Explainable AI

Explainable-AI-Analysis Overview

I cannot release the data used for the analysis or the full analysis as it has not yet been published. ExplainableAI_Excerpt.Rmd contains the base of the explainable AI analysis including building and tuning the hyperparameters for the base learners, building a stack ensemble model, and conducting a basic feature importance analysis.

Data

Environment Data

- The weather dataset used to construct the environmental variables contains over 12 million observations of daily weather for the past 50 years in locations around the world. 15 different weather measurements are included.
- Environmental variables like phototermal quotient, photothermal reduction factor, daily thermal time, and more were calculated or modeled using the weather data.
- Weather data was joined to crop yield data by latitude / longitude.

Crop data

- Over 220k observations from locations around the world. Crop yields from multiple observations in the same location and year are averaged.
- Data spans a 50 year period. Not every location is represented each year.
- Includes data such as crop yield and information on key dates in the growth cycle.
- Environmental variables are averaged over key periods in the growth cycles

Analysis

- 4 base learners are integrated to create the stack ensemble model
    - Base models: KNN model, Ranger model, XGboost model, and elastic net model
- Feature importance determined with a loss reduction analysis

```r
library(ranger, quietly = TRUE)
suppressPackageStartupMessages(library(glmnet, quietly = TRUE))
library(xgboost, quietly = TRUE)
library(caret, quietly = TRUE)
library(ggplot2, quietly = TRUE)
library(ggmap, quietly = TRUE)
```

```
## ℹ Google's Terms of Service: <https://mapsplatform.google.com>
##   Stadia Maps' Terms of Service: <https://stadiamaps.com/terms-of-service/>
##   OpenStreetMap's Tile Usage Policy: <https://operations.osmfoundation.org/policies/t
iles/>
## ℹ Please cite ggmap if you use it! Use `citation("ggmap")` for details.
```

```
library(maps, quietly = TRUE)
library(mapdata, quietly = TRUE)
library(dplyr, quietly = TRUE)
```

```
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:xgboost':
##
##     slice
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#library(data.table)
library(DALEX, quietly = TRUE)
```

```
## Welcome to DALEX (version: 2.4.3).
## Find examples and detailed introduction at: http://ema.drwhy.ai/
## Additional features will be available after installation of: ggpubr.
## Use 'install_dependencies()' to get all suggested dependencies
##
## Attaching package: 'DALEX'
##
## The following object is masked from 'package:dplyr':
##
##     explain
```

Base Learner Hyperparameter Tuning:

Note: The run time on this code is extremely long, so I am not evaluating the hyperparameter tuning code chunks to save time - their output is not included.

```
setwd("~/Desktop/ExplainableAI")
df <- read.csv("ExplainableAI_ModelData.csv")

# split into training and test data
train_idx <- sample(seq_len(nrow(df)),size = 0.9*nrow(df)) # 90% of data for training
df_train <- df[train_idx,-1]
df_test <- df[-train_idx,-1]
```

I am training 4 different classes of machine learning models that will be used as base learners to build a stack ensemble model. The stack ensemble model combines the predictions of the base models to yield the final predictions. This helps to account for patterns of errors and biases in the individual base learners.

```
# create folds for cv in hyperparameter tuning
nfolds <- 5
fold_idx <- createFolds(df_train, k = 5)
```

Elastic Net Model:

```
# Conduct preliminary tests to get an idea of good ranges to test for lambda.
train_x <- data.matrix(df_train[,-1])
test_x <- data.matrix(df_test[,-1])
train_y <- df_train[,1]
test_y <- df_test[,1]
# fit models for 11 different choices of alpha
for (i in 0:10) {
  assign(paste("fit", i, sep=""), cv.glmnet(train_x, train_y,alpha=i/10))
}

gnet <- cv.glmnet(train_x,train_y,alpha = 0.9)
gnet$lambda.min
```

```
# Hyperparameter testing
alpha_list <- c(0.7,0.75,0.8,0.85,0.9,0.95,0.9)
lambda_list <-  c(0.0001,0.00025,0.0005,0.00075,0.001,0.0025,0.005,0.0075,0.01,0.025,0.0
5)

enet_parameters <- data.frame(alpha=numeric(),lambda=numeric(),error = numeric())
error <- rep(NA,nfolds)

i <- 1
for(alpha in alpha_list){
  for(lambda in lambda_list){

    error <- rep(NA,nfolds)
    j <- 1
    for(idx in fold_idx){
      cv_train <- df_train[-idx,]
      cv_test <- df_train[idx,]
      fit <- glmnet(as.matrix(cv_train[,-1]),as.matrix(cv_train[,1]),alpha = alpha, lamb
da = lambda)
      preds <- predict(fit, as.matrix(cv_test[,-1]))
      error[j] <- mean(abs(preds - cv_test[,1]) / cv_test[,1]) # using mean absolute rel
ative error
      j <- j + 1
    }
    mae <- mean(error)
    new_row <- c(alpha,lambda,mae)
    enet_parameters <- rbind(enet_parameters,new_row)
    i <- i + 1
  }
}
```

```
enet_parameters[which.min(enet_parameters[,3]),]
enet_final <- glmnet(train_x,train_y,alpha = 0.9, lambda = 0.005)
test_preds <- predict(enet_final, newx = test_x)
enet_error <- mean(abs(test_preds - test_y) / test_y)
enet_error
```

