FinalPaper

Alex Federspiel

4/16/2022

Summary

The Acea Group is an Italian multiutility operator with a water sector that comprises of aquifers, lakes, rivers, and springs. They want to be able to use rainfall data from surrounding cities to accurately predict the water levels of these bodies of water. The body of water in focus here is Lake Bilancino.

The main difficulty for predicting the levels comes from the fact that rainfall doesn't reach the lake immediately. It needs to seeps into the ground and travel its way down which takes an unknown amount of time. Therefore, many additional rainfall variables were added into the model to account for the different possibilities of delay. A LASSO model was used for feature selection to cut these in half. The resulting predictors were then used as the basis to predict Lake Billancino's water level.

Five different models were used to run the dataset. A LASSO model was selected from the linear family, a regression tree and random forest were selected from the tree family, KNN was selected from the nonlinear family, and finally gradient boosting was used due to its predictive power.

The results showed that gradient boosting performed the best in terms of predicting the final year of water level data. However, LASSO almost seemed able to predict the peaks and valleys the best but with a delay.

Variables

The dataset I'm looking at focuses on only one of the water sources: Lake Bilancino. The dataset spans from June 6th, 2002 to June 30th, 2020 and contains data on the flow rate of the lake, the measure of the lake's water level, the temperature, and the rainfall of five cities surrounding Lake Bilancino—San Piero, Mangona, Sant'Agata, Cavallina, and Le Croci. Unfortunately, most of the data before 2004 is missing.

An early obstacle was the scope of the variables. The data was presented in such a way that the variables didn't show any correlations, even when logically there should have been one: for example, rainfall and lake level showed no correlation. The crux of the issue was that many of these variables were represented over time with the "Date" variable. In order to fix this, I had to change the scope of most of the variables. The main change was turning the outcome variable, "Lake Level," into the "Change in Lake Level." This allowed me to model the data in a more efficient and comprehensive manner as I could begin showing correlations between variables. (See Appendix 1)

The second hurdle was the hydrology of rainfall. The amount of rain recorded doesn't all reach the lake and it doesn't get there immediately. Instead, the rain seeps into the ground in a process known as infiltration. Some of the factors that can affect infiltration are the type of ground, the saturation of the ground, and even the slope of the land. This infiltrated water, which is called groundwater, moves slowly, and can be lost due to evaporation or evapotranspiration as the water is utilized by the local plantlife. Finally, the remaining groundwater makes it to the waterbody. It's unknown how much rainfall is lost along the way and how long it takes the rain for each city to reach the lake.

To try and account for this I created rainfall delay predictors. This took the rainfall recorded for a day and delayed it by a set number of days. This essentially pushed all the rainfall data down in the dataset by how many days it was delayed. If rainfall was delayed for 3 days, the dataset would have rainfall variables for no delay, a delay of 1 day, a delay of 2 days, and a delay of 3 days. I delayed the rainfall by 14 days which brought the number of variables up to 80. (See Appendix 2)

Finally, in order to attempt to incorporate season change into the model, I created a numerical predictor for the month.

Feature Selection

I first ran the dataset through a LASSO model in order to reduce the number of variables. Not all the rainfall delay variable are important and having LASSO reduce many of their coefficients to zero was a good starting point of indicating which weren't necessary. The LASSO model set the coefficients of almost half the rainfall delays to zero which helped reduce the dimensions of the dataset. (See Appendix 3)

The first main dataset was created using the 41 variables recommended by the LASSO model.

Model Selection

The models I chose to look at were LASSO from the linear family, regression trees, random forests, and GBM from the tree family, and KNN from the nonlinear family. Prediction was definitely a priority for this model but I also included LASSO and regression trees due to their interpretability. Being able to see which delays have the largest coefficients can also be beneficial for constructing future models.

While running these models the first time, I also looked at the top 20 most important delay variables for each model. In an Excel file, I quickly took their averages across all the models and then looked at which delay variables had the highest average. This led me to my second dataset which has only 24 variables. My hope is that these would be the most important delay variables of them all. (See Appendix 4)

Finally, the final year of data was removed from the dataset and set aside as the test set.

Models

Looking at just RMSE, the models performed fairly well. The main takeaway was that the dataset 2 (with 24 variables) didn't appear to perform any better than the first dataset. This could be due to the fact that there are less variables in the dataset in which case and adjusted R^2 value would be more appropriate.

The LASSO model performed better than anticipated and gave RMSE values of 0.3 and 0.31 for each dataset respectively (See Appendix 5). The regression tree model performed worse with 0.36 and 0.35 (See Appendix 6). Random forest was second best with 0.26 for both (See Appendix 7). KNN got 0.34 and 0.32 (See Appendix 8). Finally, gradient boosting performed the best with 0.23 and 0.25 (See Appendix 9).

