How Much Did It Rain – Kaggle Competition

Tyler Byers

December 6, 2015

About

This is Part 3 of 3 for the final project for the University of Washington Professional and Continuing Education Data Science certificate class #3 of 3. I chose the Kaggle competition "How Much Did it Rain" (https://www.kaggle.com/c/how-much-did-it-rain-ii). I did not work with any team members for this project. For all my project files, please see my repo on GitHub: https://github.com/tybyers/kaggle_howmuchdiditrain.

Competition Description and Summary

This competition was to predict how much it rained in a given location based on radar readings. Because the amount of rainfall can be highly localized to an area, and because it is impossible to put a rain gauge in every location, or even in a lot of different locations, calculating the amount of rainfall a certain location received can be important. Since these data in particular are for midwestern corn-growing states, estimating the amount of rain a certain farm received can be important – whether to check for crop damage, or to plan harvest, or perhaps do localized flooding mitigation. If we can use radar to accurately calculate the rainfall in a given location, it may help the farmers with these problems. This has applicability to more than just farms too.

This is a supervised learning problem, because we have a target variable `Expected` amount of rain in mm in an hour. We use the radar measurements to predict the target variable. The evaluation metric is Mean Absolute Error (MAE) for the Kaggle competition.

There were many parts of this competition that made it difficult and different from a typical Kaggle competition. For example:

- There are a lot of values for the target variable, `Expected`, that are outliers. We have to filter these out for training purposes; *however* these values are still considered in the final scoring.
- The goal is to predict how many mm of rain fell at a rain gauge in a single hour. However, for each hour, we can have several radar observations. So, in essence, we sort of have to aggregate many observations down to one hour.
- Around 38% of our data has must be filtered out for training purposes. This is explained more later.
- A very large percentage of total radar observations across the different variables (63% of total values are NA).

Data

The data set for this competition is rather large, and one of the challenges was getting the data into a manageable format. Some facts about the data:

- Observations in `train` data: 13,765,201.
- Columns in `train` data: 24 (including target variable).
- Unique `Id` values in `train` data: 1,180,845 (so 1.2 million "hours" of data).
- Observations in `test` data: 8,022,756
- Unique `Id` values in `test` data: 717,625.

A description of the data set is below, as reprinted from the competition data page the data columns are:

- Id: A unique number for the set of observations over an hour at a gauge.
- minutes_past: For each set of radar observations, the minutes past the top of the hour that the radar observations were carried out. Radar observations are snapshots at that point in time.
- radardist_km: Distance of gauge from the radar whose observations are being reported.
- Ref: Radar reflectivity in km
- Ref_5x5_10th: 10th percentile of reflectivity values in 5x5 neighborhood around the gauge.
- Ref_5x5_50th: 50th percentile
- Ref_5x5_90th: 90th percentile
- RefComposite: Maximum reflectivity in the vertical column above gauge. In dBZ.
- RefComposite_5x5_10th
- Ref Composite_5x5_50th
- RefComposite_5x5_90th
- RhoHV: Correlation coefficient (unitless)
- RhoHV_5x5_10th
- RhoHV_5x5_50th
- RhoHV_5x5_90th
- Zdr: Differential reflectivity in dB
- Zdr_5x5_10th
- Zdr_5x5_50th
- Zdr_5x5_90th
- Kdp: Specific differential phase (deg/km)
- Kdp_5x5_10th
- Kdp_5x5_50th
- Kdp_5x5_90th
- **Expected**: Actual gauge observation in mm at the end of the hour. (This is the *target* variable).

Scoring

The scoring for this competition is Mean Absolute Error: `mean(abs(actual - prediction))`, which is a simple metric. However, we couldn't score based on *all* the data – it had to be done only for those `Id` hours where all the `Ref` values were non-null. For more information see the Kaggle forum discussion: (https://www.kaggle.com/c/how-much-did-it-rain-ii/forums/t/16622/ignored-ids), which says:

> Please note that the MAE is now being computed only on Ids where at least one of the Ref values is non-null. In other words, it doesn't matter what value you predict for those Ids. All previous submissions have been rescored.

Modeling

This section explains the steps I went through to transform the data and create a model for submission.

Marshall-Palmer Transformation

One of the first things I did was to apply the Marshall-Palmer Transformation to all of my `Ref` values in the dataset (I did this for all the features that have `Ref` in them). The Marshall-Palmer is a single-pol rainrate equation, and the transformation equation can be applied with the following R code: `(10^(Ref/10)/200)^(5/8)`. For more information, see the Kaggle forum.

Flattening Observations

After applying the Marshall-Palmer transformation, I had to "flatten" the gauge-minute observations into gauge-hour observations.

For example, if we look at gauge-hour Id = 23513. Below is the first 5 columns from that data set:

```
##        Id minutes_past radardist_km  Ref Ref_5x5_10th Ref_5x5_50th
## 1  23513            2            7 24.0         21.0         24.0
## 2  23513            6            7 25.5         22.5         27.0
## 3  23513           10            7 17.5         17.5         25.0
## 4  23513           15            7 28.0         24.5         27.0
## 5  23513           20            7 30.0         27.5         30.5
## 6  23513           24            7 32.5         28.0         32.0
## 7  23513           29            7 30.0         28.5         32.0
## 8  23513           34            7 30.5         28.5         31.5
## 9  23513           38            7 37.5         30.5         35.0
## 10 23513           43            7 33.0         32.0         33.5
## 11 23513           48            7 34.5         31.5         35.0
## 12 23513           53            7 34.5         32.5         35.0
## 13 23513           58            7 36.0         29.5         34.0
```

Figure 1 shows the `Ref` column plotted versus `minutes_past` for this gauge-hour. Note this is raw `Ref`, not transformed via Marshall-Palmer.

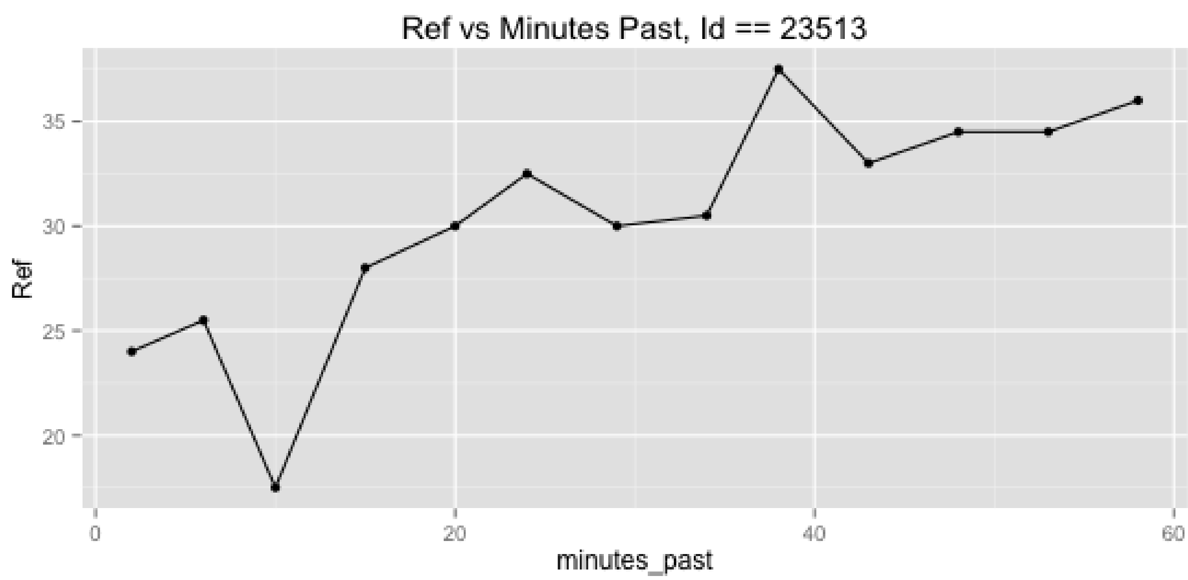

Figure 1: Ref vs. minutes_past For Id = 23513

Now, there are many ways we could "flatten" this data into a single gauge-hour reading. The way I decided to do so was to find the area under the curve, setting anchors at points (-0.01,0), (60.01, 0). I used the `auc` function from the `MESS` package to do these calculations – and made them pretty fast using `dplyr` command `group_by(Id) %>% summarise(...)`. These anchor choices may not be appropriate, but some of the gauge readings might have only had one valid point per hour, so we had to start somewhere.