Ranger Model:

```
# Hyperparameter tuning
num_trees <- c(350,500,1000)
m_try <- c(3,5,7,10)
min_node <- c(3,5,15)
ranger_parameters <- data.frame(ntrees=numeric(),m_try=numeric(), min_node=numeric(), er
ror = numeric())

i <- 1
for(ntrees in num_trees){
  print(ntrees)
  for(m in m_try){
    for(min in min_node){
      error <- rep(NA,nfolds)
      j <- 1
      for(idx in fold_idx){

        cv_train <- df_train[-idx,]
        cv_test <- df_train[idx,]

        forest <- ranger(GRAIN_YIELD ~ ., data = cv_train, num.trees = ntrees, mtry = m,
                           min.node.size = min, verbose = F)
        preds <- predict(forest, data = cv_test)$prediction
        error[j] <- mean(abs(preds - cv_test$GRAIN_YIELD)/(cv_test$GRAIN_YIELD))
        j <- j+1
      }

      mae <- mean(error)
      new_row <- c(ntrees,m,min,mae)
      ranger_parameters <- rbind(ranger_parameters,new_row)
      i <- i + 1
    }
  }
}
```

```
ranger_parameters[which.min(ranger_parameters[,4]),]
ranger_final <- ranger(GRAIN_YIELD ~ ., data = df_train, num.trees = 500, mtry = 3,
                           min.node.size = 3, verbose = F)
ranger_preds <- predict(ranger_final, data = df_test)$prediction
ranger_preds_train <- predict(ranger_final, data = df_train)$prediction
error_ranger <- mean((ranger_preds - df_test$GRAIN_YIELD) / df_test$GRAIN_YIELD)
error_ranger
```

XGBoost Model:

```
#Hyperparameter tuning:
rounds <- seq(100,500,length = 5)
eta <- c(0.001, 0.005, 0.0075,0.01, 0.025, 0.05, 0.075, 0.01, 0.25)
max_depth <- c(3,5,7,10,15)
xgboost_parameters <- data.frame(nrounds=numeric,eta=numeric(), max_depth=numeric(), err
or = numeric())

for(nrounds in rounds){
  print(nrounds)
  for(n in eta){
    for(depth in max_depth){
      print(depth)

      error <- rep(NA,5) # 5 fold cross validation
      j <- 1

      for(idx in fold_idx){
        cv_train <- df_train[-idx,]
        cv_test <- df_train[idx,]

        model <- xgboost(data = as.matrix(cv_train[,-1]), label = cv_train$GRAIN_YIELD,
                         max_depth=depth,
                         eta = n,
                         nrounds = nrounds, verbose = F)

        preds <- predict(model, as.matrix(cv_test[,-1]))
        error[j] <- mean(abs(preds - cv_test$GRAIN_YIELD) / cv_test$GRAIN_YIELD) # calcu
lates mean absolute error
        j <- j + 1

      }

      mae <- mean(error) # mean of the mse / mae
      print(mae)
      new_row <- c(nrounds,n,depth,mae)
      xgboost_parameters <- rbind(xgboost_parameters,new_row)

    }
  }
}
```

```
xgboost_parameters[which.min(xgboost_parameters[,4]),]

xgboost_model <- xgboost(data = as.matrix(df_train[,-1]), label = df_train$GRAIN_YIELD,
                         max_depth=10,
                         eta = 0.01,
                         nrounds = 300, verbose = F) # final model with parameters chose
n by cv
xgboost_preds <- predict(xgboost_model, as.matrix(df_test[,-1]))
xgboost_preds_train <- predict(xgboost_model, as.matrix(df_train[,-1]))
xgboost_error <- mean(abs(xgboost_preds - df_test$GRAIN_YIELD) / df_test$GRAIN_YIELD)
xgboost_error
```