Outcomes

While the RMSE values seemed decent enough, the true heart of the problem was being able to predict the water level of Lake Bilancino. Using the final year of the dataset which I had set aside, I had each model try and predict what it though the weekly change in lake level would be. Then I created a function that essentially took the weekly change values and converted them back into daily lake level values. (See Appendix 10)

Finally, the predicted lake levels from each model were plotted along with the true lake level values (See Appendix 11). The two main plots of note were the LASSO plot (11A) and the GBM plot (11E).

The LASSO model was interesting because of how well it almost succeeds. Linear models are very interpretable and having a good predictor model that's easy to interpret is a huge advantage. While the model appears to have random fluctuations on more of the straight portions it almost appears to interpret the peaks and valleys with good precision. That is, if you ignore the fact that it predict them happening weeks after the actual ones actually happen. As on now, the model doesn't do a good job of prediction, but with some additional tuning I think this could potentially flourish over the other models.

The objectively best performing model was the gradient boosting model. While this model did a better job predicting a lot of the graph, it didn't do so well in the middle. All the models seemed to have trouble with this quick fluctuating water level actually, and GBM seemed to handle it the best.

Conclusions

This leads to some final questions. The first being, what is the allowance of error here? Looking at the GBM graph, at the last peak in the middle we see that both models are off. For simplicity, say the predicted lake level is 30 meters. The 24 variable graph is off by 1 meter while the 41 variable model is off by 0.5 meters. Are either of these acceptable? Both look big on the graph but maybe that much of a change is negligible. If this is the case, then I'd say the GBM model predicts the water levels well. However, if more accuracy is needed, then no, GBM doesn't predict well.

The second question I have is if under-predicting is more beneficial? Suppose the actual lake level is 30 meters and the model predicts 28. THe Acea Group could see this, potentially impose some water restrictions on the surrounding cities in anticipation only to find that the water levels are actually higher than expected and they can remove the restrictions. But with the reverse, they don't expect water shortage, only to find last minute that there is one. In this case, the models could be changed to favor under predicting values more than over predicting. Maybe add high penalties if the models over predict.

Finally, a better model could definitely be constructed. None of the models seem to be able to predict sudden changes very well. Forecasting and time series techniques should probably be implemented in order to fix the issues occuring.

Appendix

1. Changing Lake Level to Weekly Lake Level

```r
## let t be the day we're looking at. then t-7 is a week ago.
## change in lake level over a week will be (LL at t) - (LL at t-7)
lake_change <- function(t) {
  difference <- signif(LB$Lake_Level[t] - LB$Lake_Level[t-7], 4)
  return(difference)
}

lake_change_vector <- c()

for (j in 1:nrow(LB)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_change_vector <- c(lake_change_vector, lake_change(j))
  }

  else{
    lake_change_vector <- c(lake_change_vector, "NA")
  }
}
```

```
LB <- LB %>% mutate(Weekly_Lake_Change = as.numeric(lake_change_vector))
```

2. Rainfall Delay of 14

```
time_travel = 14
LB_14 <- LB %>% select(Date, Month, Avg_Fort_Temp, Weekly_Lake_Change, Flow_Rate)
LB_14 <- LB_14 %>% mutate(SP_0 = LB_restricted$Rainfall_S_Piero,
                                   M_0 = LB_restricted$Rainfall_Mangona,
                                   SA_0 = LB_restricted$Rainfall_S_Agata,
                                   C_0 = LB_restricted$Rainfall_Cavallina,
                                   LC_0 = LB_restricted$Rainfall_Le_Croci)
for (t in 1:time_travel) {
  for (j in 6:10) {
  transfer_vector <- c()
    for (i in 1:nrow(LB_14)) {
      if(i - t > 0) {
        transfer_vector <- c(transfer_vector, LB_14[[i-t, j]])
      }
      else {
        transfer_vector <- c(transfer_vector, "NA")
      }
    }
    if(j == 6){name <- paste("SP", t, sep = "_")}
    else if(j == 7){name <- paste("M", t, sep = "_")}
    else if(j == 8){name <- paste("SA", t, sep = "_")}
    else if(j == 9){name <- paste("C", t, sep = "_")}
    else{name <- paste("LC", t, sep = "_")}
  LB_14[[name]] <- with(LB_14, as.numeric(transfer_vector))
  }
}
if(time_travel > 15) {
  LB_14 <- LB_14[(time_travel + 1):nrow(LB_14),]
} else {
  LB_14 <- LB_14[15:nrow(LB_14),]
}
```