In the formulas I show in my models below, I denote these transformed variables with a `_auc` appended to the end of the original feature name (i.e. `Ref` becomes `Ref_auc`).

Adding Inverse km^2

Another variable I added was the inverse of the `radardist_km^2`. I realized that radar is likely affected by distance in an inverse-square fashion (like intensity of sound or light), so added this as a variable, and it helped when I was doing cross-validation.

Changing Target Variable Distribution

Another transformation I did which resulted in much better results was to change the distribution of the data. After filtering out `Expected` values of greater than 350, which were bad readings, I noticed that the distribution frequency for the `Expected` values was strongly skew-right, as shown in Figure 2.

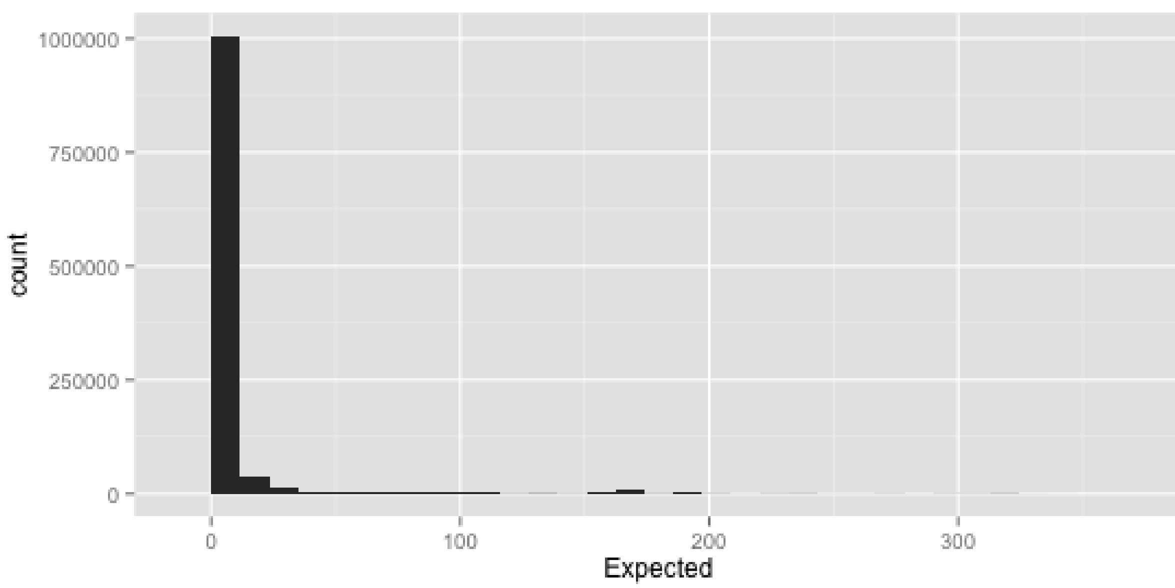

Figure 2: Histogram of Expected (Filtered out Expected > 350)

But, if we take the log of `Expected` (plus 1 so there isn't an error for `Expected = 0`), then we get the distribution in Figure 3. This distribution is more near-normal, which helps give us much better predictive ability.

Cross Validation

When testing models, I performed a 10-fold cross validation technique, and averaged the MAE from each fold's model. One mistake I made here was to not set the random seed every time I entered into the CV function – this would have made results reproducible and comparible across folds.

The code for my CV functions may be found in Appendix A – I had a few different such functions. Each of these functions starts with a name like `run_mods`.

CART Model

One of my first submitted models was a CART decision tree model. I used the formula shown below. My 10-fold CV score for this model was 23.3009 and my Kaggle score was 23.93418.

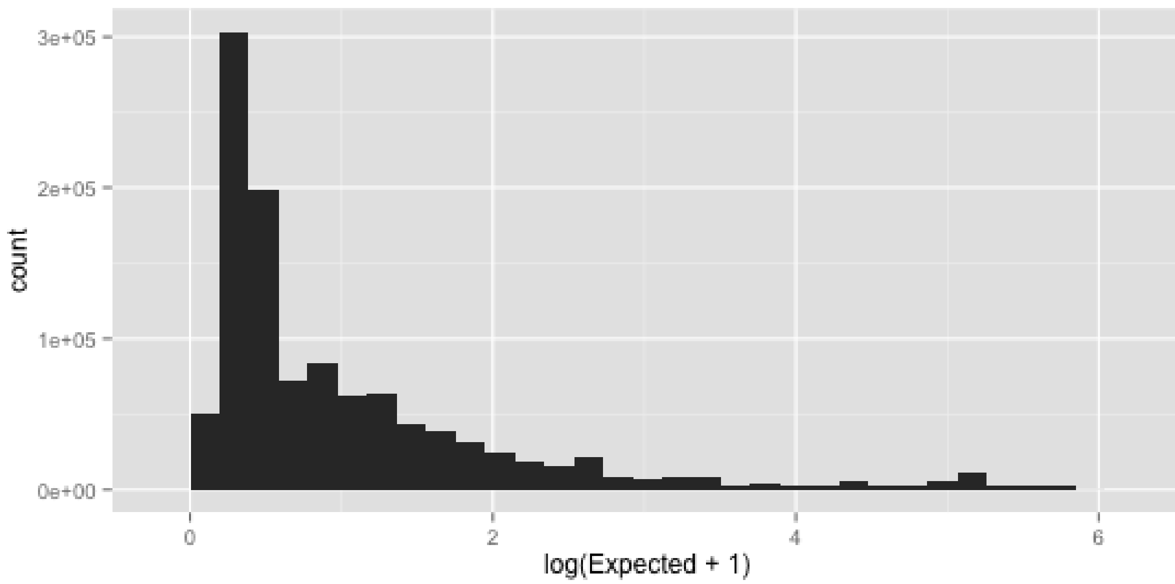

Figure 3: Histogram of log(Expected) (Filtered out Expected > 350)

```
fol_expectedlog = Expected_log ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_5x5_10th_auc +
  RefComposite_5x5_50th_auc
cart <- rpart(fol_expectedlog, data = train_allref_auc_filtered)
```

Figure 4 shows a plot of my final decision tree model that I used for submission. Clearly the tree did not use all of the features (only 3 of them!), and it only split the target into four different values, so it's not a great model. Please note that the values for the decision tree are the log of the Expected values – need `exp(x) - 1` to remove log. A summary of CART model is shown in Appendix B.

```
mycart <- readRDS('./final_cart_model.rds')
prp(mycart)
```

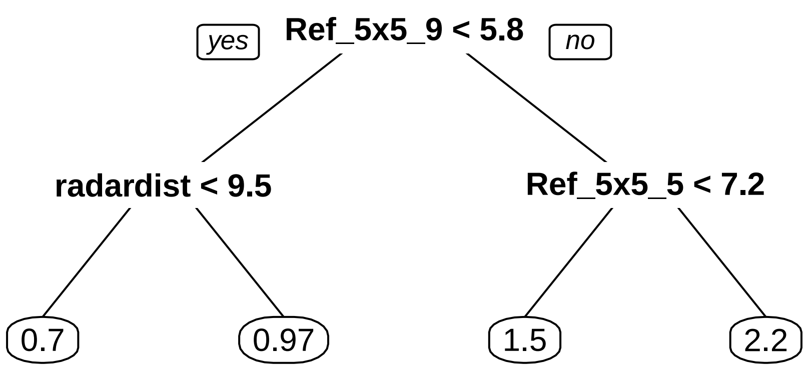

Figure 4: CART Decision Tree for Submitted CART Model

GBM Model

I then turned to a gradient boosted machine, or `gbm` model. At first I got very poor results – much worse than for the CART model. Then I played around with the distributions, and once I changed the distribution of the target variable to `log(Expected)` and chose a `gaussian` instead of `laplace` distribution, results started getting better.