KNN Model:

```
# Hyperparameter tuning:
# Standardize all data according to the training set
train_std <- df_train
test_std <- df_test
for(i in names(df_train[-1])){
  colmean <- mean(df_train[,i])
  col_sd <- sd(df_train[,i])
  train_std[,i] <- (df_train[,i] - colmean) / col_sd
  test_std[,i] <- (df_test[,i] - colmean) / col_sd
}

# create function to perform KNN for any given k
knn_func <- function(train, validation, k){

 validation_preds <- rep(NA,nrow(validation))

 for(j in 1:nrow(validation)){
   train_temp <- train
   for (col in names(train)[-1]){
     val <- as.numeric(validation[j,col])
     train_temp[,col] <- (train[,col] - val)^2  # distance between observation and neigh
bor squared
   }

   train_temp <- transform(train_temp, dist=rowSums(train_temp[,-1])) # euclidean distan
ce
   temp <- train_temp[order(train_temp$dist),]
   neighbors <- temp[1:k,] #closest k neighbors based on euclidean distance
   validation_preds[j] <- mean(neighbors$GRAIN_YIELD)
  }

 return(validation_preds)

}
```

```r
ks <- seq(5,100,by = 5)
k_errors <- rep(NA,length(ks))

i <- 1
for(k in ks){
  print(k)
  error <- rep(NA,5)
  j <- 1
  for(idx in fold_idx){
    cv_train <- train_std[-idx,]
    cv_test <- train_std[idx,]
    preds <- knn_func(cv_train,cv_test,k) # call knn function
    error[j] <- mean(abs(preds - cv_test$GRAIN_YIELD)/(cv_test$GRAIN_YIELD))
    j <- j + 1
  }
  k_errors[i] <- mean(error)
  i <- i + 1
}

ks[which.min(k_errors)]
min(k_errors)

# CV chosen parameter: k = 15
```

Explainable Analyses:

Build Stack Ensemble Model

```r
setwd("~/Desktop")
df <- read.csv("ExplainableAI_ModelData.csv")
df <- df[,-1]

df_model <- df[,c(15,16,17,25,30:39)] # choose variables needed for the model
df_final <- aggregate(.~SiteYear, data = df_model, mean)

x_lat <- df_final$Lat
x_lon <- df_final$Long_QC
x_model <- df_final[,-c(1:4)]
y_model <- as.vector(df_final$GRAIN_YIELD)
```

```
KNN_predict <- function(x_train, y_train, x_test, k){

  preds <- rep(NA,nrow(x_test))

  for(col in colnames(x_train)){
    col_mean <- mean(x_train[,col])
    col_sd <- sd(x_train[,col])
    x_train[,col] <- (x_train[,col] - col_mean) / col_sd
    x_test[,col] <- (x_test[,col] - col_mean) / col_sd
  }

 for(j in 1:nrow(x_test)){

   train_temp <- x_train

   for (col in colnames(x_train)){
     val <- as.numeric(x_test[j,col])
     train_temp[,col] <- (x_train[,col] - val)^2 # calculate distance b/n training data
and this observation
   }

   train_temp <- transform(train_temp, dist=rowSums(train_temp))
   neighbor_idx <- order(train_temp$dist)
   neighbors <- y_train[neighbor_idx[1:15]]
   preds[j] <- mean(neighbors) # prediction is the average of the 15 closest neighbors
  }

 return(preds)

}
```

```
# create function to build stack ensemble model
StackEnsembleModel <- function(x_train, y_train){

  # create empty list to store predictions from each model
  ranger_preds <- rep(NA, nrow(x_train))
  xgboost_preds <- rep(NA< nrow(x_train))
  enet_preds <- rep(NA,nrow(x_train))
  knn_preds <- rep(NA,nrow(x_train))

  # create folds
  set.seed(27)
  fold_idx <- createFolds(y_train, k = 5)

  for(idx in fold_idx){

    x_train_cv <- x_train[-idx,]
    y_train_cv <- y_train[-idx]
    x_test_cv <- x_train[idx,]

    # build models - hyper parameters chosen by cross validation earlier
    ranger_model <- ranger(y_train_cv ~ ., data = x_train_cv, num.trees = 500, mtry = 3,
                         min.node.size = 3, verbose = F)
    xgboost_model <- xgboost(data = as.matrix(x_train_cv), label = y_train_cv,
                       max_depth=10,
                       eta = 0.01,
                       nrounds = 300, verbose = F)
    glmnet_model <- glmnet(as.matrix(x_train_cv),as.matrix(y_train_cv),alpha = 0.9, lamb
da = 0.005)

    # make predictions on unseen data
    ranger_preds[idx] <- predict(ranger_model, data = x_test_cv)$prediction
    xgboost_preds[idx] <- predict(xgboost_model, as.matrix(x_test_cv))
    enet_preds[idx] <- predict(glmnet_model, as.matrix(x_test_cv))
    knn_preds[idx] <- KNN_predict(x_train_cv,y_train_cv,x_test_cv, k = 15)
  }

  preds_train <- data.frame(ranger_preds,xgboost_preds,enet_preds,knn_preds,y_train)
  colnames(preds_train) <- c("Ranger","XGBoost","ElasticNet","KNN","GRAIN_YIELD")

  # build Stack Ensemble Model
  model_final <- lm(GRAIN_YIELD ~ ., data = preds_train) # build model based on predicti
ons from base learners
  return(model_final) # returning the regression model

}

# create custom predict function that uses the stack ensemble models to make predictions
predict_StackEnsemble <- function(model,newdata){

  ranger_preds <- predict(model[[1]], data = newdata)$prediction
  xgboost_preds <- predict(model[[2]], as.matrix(newdata))
  enet_preds <- predict(model[[3]], as.matrix(newdata))
  knn_preds <- KNN_predict(model[[4]][[1]],model[[4]][[2]],newdata, k = 15)

  new_x <- data.frame(ranger_preds,xgboost_preds,enet_preds,knn_preds)
  colnames(new_x) <- c("Ranger","XGBoost","ElasticNet","KNN")

  final_predictions <- predict(model[[5]], newdata = new_x) # calling Stack Ensemble mod
el
  return(as.vector(final_predictions))
}
```

```
# build stack ensemble model
StackEnsemble_model <- StackEnsembleModel(x_model,y_model)

# create finalized base learners
ranger_model <- ranger(y_model ~ ., data = x_model, num.trees = 500, mtry = 3,
                             min.node.size = 3, verbose = F)
xgboost_model <- xgboost(data = as.matrix(x_model), label = y_model,
                           max_depth=10,
                           eta = 0.01,
                           nrounds = 300, verbose = F)
glmnet_model <- glmnet(as.matrix(x_model),as.matrix(y_model),alpha = 0.9, lambda = 0.00
5)
knn_model <- list(x_model,y_model)
model_list <- list(ranger_model,xgboost_model,glmnet_model,knn_model,StackEnsemble_mode
l)
```