3. LASSO Feature Selection

```
l4_lasso <- cv.glmnet(x_14, y_14, alpha = 1)
lambda_14 <- l4_lasso$lambda.min

## example plot
## plot(l4_lasso)
```

```
l4_best_lasso <- glmnet(x_14, y_14, alpha = 1, lambda = lambda_14)
coef(l4_best_lasso)
```

```
## 79 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## (Intercept)    3.803617e-01
## Month         -6.617616e-03
## Avg_Fort_Temp -8.960064e-03
## Flow_Rate     -1.743014e-02
## SP_0           .
## M_0            .
## SA_0           .
## C_0            4.850325e-04
## LC_0           2.293551e-04
## SP_1          -2.874750e-04
## M_1            5.452246e-03
## SA_1           .
## C_1            3.078240e-03
## LC_1           .
## SP_2           .
## M_2            6.356689e-03
## SA_2           .
## C_2            4.287512e-03
## LC_2           .
## SP_3           .
## M_3            7.389682e-03
## SA_3           .
## C_3            3.716266e-03
## LC_3           .
## SP_4           3.778257e-03
## M_4            6.898764e-03
## SA_4           .
## C_4            4.322462e-03
## LC_4          -1.252232e-03
## SP_5           2.098209e-03
## M_5            6.443853e-03
## SA_5           .
## C_5            2.239708e-03
## LC_5           .
## SP_6           7.156642e-05
## M_6            7.443749e-03
## SA_6           .
## C_6            2.713834e-03
## LC_6           .
## SP_7           2.772732e-03
## M_7            6.749423e-03
## SA_7           1.353067e-03
## C_7            .
## LC_7           .
## SP_8           .
## M_8            1.998647e-03
## SA_8           .
## C_8            .
## LC_8           .
## SP_9           .
## M_9            4.616105e-04
## SA_9           .
## C_9            .
## LC_9            .
## SP_10           .
## M_10            .
## SA_10           .
## C_10           -3.515779e-04
## LC_10           .
## SP_11           .
## M_11            .
## SA_11           .
## C_11           -2.325362e-04
## LC_11           .
## SP_12          -5.055482e-04
## M_12            .
## SA_12           .
## C_12            .
## LC_12           .
## SP_13           .
## M_13            .
## SA_13           .
## C_13           -3.490074e-04
## LC_13           .
## SP_14           .
## M_14           -8.886220e-04
## SA_14           .
## C_14            .
## LC_14           .
```

4. Two Different Datasets

```
LB_final <- LB_14 %>% select(-c(SA_0, SA_1, SA_2, SA_3, SA_4, SA_5, SA_6, SA_8,
                               SA_9, SA_10, SA_11, SA_12, SA_13, SA_14, SP_1,
                               SP_2, SP_4, SP_5, SP_9, SP_12, SP_13, LC_1, LC_2,
                               LC_4, LC_8, LC_9, LC_10, LC_11, LC_12, LC_13,
                               LC_14, C_7, C_8, C_9, C_10, C_14, M_10, M_12, M_13))
LB_final_2 <- LB_final %>% select(-c(SP_0, SP_8, SP_10, SP_11, SP_14, M_0, M_1,
                                     M_8, M_9, M_11, M_14, C_0, C_1, C_11, C_12,
                                     C_13, LC_0))


LB_final_year <- LB_final %>% filter((year(Date)>=2019 & month(Date)>06)|(year(Date)==2020))
LB_final <- LB_final %>% filter((year(Date)>=2019 & month(Date)<=06)|(year(Date)<=2018))

LB_final_year_2 <- LB_final_2 %>% filter((year(Date)>=2019 & month(Date)>06)|(year(Date)==2020))
LB_final_2 <- LB_final_2 %>% filter((year(Date)>=2019 & month(Date)<=06)|(year(Date)<=2018))
```

5. LASSO model

```
set.seed(42)

lassoGrid <- expand.grid(lambda = c(0, 0.01, 0.05, 0.1),
                           fraction = seq(0.01, 1, length = 20))

final_LASSO <- train(x = x, y = y, method = "enet",
                           tuneGrid = lassoGrid, trControl = ctrl)
final_LASSO_2 <- train(x = x_2, y = y_2, method = "enet",
                           tuneGrid = lassoGrid, trControl = ctrl)

final_LASSO$results[final_LASSO$results$RMSE == min(final_LASSO$results$RMSE),]
```

```
##    lambda fraction      RMSE  Rsquared       MAE     RMSESD RsquaredSD
## 60   0.05        1 0.3009173 0.5562291 0.2004546 0.02987687 0.06702274
##         MAESD
## 60 0.01208703
```

```
final_LASSO_2$results[final_LASSO_2$results$RMSE == min(final_LASSO_2$results$RMSE),]
```

```
##    lambda fraction      RMSE  Rsquared       MAE     RMSESD RsquaredSD
## 60   0.05        1 0.3053461 0.5254036 0.2015389 0.04200181 0.02799507
##         MAESD
## 60 0.01409045
```

```
## Analyses
#ggplot(final_LASSO)
#ggplot(final_LASSO_2)

#plot(varImp(final_LASSO), 20)
#plot(varImp(final_LASSO_2), 20)
```

