-fold cross-validation:
                  training_frame = train_h2o,
                  nfolds = 10,
                  # Gaussian is best for MSE loss, but can try
                  # other distributions ("laplace", "quantile"):
                  distribution="gaussian",
                  # stop as soon as mse doesn't improve by
                  # more than 0.1% on the validation set,
                  # for 2 consecutive scoring events:
                  stopping_rounds = 2,
                  stopping_tolerance = 1e-3,
                  stopping_metric = "MSE",
                  # how often to score (affects early stopping):
                  score_tree_interval = 100,
                  ## seed to control the sampling of the
                  ## Cartesian hyper-parameter space:
                  seed = 123456,
                  hyper_params = hyper_params,
                  search_criteria = search_criteria)

# Sort the results by RMSE
gbm_sorted_grid <- h2o.getGrid(grid_id = "mygrid", sort_by = "rmse")
print(gbm_sorted_grid)

# Save the best model as an object, and look at the parameters
# Also shows how it performed for k-folds

best_model <- h2o.getModel(gbm_sorted_grid@model_ids[[1]])
summary(best_model)

# Look at how many models were made
length(gbm_sorted_grid@model_ids)
as.data.frame(summary(gbm_sorted_grid))

# The following is code for submitting to Kaggle
# pred.boost.7 <- h2o.predict(best_model, test_h2o)
# sub.boost.7 <- cbind(test$Id, as.matrix(pred.boost.7))
# colnames(sub.boost.7) <- c("Id", "pred")
# write.csv(sub.boost.7, "sub_boost_final.csv", row.names = FALSE)
```

References:

Boehmke, B. *Gradient Boosting Machines · UC Business Analytics R Programming Guide*. UC-r.github.io. Retrieved 10 December 2021, from http://uc-r.github.io/gbm_regression#h2o.

Friedman, J. (1999). Jerryfriedman.su.domains. Retrieved 10 December 2021, from https://jerryfriedman.su.domains/ftp/stobst.pdf.

Gradient Boosting Machine (GBM) — H2O 3.34.0.4 documentation. H2O.ai. Retrieved 10 December 2021, from https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/gbm.html.

Methodology

Three main methods led to me creating my final model, which used the technique of boosting via the **gbm** package in R. These methods were underscored by the bias-variance tradeoff which I attempted to address by splitting my training data further into a training_beta and testing_beta dataset as a comparison technique for different methods and by using k-fold cross validation to compare models within methods. I also focused on three specific methods, including bagging, boosting, and

1. Bias-Variance
    a. When only given a training dataset and the x-values of the testing dataset, it was difficult to estimate testing error rates. As a result, without proper measures in place like the two I attempted to enlist above, the models one might create could be too focused on the training data and lead to overfitting.
    b.
2. Different Models
3. Tuning Parameters

As I'll outline in this report, I first began by creating training and testing datasets of my own (called train_beta and test_beta) to see how my results held up before submitting – the 5 submissions per day limit proved to be a difficult obstacle. If one focuses on metrics that can score the performance of a model on a training dataset, such as Mean Squared Error, issues like overfitting can occur when the model is applied to a testing dataset. By creating my own train-test split from the given training dataset, I tried to minimize this issue. Another technique that I used frequently was k-fold cross validation Furthermore, using techniques like cross-validation, I further aimed to create models that held up under the bias-variance tradeoff lens.

Results

Appendix

References