GBM Grid Search with Cross Validation To try to get the best gbm model possible, I created a grid search using `interaction.depth = c(1,2,3,4)`, `n.trees = c(100, 250, 500)`, and `shrinkage = c(0.001, 0.01, 0.05, 0.1)`. I completed a 10-fold cross-validation using the following formula:

```
fol_thruKdp_gbm <- fol <- Expected_log ~ radardist_km + radardist_km_sq_inv +
  Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc + RhoHV_auc + Zdr_auc + Kdp_auc
```

The results for the grid search I completed are below (this took about 36 hours to run all of these):

```
##    n.trees interaction.depth shrinkage      MAE
## 1      100                 1     0.001 23.57116
## 2      250                 1     0.001 23.53247
## 3      500                 1     0.001 23.48102
## 4      100                 2     0.001 23.56789
## 5      250                 2     0.001 23.52247
## 6      500                 2     0.001 23.46112
## 7      100                 3     0.001 23.56339
## 8      250                 3     0.001 23.51329
## 9      500                 3     0.001 23.44369
## 10     100                 4     0.001 23.56118
## 11     250                 4     0.001 23.50634
## 12     500                 4     0.001 23.43548
## 13     100                 1     0.010 23.40711
## 14     250                 1     0.010 23.31699
## 15     500                 1     0.010 23.25450
## 16     100                 2     0.010 23.37259
## 17     250                 2     0.010 23.26084
## 18     500                 2     0.010 23.20845
## 19     100                 3     0.010 23.35023
## 20     250                 3     0.010 23.23798
## 21     500                 3     0.010 23.19598
## 22     100                 4     0.010 23.33892
## 23     250                 4     0.010 23.22756
## 24     500                 4     0.010 23.18892
## 25     100                 1     0.050 23.25326
## 26     250                 1     0.050 23.20077
## 27     500                 1     0.050 23.19037
## 28     100                 2     0.050 23.21046
## 29     250                 2     0.050 23.18260
## 30     500                 2     0.050 23.17212
## 31     100                 3     0.050 23.19611
## 32     250                 3     0.050 23.17572
## 33     500                 3     0.050 23.16249
```

```
## 34      100                 4     0.050 23.18832
## 35      250                 4     0.050 23.16998
## 36      500                 4     0.050 23.15382
## 37      100                 1     0.100 23.20996
## 38      250                 1     0.100 23.18989
## 39      500                 1     0.100 23.18642
## 40      100                 2     0.100 23.18565
## 41      250                 2     0.100 23.17198
## 42      500                 2     0.100 23.15993
## 43      100                 3     0.100 23.17971
## 44      250                 3     0.100 23.16098
## 45      500                 3     0.100 23.14826
## 46      100                 4     0.100 23.17248
## 47      250                 4     0.100 23.15499
## 48      500                 4     0.100 23.14181
```

Figure 5 shows the curves for the above data – lines are grouped by n.tree/shrinkage pairs. We can see that the best MAEs occur with more trees, higher shrinkage, and more interaction depth. We probably could have gone a little farther with the interaction depth and number of trees, but the gain was definitely leveling out.

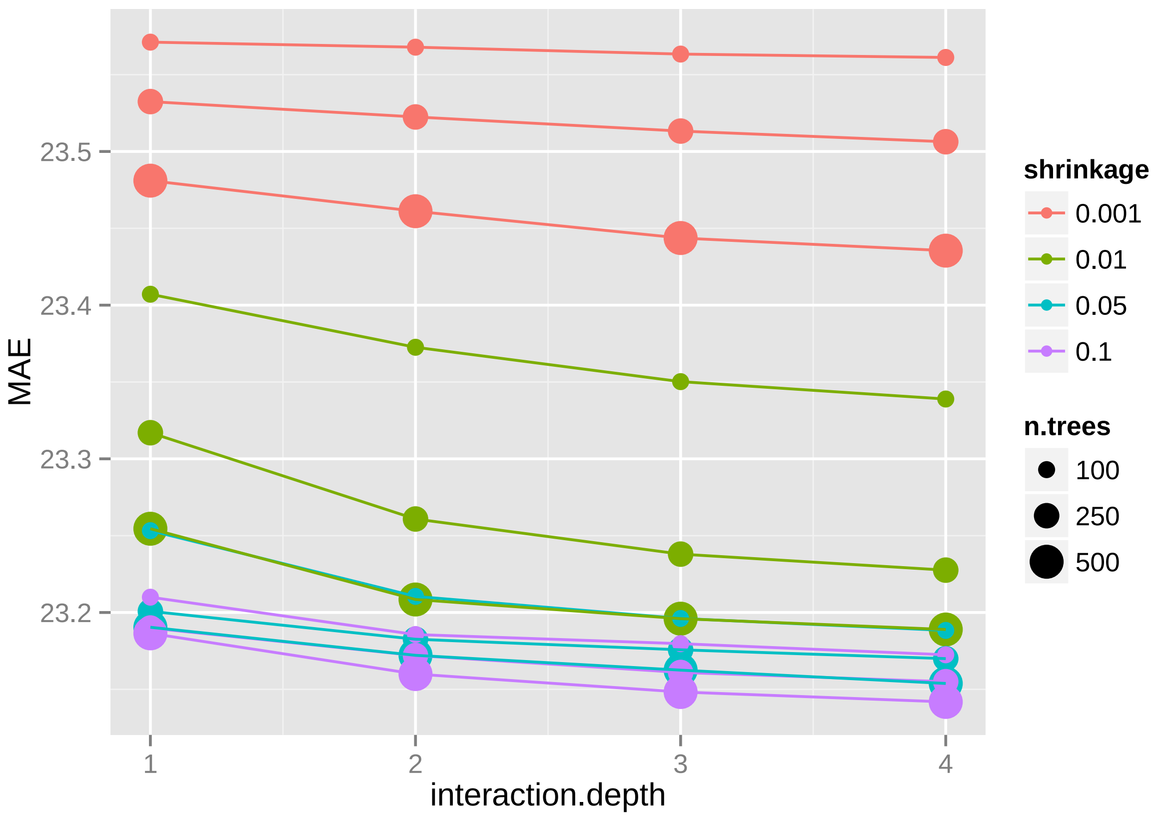

Figure 5: 10-Fold CV MAE Values for Given GBM Tunes

Figure 6 shows a zoomed-in view of the best several models, since on the previous scale we couldn't see the change very well.

Final GBM Model Based on this, I chose my final gbm model to be , , and . Figure 7 is the model summary – basically the variable importances.

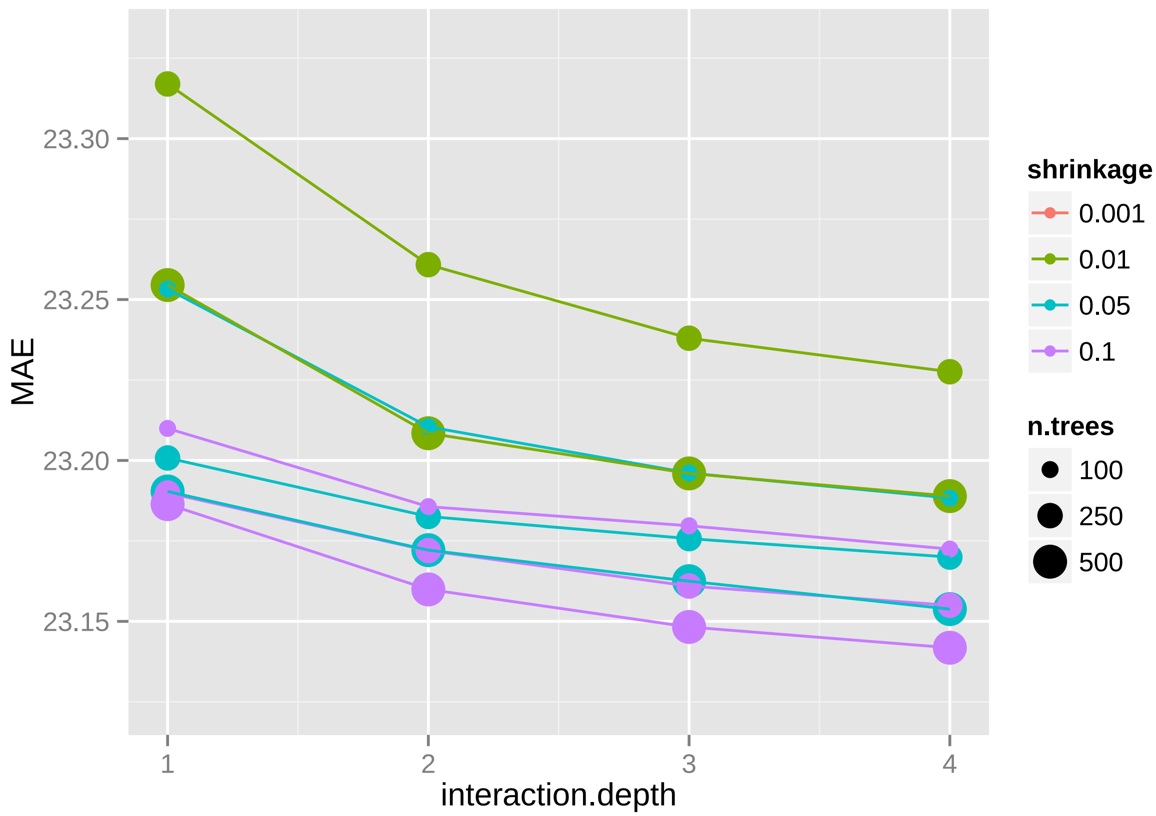

Figure 6: 10-Fold CV MAE for Best GBM Tunes

```r
mygbm <- readRDS('./gbm_mod_500_4_0.1.rds')
summary(mygbm)
```