6. Regreeion Tree Model

```
set.seed(42)

final_tree <- train(x = x, y = y, method = "rpart",
                        tuneLength = 15, trControl = ctrl)
final_tree_2 <- train(x = x_2, y = y_2, method = "rpart",
                        tuneLength = 15, trControl = ctrl)

final_tree$results[final_tree$results$RMSE == min(final_tree$results$RMSE),]
```

```
##            cp      RMSE Rsquared       MAE     RMSESD RsquaredSD      MAESD
## 1 0.009708464 0.3623308 0.373389 0.1933407 0.04347559 0.08911661 0.01364725
```

```
final_tree_2$results[final_tree_2$results$RMSE == min(final_tree_2$results$RMSE),]
```

```
##            cp      RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
## 1 0.009527520 0.3452985 0.4043538 0.1856353 0.0456375  0.0682092 0.0139509
## 2 0.009530212 0.3452985 0.4043538 0.1856353 0.0456375  0.0682092 0.0139509
```

```r
##Analyses
#ggplot(final_tree)
#ggplot(final_tree_2)

#plot(varImp(final_tree), 20)
#plot(varImp(final_tree_2), 20)
```

7. Random Forest model

```r
set.seed(42)

mtryGrid <- data.frame(mtry = floor(seq(1, ncol(x), length = 20)))
mtryGrid_2 <- data.frame(mtry = floor(seq(1, ncol(x_2), length = 20)))

final_rf <- train(x = x, y = y, method = "rf", tuneGrid = mtryGrid,
                  ntree = 100, importance = TRUE, trControl = ctrl)
final_rf_2 <- train(x = x_2, y = y_2, method = "rf", tuneGrid = mtryGrid_2,
                  ntree = 100, importance = TRUE, trControl = ctrl)

final_rf$results[final_rf$results$RMSE == min(final_rf$results$RMSE),]
```

```
##    mtry      RMSE  Rsquared      MAE    RMSESD RsquaredSD       MAESD
## 20   39 0.2602171 0.6792385 0.128203 0.0365854 0.04042242 0.008809783
```

```r
final_rf_2$results[final_rf_2$results$RMSE == min(final_rf_2$results$RMSE),]
```

```
##    mtry      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
## 19   20 0.2603387 0.6591594 0.1317867 0.04776262  0.0526787 0.01047264
```

```r
## Analyses
#ggplot(final_rf)
#ggplot(final_rf_2)

#plot(varImp(final_rf), 20)
#plot(varImp(final_rf_2), 20)
```

8. KNN model

```r
set.seed(42)

final_KNN <- train(x = x, y = y, method = "knn", trControl = ctrl,
                   tuneLength = 10)
final_KNN_2 <- train(x = x_2, y = y_2, method = "knn", trControl = ctrl,
                   tuneLength = 10)

final_KNN$results[final_KNN$results$RMSE == min(final_KNN$results$RMSE),]
```

```
##   k      RMSE  Rsquared       MAE     RMSESD RsquaredSD     MAESD
## 3 9 0.3407325 0.4768089 0.1901311 0.05312237 0.05955177 0.0131055
```

```r
final_KNN_2$results[final_KNN_2$results$RMSE == min(final_KNN_2$results$RMSE),]
```

```
##   k      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
## 1 5 0.3197917 0.4894365 0.1705645 0.04198845 0.07493683 0.01319062
```

```r
## Analyses
#ggplot(final_KNN)
#ggplot(final_KNN_2)

#plot(varImp(final_KNN), 20)
#plot(varImp(final_KNN_2), 20)
```

9.Gradient Boosted model

```r
set.seed(42)

gbmGrid <- expand.grid(interaction.depth = seq(1, 7, by = 2),
                       n.trees = seq(100, 1000, by = 50),
                       n.minobsinnode = 10,
                       shrinkage = c(0.01, 0.1))

final_gbm <- train(x = x, y = y, method = "gbm", tuneGrid = gbmGrid,
                   trControl = ctrl, verbose = FALSE)
final_gbm_2 <- train(x = x_2, y = y_2, method = "gbm", tuneGrid = gbmGrid,
                   trControl = ctrl, verbose = FALSE)

final_gbm$results[final_gbm$results$RMSE == min(final_gbm$results$RMSE),]
```

```
##     shrinkage interaction.depth n.minobsinnode n.trees      RMSE  Rsquared
## 152       0.1                 7             10    1000 0.2317474 0.7315513
##           MAE    RMSESD RsquaredSD       MAESD
## 152 0.1313826 0.0212875 0.05018727 0.007547915
```

```r
final_gbm_2$results[final_gbm_2$results$RMSE == min(final_gbm_2$results$RMSE),]
```