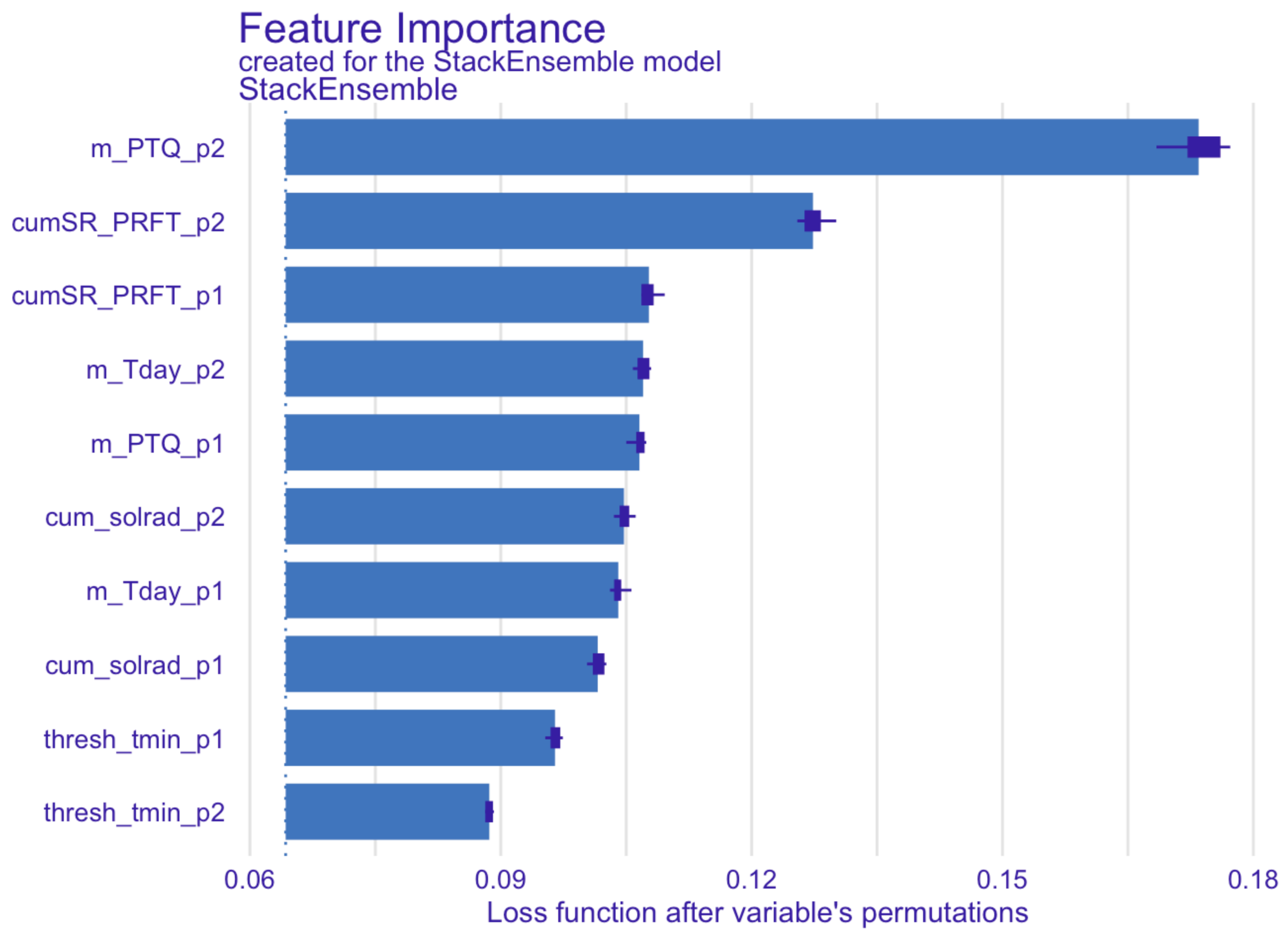
```
# create explainer:
explainer_StackEnsemble <- explain(
  model = model_list,
  data = x_model,
  y = y_model,
  predict_function = predict_StackEnsemble,
  label = "StackEnsemble"
)
```

```
## Preparation of a new explainer is initiated
##   -> model label       :  StackEnsemble
##   -> data              :  2142  rows  10  cols
##   -> target variable   :  2142  values
##   -> predict function  :  predict_StackEnsemble
##   -> predicted values  :  No value for predict function target column. (  default  )
##   -> model_info        :  package Model of class: list package unrecognized , ver. Un
known , task regression (  default  )
##   -> predicted values  :  numerical, min =  2.287883 , mean =  5.910161 , max =  12.3
2244
##   -> residual function :  difference between y and yhat (  default  )
##   -> residuals         :  numerical, min =  -2.132893 , mean =  0.01355981 , max =
3.026002
##   A new explainer has been created!
```

Feature Importance

The importance of a feature is estimated based off of how much the mean absolute relative error increases compared to the original model when a feature is permuted. This feature importance analysis considers each environmental factor included in the prediction model.

```
# create custom loss function (mean absolute relative error)
MAE_loss <- function(observed,predicted){
  return(mean(abs(observed-predicted) / observed))
}
variable_importance <- model_parts(explainer = explainer_StackEnsemble, N = NULL, loss_f
unction = MAE_loss)
print(plot(variable_importance))
```



This

shows that the greatest loss in prediction accuracy occurs when the the photothermal quotient variable (PTQ) is permuted, indicating it is the most “important” variable in generating accurate predictions.