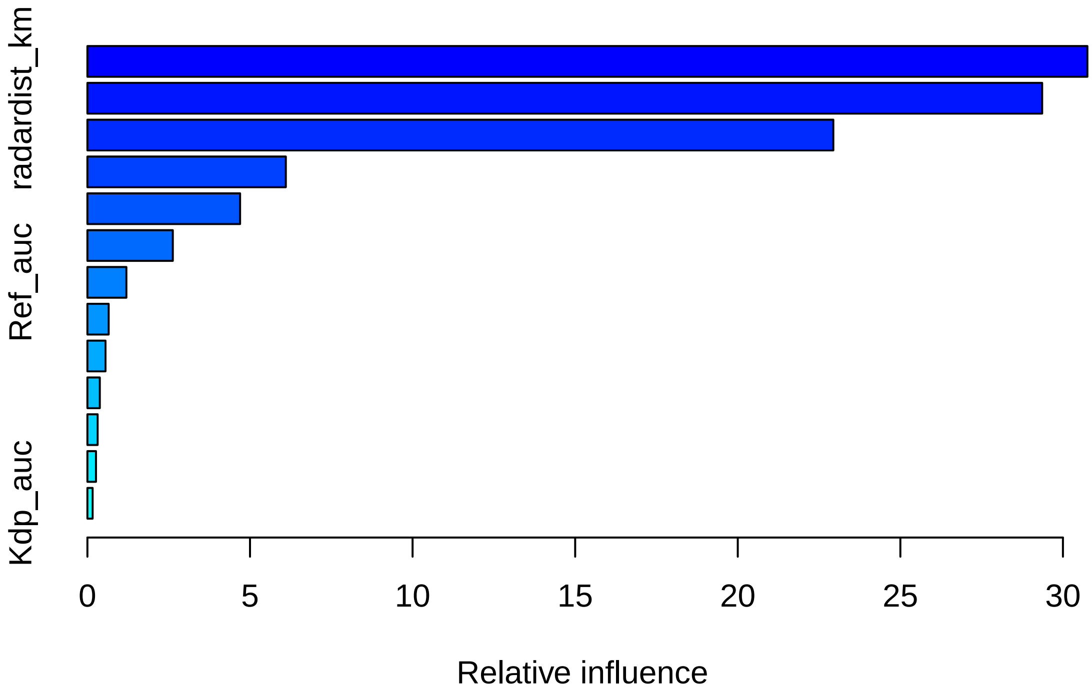

Figure 7: Relative Influence of Features, Best GBM Model

```
##                                                 var    rel.inf
## Ref_5x5_90th_auc                   Ref_5x5_90th_auc 30.7517908
## radardist_km                           radardist_km 29.3610536
## radardist_km_sq_inv             radardist_km_sq_inv 22.9382438
## RefComposite_5x5_90th_auc RefComposite_5x5_90th_auc  6.1014502
## Ref_5x5_50th_auc                   Ref_5x5_50th_auc  4.6950918
## RhoHV_auc                                 RhoHV_auc  2.6260057
## Ref_auc                                     Ref_auc  1.1978888
## RefComposite_5x5_50th_auc RefComposite_5x5_50th_auc  0.6537548
## RefComposite_5x5_10th_auc RefComposite_5x5_10th_auc  0.5563159
## Zdr_auc                                     Zdr_auc  0.3814320
## RefComposite_auc                   RefComposite_auc  0.3140543
## Ref_5x5_10th_auc                   Ref_5x5_10th_auc  0.2629104
## Kdp_auc                                     Kdp_auc  0.1600081
```

Competition Submission, Results

I submitted a total of 7 entries to the Kaggle competition. The best GBM model gave me the best Kaggle score.

Public Leaderboard

Figure 8 is my public leaderboard score just after I submitted my final model. It is a bit worse than my CV score, but still in the neighborhood.

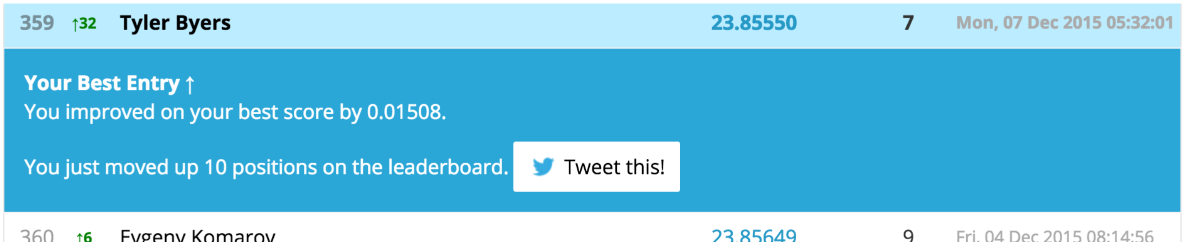

Figure 8: Kaggle Public Leaderboard Standings, Final Model Submitted

Private Leaderboard

The Private, or final, leaderboard was revealed at 5:00 MST on 7 December. I more or less stayed in the same position, but my score dropped considerably (so did everyone else's score).

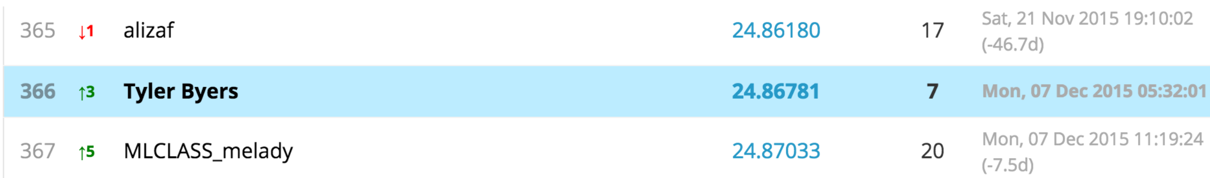

Figure 9: Kaggle Private Leaderboard (final) Results

Conclusions

I learned a lot and had fun doing this competition. Had I started this competition sooner, and given myself more time to work on it over the Thanksgiving holiday, I would have:

- Used more variables – did not use them all.
- Use RhoHV to filter out bad values, per some information I found online.
- Learned more about the other variables in the data.
- Tried a different transformation besides Marshall-Palmer.
- Changed my method of calculating auc or flattening variables. It clearly didn't work as well as I hoped (I'm still in the bottom half of the leaderboard).
- Done more with my GBM grids – perhaps could have gained a little more with more trees or tweaking other results.

However, this was a good project and fun to work on, and I am happy I chose it.