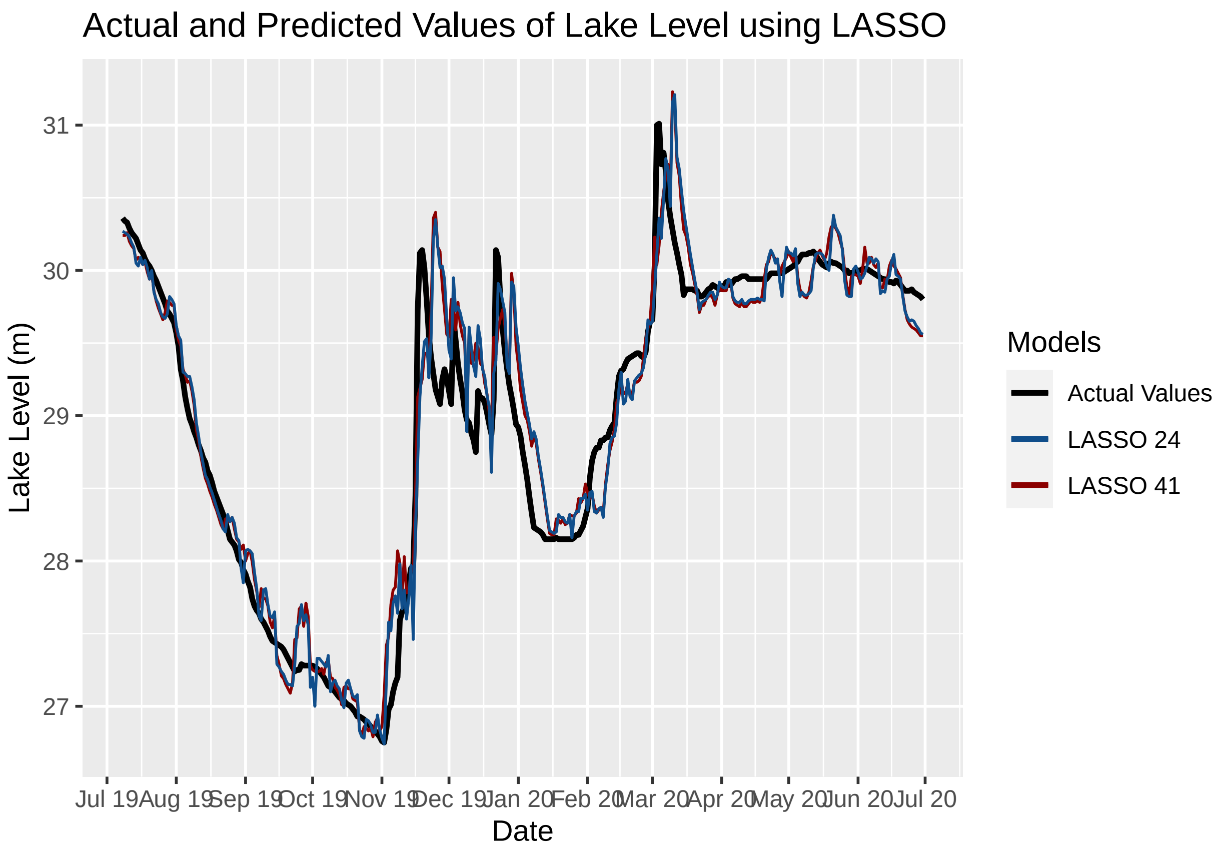
```
##     shrinkage interaction.depth n.minobsinnode n.trees      RMSE  Rsquared
## 144       0.1                 7             10     600 0.2460919 0.6886847
##           MAE     RMSESD RsquaredSD       MAESD
## 144 0.1365576 0.02669094 0.06163628 0.008451619
```

```r
## Analyses
#ggplot(final_gbm)
#ggplot(final_gbm_2)

#plot(varImp(final_gbm), 20)
#plot(varImp(final_gbm_2), 20)
```

