Water Temperature Prediction

Day of Year and Depth XGBoost Model

Gregory Harrison, Claire Oberg, Zihao Li

Nov 28, 2022

Read in Current Data from Lake Sunapee

```
raw <- readr::read_csv("https://s3.flare-forecast.org/targets/sunp/sunp-targets-insitu.csv")
#raw <- readr::read_csv("sunp-targets-insitu.csv")
```

Convert Data to Dataframe

```
# Convert to Dataframe
df = data.frame(raw)

# Generate year and day of year from the date column
df[['year']] <- strptime(df[['date']], format = "%Y-%m-%d")$year + 1900
df[['dayofyear']] <- as.numeric( format(df[['date']], '%j'))

# Filter data according to our assumptions being:
# only calculating at noon
# years during and after 2011 and not including 2015 and 2017 for low amounts of data
# Day of year range only between 162 and 278
df <- df %>%
  filter(hour == 12) %>%
  filter(year >= 2011) %>%
  filter(year != 2017) %>%
  filter(year != 2015) %>%
  filter(dayofyear >= 162) %>%
  filter(dayofyear <= 278)


#Remove the date, variable, and hour columns from our dataframe
df$date = NULL
df$variable = NULL
df$hour = NULL
```

Separate Training and Testing Sets

```
# Separate our data into training (all years but 2013 and 2022) and
# testings (years 2013 and 2022)
waterTrain = df[df$year != 2013 & df$year != 2022, ]
waterTest = df[df$year == 2013 | df$year == 2022, ]

# Drop Columns with NA's in them
waterTrain = na.omit(waterTrain)
waterTest = na.omit(waterTest)
```

Convert Training and Testing set for XGBoost

```
# Remove the year columns now that we've used them to seperate the date
waterTrain$year = NULL
waterTest$year = NULL

# Generate our labels as the current water temperature
```

```
train.label = waterTrain$value
test.label = waterTest$value

# Remove the temperature values from the input data
waterTrain$value = NULL
waterTest$value = NULL

# Convert the input data to a matrix for xgboost
train.data = as.matrix(waterTrain)
test.data = as.matrix(waterTest)
```

Train Model

```
# Generate Training Input for XGBoost
dtrain<-xgb.DMatrix(data = train.data, label = train.label)
# Train our model
bst <- xgboost(data = dtrain, max.depth = 10, eta = 0.3, nthread = 2, nrounds = 20, verbose = 1)
```

```
[1] train-rmse:13.717077
[2] train-rmse:9.653387
[3] train-rmse:6.818378
[4] train-rmse:4.849486
[5] train-rmse:3.490157
[6] train-rmse:2.563179
[7] train-rmse:1.942530
[8] train-rmse:1.540467
[9] train-rmse:1.287220
[10]    train-rmse:1.132652
[11]    train-rmse:1.040263
[12]    train-rmse:0.980041
[13]    train-rmse:0.948792
[14]    train-rmse:0.924430
[15]    train-rmse:0.911197
[16]    train-rmse:0.902342
[17]    train-rmse:0.893986
[18]    train-rmse:0.890046
[19]    train-rmse:0.887670
[20]    train-rmse:0.883808
```

```
# Product Predictions for our Testing Dataset
pred <- predict(bst, test.data)
# Calculate Mean Absolute Error
mean(abs(pred-test.label))
```

```
[1] 1.400848
```

```
# Calculate Root Mean Squared Error
sqrt(mean((pred-test.label)^2))
```

```
[1] 1.683675
```

Generate and Save Histogram of Error

```
error = pred-test.label
DOY = data.frame(actual = test.label, predicted=pred, err = error)

errplt = ggplot(DOY, aes(x=err)) +
 geom_histogram(aes(y=..density..), colour="black", fill="grey")+
 xlim(-6.5,6.5) +
 ggtitle("Model 1 Error") +
 xlab("Error (\u00B0C)") +
 ylab("Density") + theme(text = element_text(size = 30))
```

```
ggsave(plot = errplt, width = 7.5, height = 4.5, dpi = 340, filename = "DOYError.PNG")
```

Generate and Save Predicted vs. Actual Plot

```
abplt = ggplot(DOY, aes(x = actual, y = predicted)) + geom_point(shape=23)+ ggtitle("Model 1 Prediction Result
 xlab("Predicted Value (\u00B0C)") +
 ylab("Actual Value (\u00B0C)") + geom_abline() + theme(text = element_text(size = 30))


ggsave(plot = abplt, width = 7.5, height = 4.5, dpi = 340, filename = "DOYplot.PNG")
```