Appendix A – R Script

Below is the entirety of the script I built as I was completing this project. It includes comments that I put as I was going along.

```
##--------------------------------------------
##
## Kaggle Final Project Part #2 of 2
##
## Class: Methods for Data Science at Scale
```

```r
## UW Data Science Certificate Class #3 of 3
##
## Tyler Byers
## Nov 30, 2015
##--------------------------------------------

## This documents my work on the Kaggle Project for the UW Methods for
##  Data Science at Scale course.  This is part #2 of my project.

## This is the only document I am turning in.  You may read my
##  code and look at my notes to see what I have tried.  I tried to document
## my local CV scores as well as my Kaggle scores when submitted.

## Suggest you do NOT run the following code, as it may take hours to run!!

##-----Set working directory-----
setwd('~/UW_DataScience/DataAtScale/kaggle_howmuchididitrain/')

##-----Load Libraries-----
library(dplyr); library(randomForest); library(rpart);
library(caret); library(ggplot2); library(MESS); library(randomForest)
library(gbm)

##-----Load Data Sets --------------
train <- read.csv('./data/train.csv')
test <- read.csv('./data/test.csv')

## Filter out train$Expected
#train <- train %>% filter(Expected < 350)
## Realized later that doing this messes up the scoring, so need to leave all values in.
## Maybe when training we can ignore it.

## This is a mostly a feature engineering project.
##  First thing I'm going to do is just take the mean and
##  median of each radar reading and then model with that.

train_1perhr <- train %>% group_by(Id) %>%
  summarise(n_obs = n(), radardist_km = first(radardist_km),
            Ref_mean = mean(Ref, na.rm = TRUE), #Ref_med = median(Ref, na.rm = TRUE),
            Ref_5x5_10th_mean = mean(Ref_5x5_10th, na.rm = TRUE),
            Ref_5x5_50th_mean = mean(Ref_5x5_50th, na.rm = TRUE),
            Ref_5x5_90th_mean = mean(Ref_5x5_90th, na.rm = TRUE),
            RefComposite_mean = mean(RefComposite, na.rm = TRUE),
            RefComposite_5x5_10th_mean = mean(RefComposite_5x5_10th, na.rm = TRUE),
            RefComposite_5x5_50th_mean = mean(RefComposite_5x5_50th, na.rm = TRUE),
            RefComposite_5x5_90th_mean = mean(RefComposite_5x5_90th, na.rm = TRUE),
            RhoHV_mean = mean(RhoHV, na.rm = TRUE),
            RhoHV_5x5_10th_mean = mean(RhoHV_5x5_10th, na.rm = TRUE),
            RhoHV_5x5_50th_mean = mean(RhoHV_5x5_50th, na.rm = TRUE),
            RhoHV_5x5_90th_mean = mean(RhoHV_5x5_90th, na.rm = TRUE),
            Zdr_mean = mean(Zdr, na.rm = TRUE),
            Zdr_5x5_10th_mean = mean(Zdr_5x5_10th, na.rm = TRUE),
            Zdr_5x5_50th_mean = mean(Zdr_5x5_50th, na.rm = TRUE),
```

```
            Zdr_5x5_90th_mean = mean(Zdr_5x5_90th, na.rm = TRUE),
            Kdp_mean = mean(Kdp, na.rm = TRUE),
            Kdp_5x5_10th_mean = mean(Kdp_5x5_10th, na.rm = TRUE),
            Kdp_5x5_50th_mean = mean(Kdp_5x5_50th, na.rm = TRUE),
            Kdp_5x5_90th_mean = mean(Kdp_5x5_90th, na.rm = TRUE),
            Expected = first(Expected))

## Function to check error
mae <- function(actual, pred) {
  mae <-  mean(abs(actual - pred))
  print(paste('MAE: ', mae))
  mae
}

write_output <- function(id, pred) {
  output <- data.frame(Id = id, Expected = pred)
  curdatetime <- strftime(Sys.time(), '%Y%m%d_%H%M%S')
  write.csv(output, paste0('output/', curdatetime, '.csv'), row.names = FALSE)
  print(paste0('Wrote file: ',  paste0('/output/', curdatetime, '.csv')))
}

## Going to do a baseline test using the mean rainfall for all observations
mean_all <- mean((train_1perhr %>% filter(Expected < 350))$Expected)  # mean of Expected with obs > 350
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)], mean_all)
# MAE: 11.77 -- hmm, doesn't seem right (later -- was with filtered Expected vals)
#  With ugly bad Expeced vals, score is 27.2388, much more in line w/ Kaggle score.
write_output(unique(test$Id), mean_all)
# kaggle score: 27.82422

## What if I take the mean of Expected, filtering out over 150, what then is my score?
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)],
    mean((train_1perhr %>% filter(Expected < 150))$Expected))
## Score: 24.5667

median(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)])
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)], 1.27)
## Score: 23.53 --- would be top of leaderboard, seems too good to be true though
write_output(unique(test$Id), 1.27)
## Kaggle score: 24.1578, so it was too good to be true :)

## Now need to figure out how to format the data so that we know how many minutes
##  each reading is good for.

# Try the Murray-Palmer transformation, using the mean ref (nothing fancy)
#  Still not doing real machine learning, just calculation, so don't need
#  to do CV or anything like that.
mp_pred <- (10^(train_1perhr$Ref_mean/10)/200)^(5/8)
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)],
    mp_pred[!is.na (mp_pred)])
## Local score: 23.361129

# Now do this calculation for the test set
test_1perhr <- test %>% group_by(Id) %>%
```

```
  summarise(n_obs = n(), radardist_km = first(radardist_km),
            Ref_mean = mean(Ref, na.rm = TRUE))
mp_pred_test <- (10^(test_1perhr$Ref_mean/10)/200)^(5/8)
## Fill in the NAs with 9999
mp_pred_test[is.na (mp_pred_test)] <- 9999
write_output(unique(test$Id), mp_pred_test)
## Kaggle score: 24.01167, moved up to 377th place


## Want to do a spline-fit under the curve.
## Can do a spline fit for each and every id
# system.time(Ref_spline <- sapply(unique(train$Id)[1:250], function(id) {
#   if(id %% 1000 == 1) {
#     print(paste(Sys.time(), ': Id = ', id))
#   }
#   #id_df <- train %>% filter(Id == id)
#   id_rows <- which(train$Id == id)
#   if(sum(!(is.na (train$Ref[id_rows]))) > 0) {
#     # do the calculation. Sort of a simple integration
#     sum(spline(x = train$minutes_past[id_rows],
#                y = train$Ref[id_rows], xout = 0:60)$y)/60
#   } else {
#     NA
#   }
# })
# )
# The above is going to take around 26 hours to do. Need a different way.


# Maybe take advantage of the group_by in dplyr
# do auc from MESS package....but cannot have NAs on the ends
valid_ids <- train_1perhr$Id[which(!is.na (train_1perhr$Ref_mean))]
system.time(Ref_spline <- train %>% filter(Id %in% valid_ids) %>%
              group_by(Id) %>%
              summarise(Ref_spline_periodic = sum(spline(x = minutes_past,
                                                         y = Ref, xout = 0:60,
                                                         method = 'periodic')$y)/60))

# This took 177 seconds! Sweet!  (with method = 'periodic')
# However, it led to some horribly unbounded spline fits, leading to Inf
#  results on the Murray Palmar fit.
sfit_pred <-  (10^(Ref_spline$Ref_spline_periodic/10)/200)^(5/8)
sfit_pred[sfit_pred > 200] <- 150
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)],
    sfit_pred)
# Local MAE: 23.4652

# Try with a natural spline fit:
Ref_spline_natural <- train %>% filter(Id %in% valid_ids) %>%
  group_by(Id) %>%
  summarise(Ref_spline_natural = sum(spline(x = minutes_past,
                                            y = Ref, xout = 0:60,
                                            method = 'natural')$y)/60)
```

```
sfit_pred_natural <-  (10^(Ref_spline_natural$Ref_spline_natural/10)/200)^(5/8)
sfit_pred_natural[sfit_pred_natural > 150] <- 150
mae(train_1perhr$Expected[!is.na (train_1perhr$Ref_mean)],
    sfit_pred_natural)
## Local score: 24.727.  Maybe not worth submitting.  That method isn't so great.

## Realized after looking at 4th posting on this page:
##  https://www.kaggle.com/c/how-much-did-it-rain-ii/forums/t/16572/38-missing-data/
## that I need to first convert Ref column to rain rate.  Maybe then I can find
## area under the curve and go from there.

# Remove those dfs to reclaim space
rm(Ref_spline); rm(Ref_spline_natural); rm(mp_pred); rm(mp_pred_test);
rm(sfit_pred_natural); rm(sfit_pred); rm(valid_ids)

# Calculate instantaneous rain rate (mm/hr):
train$Ref_irr <- (10^(train$Ref/10)/200)^(5/8)
## all the NAs should be 0 now, for 0 instant rain rate.
train$Ref_irr[is.na (train$Ref_irr)] <- 0
summary(train$Ref_irr)
# Now do the auc calc, with a 0 anchor just before 0 and after 60
system.time(train_Ref_irr_auc <- train %>% group_by(Id) %>%
              summarise(n_obs = n(),
                        Est_rain_linear = auc(x = c(-0.001, minutes_past, 60.001),
                                              y = c(0, Ref_irr, 0), type = 'linear')/60#,
                        #Est_rain_spline = auc(x = c(-0.001, minutes_past, 60.001),
                        #               y = c(0, Ref_irr, 0), type = 'spline')/60
              )
)
train_Ref_irr_auc$Est_rain_spline[train_Ref_irr_auc$Est_rain_spline < 0] <- 0
## only problem is that for the 1-observation hours we're getting NA,
##  so added anchors on either side of 0 and 60.
scorerows <- !is.na (train_1perhr$Ref_mean)
mae(train_1perhr$Expected[scorerows],
    train_Ref_irr_auc$Est_rain_linear[scorerows])
# Calc MAE: 23.3873.  Slightly lower than my "mean" method, but same area.
mae(train_1perhr$Expected[scorerows],
    train_Ref_irr_auc$Est_rain_spline[scorerows])