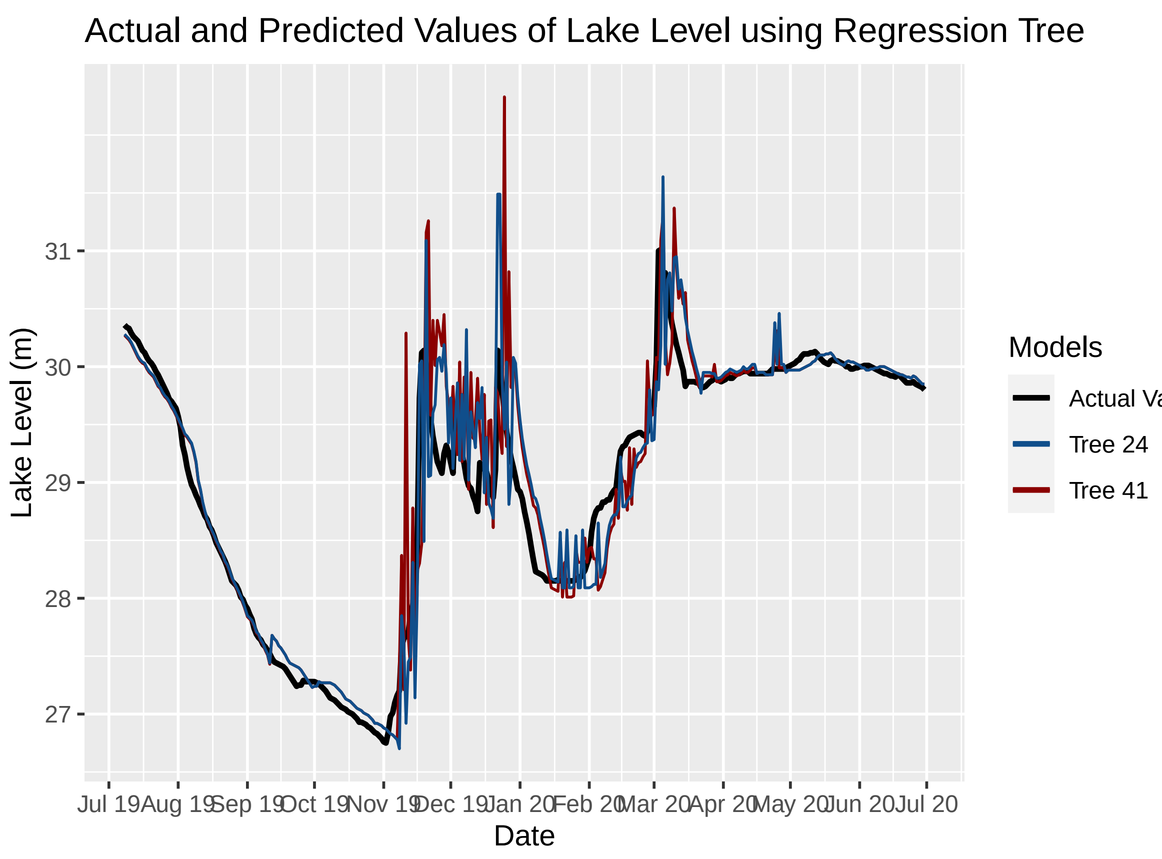
10. Converting Weekly Lake Change Back to Lake Level

```r
lasso_reverse <- function(t) {
  lasso_undifference <- signif(LB_lasso_pred$Predicted_Change[t] +
                               LB_lasso_pred$Lake_Level[t-7], 4)
  return(lasso_undifference)
}
tree_reverse <- function(t) {
  tree_undifference <- signif(LB_tree_pred$Predicted_Change[t] +
                              LB_tree_pred$Lake_Level[t-7], 4)
  return(tree_undifference)
}
rf_reverse <- function(t) {
  rf_undifference <- signif(LB_rf_pred$Predicted_Change[t] +
                            LB_rf_pred$Lake_Level[t-7], 4)
  return(rf_undifference)
}
knn_reverse <- function(t) {
  knn_undifference <- signif(LB_knn_pred$Predicted_Change[t] +
                             LB_knn_pred$Lake_Level[t-7], 4)
  return(knn_undifference)
}
gbm_reverse <- function(t) {
  gbm_undifference <- signif(LB_gbm_pred$Predicted_Change[t] +
                             LB_gbm_pred$Lake_Level[t-7], 4)
  return(gbm_undifference)
}

lake_lasso_vector <- c()
lake_tree_vector <- c()
lake_rf_vector <- c()
lake_knn_vector <- c()
lake_gbm_vector <- c()

for (j in 1:nrow(LB_lasso_pred)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_lasso_vector <- c(lake_lasso_vector, lasso_reverse(j))
  }
  else{
    lake_lasso_vector <- c(lake_lasso_vector, "NA")
  }
}
for (j in 1:nrow(LB_tree_pred)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_tree_vector <- c(lake_tree_vector, tree_reverse(j))
  }
  else{
    lake_tree_vector <- c(lake_tree_vector, "NA")
  }
}
for (j in 1:nrow(LB_rf_pred)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_rf_vector <- c(lake_rf_vector, rf_reverse(j))
  }
  else{
    lake_rf_vector <- c(lake_rf_vector, "NA")
  }
}
for (j in 1:nrow(LB_knn_pred)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_knn_vector <- c(lake_knn_vector, knn_reverse(j))
  }
  else{
    lake_knn_vector <- c(lake_knn_vector, "NA")
  }
}
for (j in 1:nrow(LB_gbm_pred)) {
  ## stops j from being negative which just calls the entire column minus the row
  if (j - 7 > 0) {
    lake_gbm_vector <- c(lake_gbm_vector, gbm_reverse(j))
  }
  else{
    lake_gbm_vector <- c(lake_gbm_vector, "NA")
  }
}

LB_lasso_pred <- LB_lasso_pred %>% mutate(lasso_Lake_Level = as.numeric(lake_lasso_vector))
LB_tree_pred <- LB_tree_pred %>% mutate(tree_Lake_Level = as.numeric(lake_tree_vector))
LB_rf_pred <- LB_rf_pred %>% mutate(rf_Lake_Level = as.numeric(lake_rf_vector))
LB_knn_pred <- LB_knn_pred %>% mutate(knn_Lake_Level = as.numeric(lake_knn_vector))
LB_gbm_pred <- LB_gbm_pred %>% mutate(gbm_Lake_Level = as.numeric(lake_gbm_vector))
```