# Spline calc MAE: 36.883.  This gets pretty unbounded, so going to use 'linear'
#  from here on out.

# That MAE of 23.3873 is close to that for my "mean" calculation. Not going to
#  submit to Kaggle at this time.
# But, want to do some modeling with all the Ref values and the km from gauge.
#  Maybe can start to get better results with that.

# Want a function with the MP transformation
mp_transform <- function(Ref_vect) {
  (10^(Ref_vect/10)/200)^(5/8)
}

train_transf <- as.data.frame(lapply(train[,5:11], function(x) { mp_transform(x) }))
names(train_transf) <- paste0(names(train_transf), '_irr')
```

```r
summary(train_transf)
train_transf <- as.data.frame(lapply(train_transf, function(x) {
  x[is.na (x)] <- 0
  x
}))
train_transf <- bind_cols(train[c('Id', 'minutes_past', 'radardist_km', 'Ref_irr')],
                          train_transf, train[c('Expected')])
# Now do AUC for each column.
train_allref_auc <- train_transf %>% group_by(Id) %>%
  summarise(n_obs = n(), radardist_km = first(radardist_km),
            Ref_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Ref_irr, 0), type = 'linear')/60,
            Ref_5x5_10th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_10th_irr, 0), type = 'linear')/60,
            Ref_5x5_50th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_50th_irr, 0), type = 'linear')/60,
            Ref_5x5_90th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_90th_irr, 0), type = 'linear')/60,
            RefComposite_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, RefComposite_irr, 0), type = 'linear')/60,
            RefComposite_5x5_10th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_10th_irr, 0),
                                            type = 'linear')/60,
            RefComposite_5x5_50th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_50th_irr, 0),
                                            type = 'linear')/60,
            RefComposite_5x5_90th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_90th_irr, 0),
                                            type = 'linear')/60
  )


# Need to know which Ids are all-NA Ref (don't train or score on these)
# Need to know which Ids have Expected > 250 (too high of readings)
allrefna_train <- train_1perhr$Id[is.na (train_1perhr$Ref_mean)]
highExpected_train <- train_1perhr$Id[train_1perhr$Expected > 250]

# Modeling (remember to filter out Expected above 200 when training), and filter
#  out the Ids with all NA Ref values when training.


run_mods <- function(data, allrefna, highExpected, fol, mod = 'rpart', k = 10) {
  print(paste(Sys.time(), ': Entering run_mods function'))
  folds <- createFolds(data$Expected, k = k)
  i <- 0
  accuracies <- lapply(folds, function(fold) {
    i <<- i + 1
    train <- data[-fold, ]
    print(paste('Fold: ', i))
    #print(paste('nrow(train):', nrow(train)))
    train_filt <- train %>% filter(!(Id %in% allrefna)) %>%
      filter(!(Id %in% highExpected))
    #print(paste('nrow(train_filt):', nrow(train_filt)))
```

```r
    test <- data[fold, ]
    if(mod == 'rpart') {
      cart <- rpart(fol, data = train_filt)
      pred <- predict(cart, newdata = test)

    } else if(mod == 'rf') {
      rf <- randomForest(fol, data = train_filt)
      print(importance(rf))
      pred <- predict(rf, newdata = test)
    } else if(mod == 'gbm') {
      gbm_mod <- gbm(formula = fol, data = train_filt, n.trees = 250,
                     interaction.depth = 3, verbose = TRUE, distribution = 'laplace')
      print(summary(gbm_mod))
      pred <- predict(gbm_mod, newdata = test, n.trees = 250)
    }
    pred_df <- data.frame(Id = test$Id, pred = pred)
    #print(paste('nrow(pred_df)', nrow(pred_df)))

    pred_df$Expected <- test$Expected
    print("Prediction summary:")
    print(summary(pred_df))
    pred_df <- pred_df %>% filter(!(Id %in% allrefna))
    print(summary(pred_df))
    #print(paste('nrow(pred_df) after filter', nrow(pred_df)))
    mae_score <- mae(pred_df$Expected, pred_df$pred)
    print(paste(Sys.time(), ': MAE #',i,':', mae_score))
    mae_score
    #list(gbm_mod, train, train_filt, test, pred_df)
  })
  print(paste('Average MAE with 10-fold cv:',
              sum(unlist(accuracies))/k))
  accuracies
}


train_allref_auc$Expected <- train_1perhr$Expected

fol <- Expected ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc

rparts <-
  run_mods(train_allref_auc, allrefna_train,
           highExpected_train, fol, mod = 'rpart', k = 10)
#[1] "Average MAE with 10-fold cv: 24.6999164195204"

# Try a randomForest
library(randomForest)
#run_mods(train_allref_auc, allrefna_train, highExpected_train, fol, mod = 'rf')
# random forest was taking HOURS to run just a few trees.  So trying something else.

fol_gbm <- fol <- Expected ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
```

```
  Ref_5x5_50th_auc + Ref_5x5_90th_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc
gbm_maes <- run_mods(train_allref_auc, allrefna_train, highExpected_train, fol_gbm,
                     mod = 'gbm', k = 10)
#  "Average MAE with 5-fold cv: 24.8861770452619" with
#   fol <- Expected ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
#  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
#  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
#  RefComposite_5x5_90th_auc
#  Need to inspect closer to see if I'm doing it right.

# Figured it out -- was using "gaussian" distribution.
#  Changed to Laplace distribution and...
#    Average MAE with 5-fold cv: 23.4393403784286" (fol_gbm)
#  "Average MAE with 10-fold cv: 23.440215205844" (fol)
#  "Average MAE with 10-fold cv: 23.4410088231101"

## Decent-looking scores there for GBM, but I'm not ready to submit to Kaggle yet.

# Noticed that it's only predicted expected between 0.9 and 1.6.  Maybe if I take
#  the log of the Expected.  Then I can reverse it later....

# Create a new function with this change.
run_mods_Exp_log <- function(data, allrefna, highExpected, fol, mod = 'rpart', k = 10) {
  print(paste(Sys.time(), ': Entering run_mods function'))
  folds <- createFolds(data$Expected, k = k)
  i <- 0
  # Take log of Expected
  data$Expected <- log(data$Expected + 1)
  accuracies <- lapply(folds, function(fold) {
    i <<- i + 1
    train <- data[-fold, ]
    print(paste('Fold: ', i))
    #print(paste('nrow(train):', nrow(train)))
    train_filt <- train %>% filter(!(Id %in% allrefna)) %>%
      filter(!(Id %in% highExpected))
    #print(paste('nrow(train_filt):', nrow(train_filt)))
    test <- data[fold, ]
    if(mod == 'rpart') {
      cart <- rpart(fol, data = train_filt)
      pred <- predict(cart, newdata = test)
    } else if(mod == 'rf') {
      rf <- randomForest(fol, data = train_filt)
      print(importance(rf))
      pred <- predict(rf, newdata = test)
    } else if(mod == 'gbm') {
      gbm_mod <- gbm(formula = fol, data = train_filt, n.trees = 250,
                     interaction.depth = 3, verbose = TRUE, distribution = 'gaussian')
      print(summary(gbm_mod))
      pred <- predict(gbm_mod, newdata = test, n.trees = 250)
    }
    pred <- exp(pred) - 1 # un-do the log of Expected
    pred_df <- data.frame(Id = test$Id, pred = pred)
```

```
    #print(paste('nrow(pred_df)', nrow(pred_df)))

    pred_df$Expected <- exp(test$Expected) - 1  # undo the log of expected
    print("Prediction summary:")
    print(summary(pred_df))
    pred_df <- pred_df %>% filter(!(Id %in% allrefna))
    print(summary(pred_df))
    #print(paste('nrow(pred_df) after filter', nrow(pred_df)))
    mae_score <- mae(pred_df$Expected, pred_df$pred)
    print(paste(Sys.time(), ': MAE #',i,':', mae_score))
    mae_score
    #list(gbm_mod, train, train_filt, test, pred_df)
  })
  print(paste('Average MAE with',k,'-fold cv:',
              sum(unlist(accuracies))/k))
  accuracies
}

rparts_exp <-
  run_mods_Exp_log(train_allref_auc, allrefna_train,
                   highExpected_train, fol, mod = 'rpart', k = 10)
#[1] "Average MAE with 10 -fold cv: 23.3009274403245"
# This is better than my previous best score! So will have to
#  use this model and submit to Kaggle.

gbm_maes <- run_mods_Exp_log(train_allref_auc, allrefna_train, highExpected_train, fol_gbm,
                             mod = 'gbm', k = 5)
## With fol_gbm formula and laplace distribution:
# "Average MAE with 5 -fold cv: 23.4607803413511"
#  "Average MAE with 5 -fold cv: 23.5136031945773"



## As of turning in part 2 on 11/30/15, I'm still doing some modeling.  I haven't chosen a
##  final model yet, and I'd still like to get in some of the other variables. But I'm
## seeig some promising results with the rpart model and taking the log of the
## Expected values.  Will report on my modeling status for the final part of the project.

# An aside -- what is the best possible score I can expect if all readings under
#  250 are spot on?
pred_best <- train$Expected
median_under250 <- median((train %>% filter(Expected < 250))$Expected)
pred_best <- ifelse(pred_best < 250, pred_best, median_under250)
mae(pred = pred_best[!(allrefna_train)], actual = train$Expected[!(allrefna_train)])
# MAE 17.62678 -- best possible!

# Ok, back on track.  I like what I saw from the rpart.
#  Need a test set formatted just like the training set.


test_transf <- as.data.frame(lapply(test[,4:11], function(x) { mp_transform(x) }))
names(test_transf) <- paste0(names(test_transf), '_irr')
summary(test_transf)
```

```r
test_transf <- as.data.frame(lapply(test_transf, function(x) {
  x[is.na (x)] <- 0
  x
}))
test_transf <- bind_cols(test[c('Id', 'minutes_past', 'radardist_km')],
                         test_transf)
# Now do AUC for each column.
test_allref_auc <- test_transf %>% group_by(Id) %>%
  summarise(n_obs = n(), radardist_km = first(radardist_km),
            Ref_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Ref_irr, 0), type = 'linear')/60,
            Ref_5x5_10th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_10th_irr, 0), type = 'linear')/60,
            Ref_5x5_50th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_50th_irr, 0), type = 'linear')/60,
            Ref_5x5_90th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, Ref_5x5_90th_irr, 0), type = 'linear')/60,
            RefComposite_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                   y = c(0, RefComposite_irr, 0), type = 'linear')/60,
            RefComposite_5x5_10th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_10th_irr, 0),
                                            type = 'linear')/60,
            RefComposite_5x5_50th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_50th_irr, 0),
                                            type = 'linear')/60,
            RefComposite_5x5_90th_auc = auc(x = c(-0.001, minutes_past, 60.001),
                                            y = c(0, RefComposite_5x5_90th_irr, 0),
                                            type = 'linear')/60
  )

allrefna_test <- test_1perhr$Id[is.na (train_1perhr$Ref_mean)]

train_allref_auc$Expected_log <- log(train_allref_auc$Expected + 1)
train_allref_auc_filtered <- train_allref_auc %>% filter(!(Id %in% allrefna_train)) %>%
  filter(!(Id %in% highExpected_train))
fol_expectedlog = Expected_log ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_5x5_10th_auc +
  RefComposite_5x5_50th_auc

cart <- rpart(fol_expectedlog, data = train_allref_auc_filtered)
summary(cart)
pred_cart <- predict(cart, newdata = test_allref_auc)
pred_cart <- exp(pred_cart) - 1
summary(pred_cart)
# Kaggle score: 23.93418 -- moved up a few spots.

# Had a thought -- maybe if we include a km^2 factor it will work even
#  better -- distance-squared is an important thing to know in "intensity" calculations.

fol <- Expected ~ radardist_km + Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc
```

```r
rparts_exp <-
  run_mods_Exp_log(train_allref_auc, allrefna_train,
                   highExpected_train, fol, mod = 'rpart', k = 10)
# with additional terms in fol:
#  [1] "Average MAE with 10 -fold cv: 23.3026793313298"

train_allref_auc$radardist_km_sq <- (train_allref_auc$radardist_km)^2
fol_kmsq <- fol <- Expected ~ radardist_km + radardist_km_sq +
  Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc
rparts_exp_kmsq <-
  run_mods_Exp_log(train_allref_auc, allrefna_train,
                   highExpected_train, fol_kmsq, mod = 'rpart', k = 10)
# [1] "Average MAE with 10 -fold cv: 23.3034193790669"
#  ^ did not help -- score basically same.


#Now going to do the auc calculations for RhoHV, Zdr and Kdp.


train_auc2 <- train %>% group_by(Id) %>%
  summarise(RhoHV_auc = auc(x = c(-0.001, minutes_past, 60.001),
                            y = c(0, RhoHV, 0), type = 'linear')/60.002,
            Zdr_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Zdr, 0), type = 'linear')/60,
            Kdp_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Kdp, 0), type = 'linear')/60)

# add above to the other train auc
train_allref_auc <- train_allref_auc %>% bind_cols(train_auc2)
# forgot to disinclude Id column from train_auc
train_allref_auc <- train_allref_auc[,c(1:14,16:18)]

# Ok now do another CART model:
train_allref_auc$radardist_km_sq_inv <- 1/(train_allref_auc$radardist_km_sq)
fol_thruKdp <- Expected ~ radardist_km + radardist_km_sq_inv +
  Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc + RhoHV_auc + Zdr_auc + Kdp_auc
rparts_thruKdp <-
  run_mods_Exp_log(train_allref_auc, allrefna_train,
                   highExpected_train, fol_thruKdp, mod = 'rpart', k = 10)
# [1] "Average MAE with 10 -fold cv: 23.3018106358255" radardist_km_sq
# basically same score
# [1] "Average MAE with 10 -fold cv: 23.2984152198045"  maybe slight better
#  but not enough to resubmit to kaggle I think


## GBM with a training grid --
gbm_n.trees <- c(100, 250, 500)
```

```
gbm_interaction.depth <- c(1, 2, 3, 4)
gbm_shrinkage <- c(0.001, 0.01, 0.05, 0.1)
gbm_grid <- expand.grid(gbm_n.trees, gbm_interaction.depth, gbm_shrinkage)
gbm_grid
# There are 48 to do.  this will take a long time to train.
names(gbm_grid) <- c('n.trees', 'interaction.depth', 'shrinkage')
gbm_grid$MAE <- NA

# Now need a 10-fold CV function
run_mods_gbm_grid <- function(data, allrefna, highExpected, fol, k = 10,
                              n.trees, interaction.depth, shrinkage) {
  print(paste(Sys.time(), ': Entering run_mods function'))
  folds <- createFolds(data$Expected, k = k)
  i <- 0
  # Take log of Expected
  data$Expected <- log(data$Expected + 1)
  accuracies <- lapply(folds, function(fold) {
    i <<- i + 1
    train <- data[-fold, ]
    print(paste('Fold: ', i))
    #print(paste('nrow(train):', nrow(train)))
    train_filt <- train %>% filter(!(Id %in% allrefna)) %>%
      filter(!(Id %in% highExpected))
    #print(paste('nrow(train_filt):', nrow(train_filt)))
    test <- data[fold, ]
    gbm_mod <- gbm(formula = fol, data = train_filt, n.trees = n.trees,
                   interaction.depth = interaction.depth,
                   shrinkage = shrinkage, verbose = TRUE, distribution = 'gaussian')
    print(summary(gbm_mod))
    pred <- predict(gbm_mod, newdata = test, n.trees = n.trees)
    pred <- exp(pred) - 1 # un-do the log of Expected
    pred_df <- data.frame(Id = test$Id, pred = pred)
    #print(paste('nrow(pred_df)', nrow(pred_df)))

    pred_df$Expected <- exp(test$Expected) - 1  # undo the log of expected
    print("Prediction summary:")
    print(summary(pred_df))
    pred_df <- pred_df %>% filter(!(Id %in% allrefna))
    print(summary(pred_df))
    #print(paste('nrow(pred_df) after filter', nrow(pred_df)))
    mae_score <- mae(pred_df$Expected, pred_df$pred)
    print(paste(Sys.time(), ': MAE #',i,':', mae_score))
    mae_score
    #list(gbm_mod, train, train_filt, test, pred_df)
  })
  print(paste('Average MAE with',k,'-fold cv:',
              sum(unlist(accuracies))/k))
  sum(unlist(accuracies))/k
}

for(i in 1:nrow(gbm_grid)) {
  gbm_grid$MAE[i] <- run_mods_gbm_grid(train_allref_auc, allrefna_train,
                                       highExpected_train, fol_thruKdp,
```

```
                                            interaction.depth = gbm_grid$interaction.depth[i],
                                            shrinkage = gbm_grid$shrinkage[i],
                                            n.trees = gbm_grid$n.trees[i], k = 10)
  print(gbm_grid)
}

# Make the test set look like the train set:

test_allref_auc$radardist_km_sq_inv <- 1/(test_allref_auc$radardist_km)^2
test_auc2 <- test %>% group_by(Id) %>%
  summarise(RhoHV_auc = auc(x = c(-0.001, minutes_past, 60.001),
                            y = c(0, RhoHV, 0), type = 'linear')/60.002,
            Zdr_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Zdr, 0), type = 'linear')/60,
            Kdp_auc = auc(x = c(-0.001, minutes_past, 60.001),
                          y = c(0, Kdp, 0), type = 'linear')/60)

# add above to the other test auc
test_allref_auc <- test_allref_auc %>% bind_cols(test_auc2)
test_allref_auc <- test_allref_auc[,c(1:12,14:16)]

fol_thruKdp_gbm <- fol <- Expected_log ~ radardist_km + radardist_km_sq_inv +
  Ref_auc + Ref_5x5_10th_auc +
  Ref_5x5_50th_auc + Ref_5x5_90th_auc + RefComposite_auc +
  RefComposite_5x5_10th_auc + RefComposite_5x5_50th_auc +
  RefComposite_5x5_90th_auc + RhoHV_auc + Zdr_auc + Kdp_auc
gbm_mod_500_4_0.01 <- gbm(formula = fol_thruKdp_gbm, data = train_allref_auc,
                              n.trees = 500,
                              interaction.depth = 4,
                              shrinkage = 0.01, verbose = TRUE,
                              distribution = 'gaussian')

pred_gbm_500_4_0.01 <- predict(gbm_mod_500_4_0.01, newdata = test_allref_auc,
                               n.trees = 500)
pred_gbm_500_4_0.01 <- exp(pred_gbm_500_4_0.01) - 1
summary(pred_gbm_500_4_0.01)
write_output(unique(test$Id), pred_gbm_500_4_0.01)
# Kaggle score: 23.93359  -- no real improvement in leaderboard

#Another model based on CV:
gbm_mod_500_4_0.05 <- gbm(formula = fol_thruKdp_gbm, data = train_allref_auc,
                          n.trees = 500,
                          interaction.depth = 4,
                          shrinkage = 0.05, verbose = TRUE,
                          distribution = 'gaussian')

pred_gbm_500_4_0.05 <- predict(gbm_mod_500_4_0.05, newdata = test_allref_auc,
                               n.trees = 500)
pred_gbm_500_4_0.05 <- exp(pred_gbm_500_4_0.05) - 1
summary(pred_gbm_500_4_0.05)
# wow, a bunch of predictions were below 0
pred_gbm_500_4_0.05[pred_gbm_500_4_0.05 < 0] <- 0
write_output(unique(test$Id), pred_gbm_500_4_0.05)
```

```r
saveRDS(gbm_mod_500_4_0.05, 'gbm_mod_500_4_0.05.rds')
```