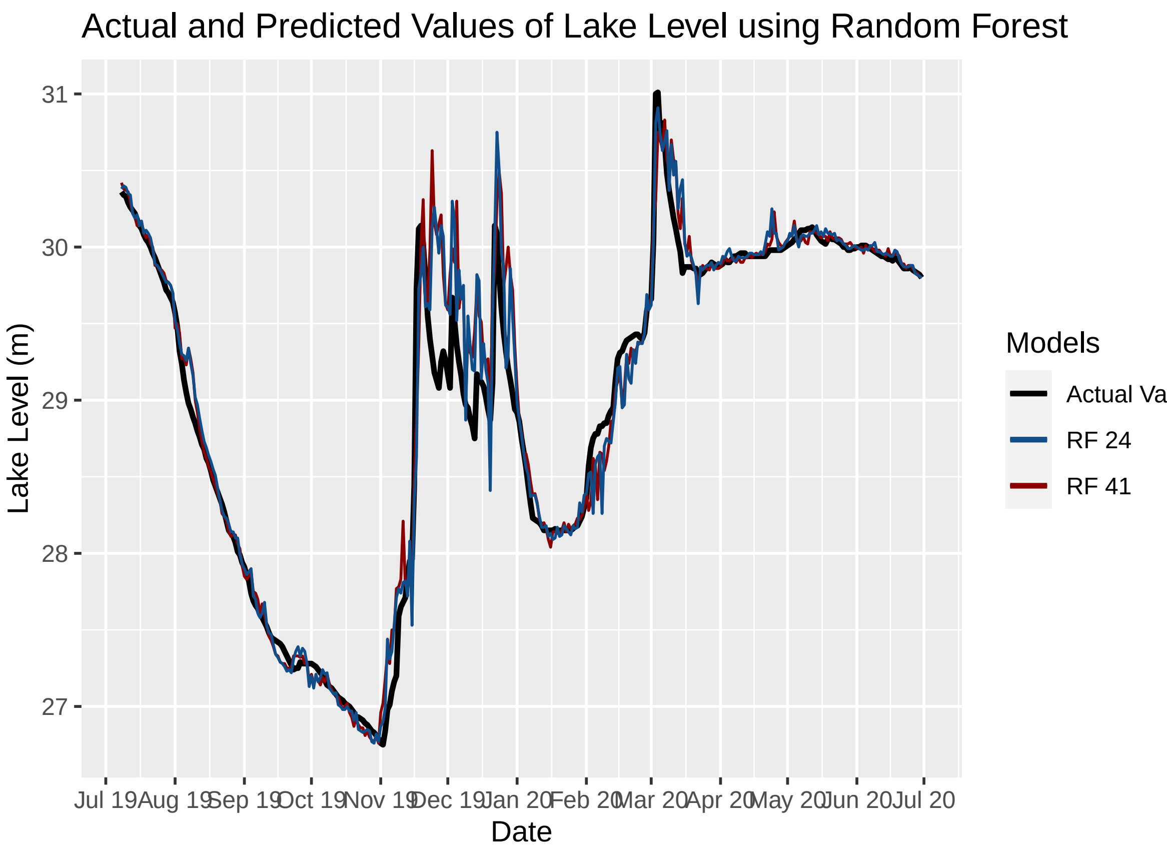
11. Predictions of Models



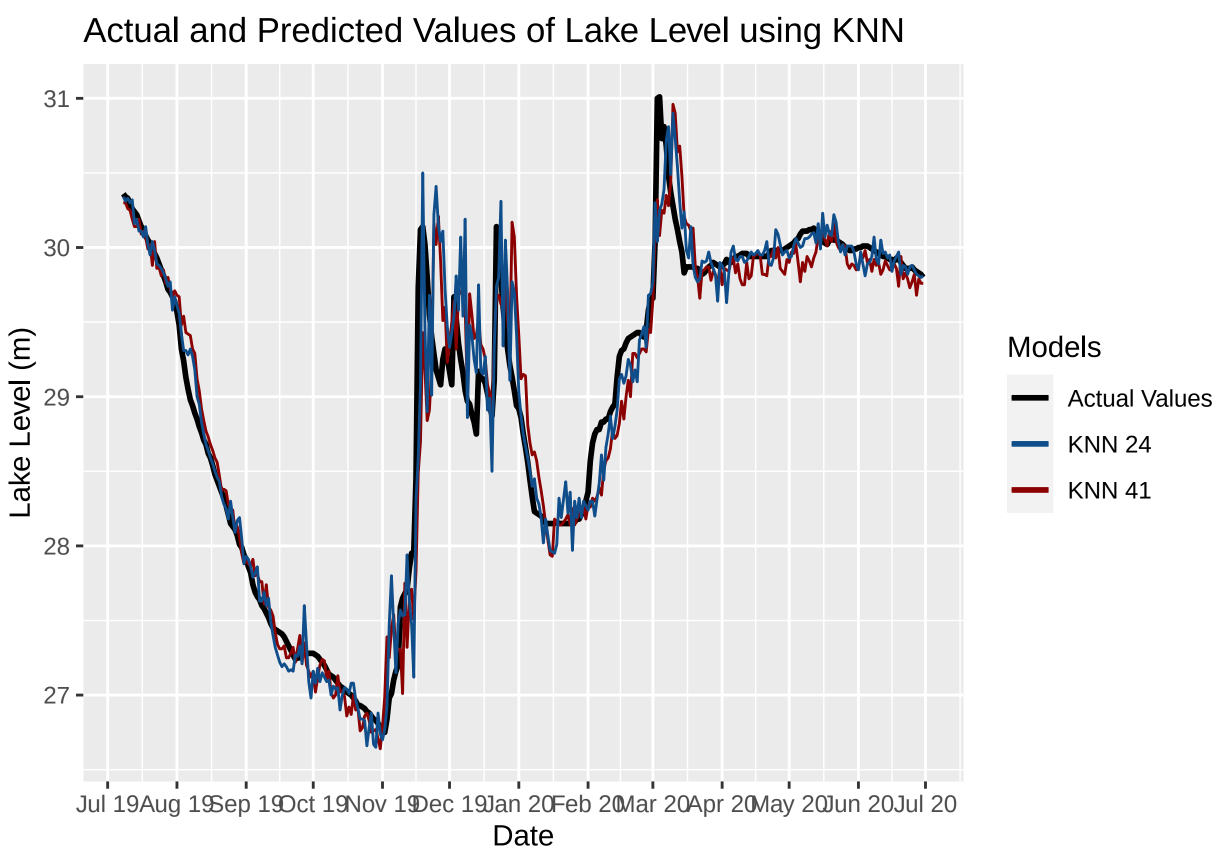
A. LASSO



B. Regression Tree

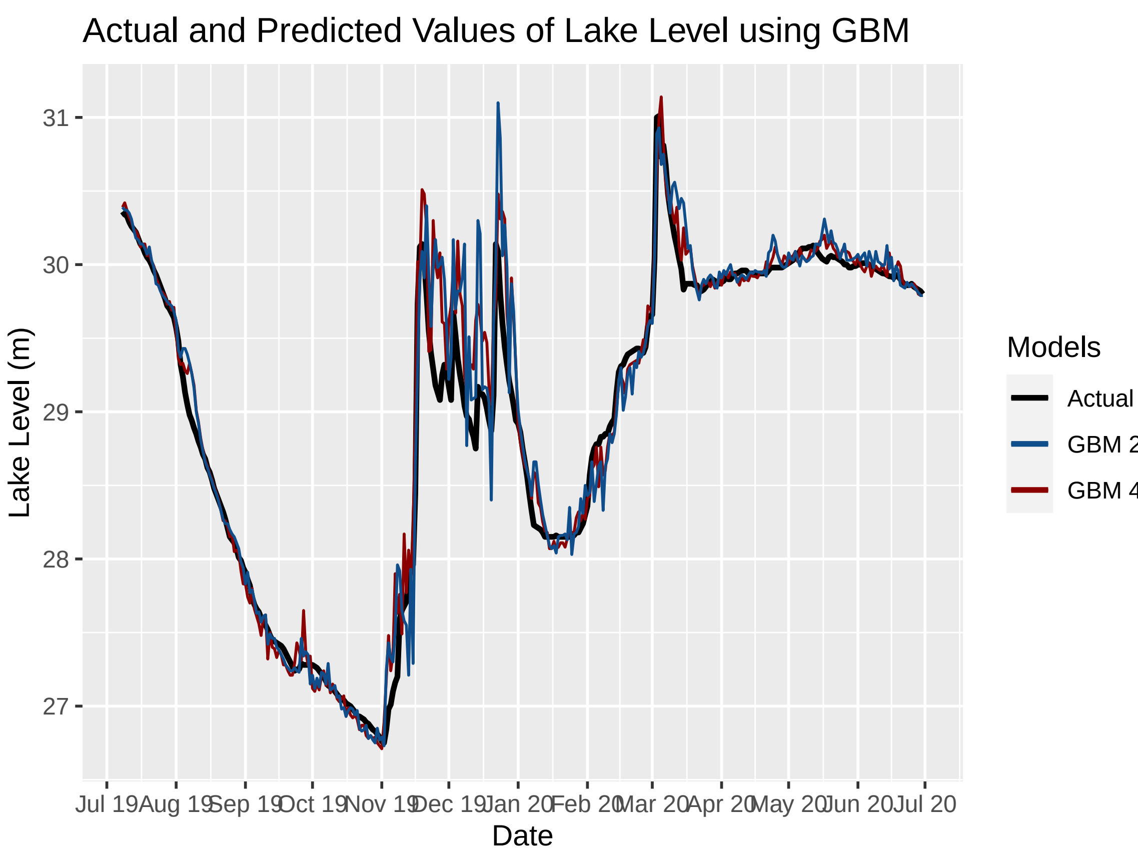


C. Random Forest



Actual and Predicted Values of Lake Level using KNN
Lake Level (m)
31
30
29
28
27
Jul 19
Aug 19
Sep 19
Oct 19
Nov 19
Dec 19
Jan 20
Feb 20
Mar 20
Apr 20
May 20
Jun 20
Jul 20
Date
Models
Actual Values
KNN 24
KNN 41

D. KNN



E. Gradient Boosted