Appendix B – CART Model Summary

```r
summary(mycart)
```

```
## Call:
## rpart(formula = fol_expectedlog, data = train_allref_auc_filtered)
##   n= 723068
##
##           CP nsplit rel error    xerror        xstd
## 1 0.12868559      0 1.0000000 1.0000048 0.002661399
## 2 0.02013490      1 0.8713144 0.8715530 0.002691920
## 3 0.01606577      2 0.8511795 0.8535602 0.002690848
## 4 0.01000000      3 0.8351137 0.8340034 0.002656737
##
## Variable importance
##          Ref_5x5_90th_auc          Ref_5x5_50th_auc
##                        21                        18
##                   Ref_auc RefComposite_5x5_50th_auc
##                        17                        16
##          Ref_5x5_10th_auc RefComposite_5x5_10th_auc
##                        13                        12
##              radardist_km
##                         2
##
## Node number 1: 723068 observations,    complexity param=0.1286856
##   mean=1.033671, MSE=0.8550311
##   left son=2 (572688 obs) right son=3 (150380 obs)
##   Primary splits:
##       Ref_5x5_90th_auc          < 5.781625  to the left,  improve=0.1286856, (0 missing)
##       Ref_auc                   < 3.042624  to the left,  improve=0.1223567, (0 missing)
##       Ref_5x5_50th_auc          < 2.830758  to the left,  improve=0.1220954, (0 missing)
##       RefComposite_5x5_50th_auc < 4.363153  to the left,  improve=0.1101547, (0 missing)
##       Ref_5x5_10th_auc          < 1.350574  to the left,  improve=0.1009547, (0 missing)
##   Surrogate splits:
##       Ref_auc                   < 3.096105  to the left,  agree=0.952, adj=0.771, (0 split)
##       Ref_5x5_50th_auc          < 2.810515  to the left,  agree=0.951, adj=0.764, (0 split)
##       RefComposite_5x5_50th_auc < 3.887572  to the left,  agree=0.941, adj=0.717, (0 split)
##       Ref_5x5_10th_auc          < 1.592973  to the left,  agree=0.913, adj=0.580, (0 split)
##       RefComposite_5x5_10th_auc < 2.443069  to the left,  agree=0.906, adj=0.550, (0 split)
##
## Node number 2: 572688 observations,    complexity param=0.01606577
##   mean=0.8636938, MSE=0.7071741
##   left son=4 (228828 obs) right son=5 (343860 obs)
##   Primary splits:
##       radardist_km     < 9.5       to the left,  improve=0.02452552, (0 missing)
##       Ref_5x5_90th_auc < 2.205512  to the left,  improve=0.02004661, (0 missing)
##       Ref_auc          < 1.15624   to the left,  improve=0.01649792, (0 missing)
##       Ref_5x5_50th_auc < 0.9853278 to the left,  improve=0.01623368, (0 missing)
##       Ref_5x5_10th_auc < 0.4311442 to the left,  improve=0.01151446, (0 missing)
```

```
##   Surrogate splits:
##       RefComposite_5x5_10th_auc < 1.178118  to the right, agree=0.644, adj=0.110, (0 split)
##       RefComposite_5x5_50th_auc < 2.072298  to the right, agree=0.643, adj=0.108, (0 split)
##       Ref_5x5_50th_auc          < 1.700849  to the right, agree=0.615, adj=0.037, (0 split)
##       Ref_5x5_10th_auc          < 0.9565667 to the right, agree=0.615, adj=0.036, (0 split)
##       Ref_auc                   < 1.897688  to the right, agree=0.612, adj=0.029, (0 split)
##
## Node number 3: 150380 observations,    complexity param=0.0201349
##   mean=1.680993, MSE=0.8890559
##   left son=6 (113266 obs) right son=7 (37114 obs)
##   Primary splits:
##       Ref_5x5_50th_auc          < 7.175581  to the left,  improve=0.09310894, (0 missing)
##       Ref_5x5_90th_auc          < 14.04444  to the left,  improve=0.09169302, (0 missing)
##       Ref_auc                   < 7.267963  to the left,  improve=0.08679795, (0 missing)
##       RefComposite_5x5_50th_auc < 13.06538  to the left,  improve=0.08057645, (0 missing)
##       Ref_5x5_10th_auc          < 4.50983   to the left,  improve=0.06865947, (0 missing)
##   Surrogate splits:
##       Ref_auc                   < 8.375708  to the left,  agree=0.932, adj=0.723, (0 split)
##       RefComposite_5x5_50th_auc < 10.97799  to the left,  agree=0.916, adj=0.660, (0 split)
##       Ref_5x5_10th_auc          < 3.962463  to the left,  agree=0.914, adj=0.650, (0 split)
##       Ref_5x5_90th_auc          < 17.45347  to the left,  agree=0.903, adj=0.606, (0 split)
##       RefComposite_5x5_10th_auc < 6.640272  to the left,  agree=0.875, adj=0.494, (0 split)
##
## Node number 4: 228828 observations
##   mean=0.7022547, MSE=0.5142862
##
## Node number 5: 343860 observations
##   mean=0.9711264, MSE=0.8066493
##
## Node number 6: 113266 observations
##   mean=1.516298, MSE=0.7548876
##
## Node number 7: 37114 observations
##   mean=2.183614, MSE=0.9631087
```