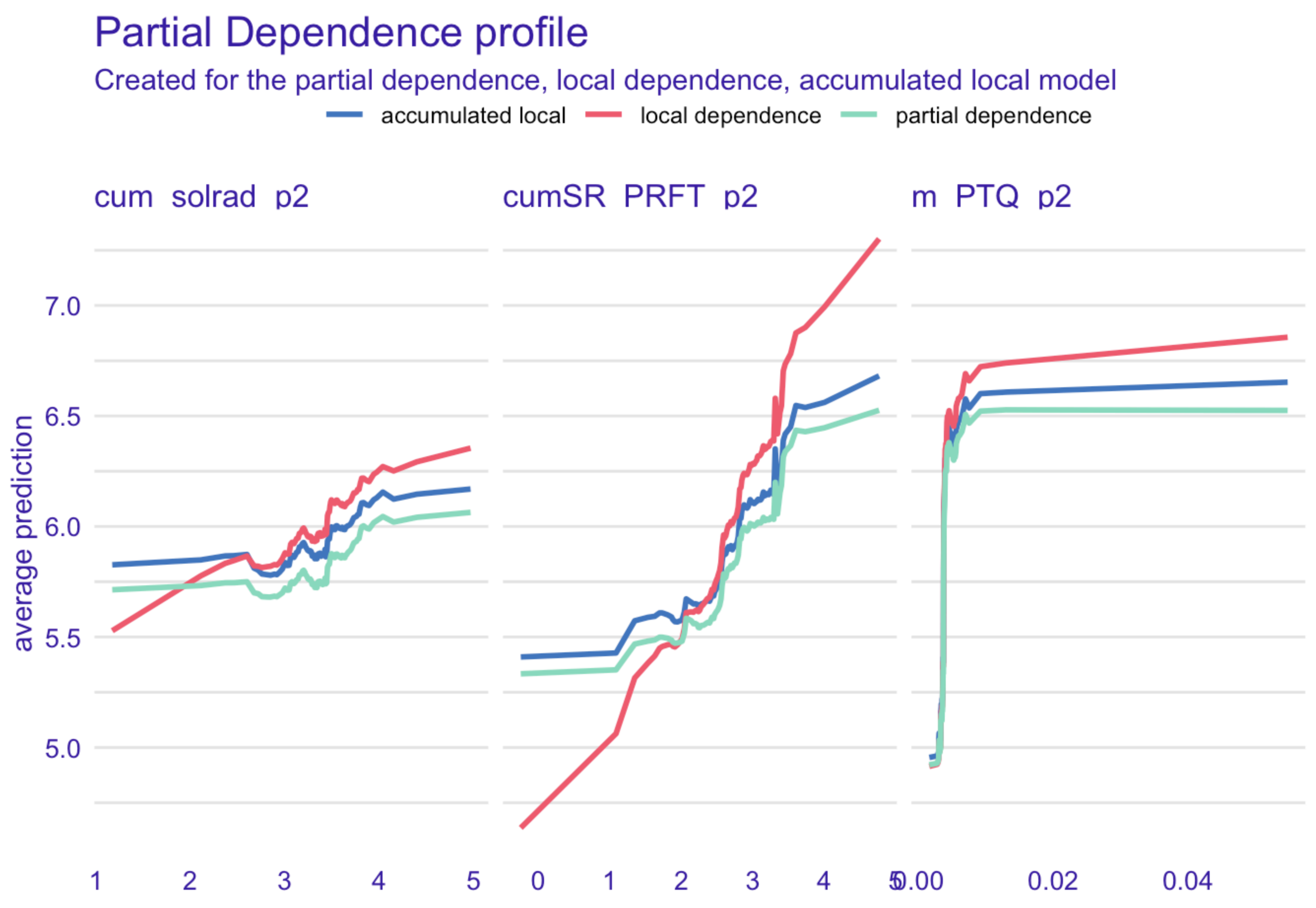
Partial Dependence, Local Dependence, and Accumulated

Local Profiles:

```
# calculate dependence
pd_StackEnsemble <- model_profile(explainer_StackEnsemble, N = 200,
                                  variable =  c("m_PTQ_p2","cum_solrad_p2","cumSR_PRFT_p
2"), type = "partial", loss_function = MAE_loss)
ld_StackEnsemble <- model_profile(explainer_StackEnsemble, N = 200,
                                  variable =  c("m_PTQ_p2","cum_solrad_p2","cumSR_PRFT_p
2"), type = "conditional",loss_function = MAE_loss)
al_StackEnsemble <- model_profile(explainer_StackEnsemble, N = 200,
                                  variable = c("m_PTQ_p2","cum_solrad_p2","cumSR_PRFT_p
2"), type = "accumulated",loss_function = MAE_loss)

# specify labels
pd_StackEnsemble$agr_profiles$`_label_` = "partial dependence"
ld_StackEnsemble$agr_profiles$`_label_` = "local dependence"
al_StackEnsemble$agr_profiles$`_label_` = "accumulated local"


plot(pd_StackEnsemble, ld_StackEnsemble, al_StackEnsemble)
```

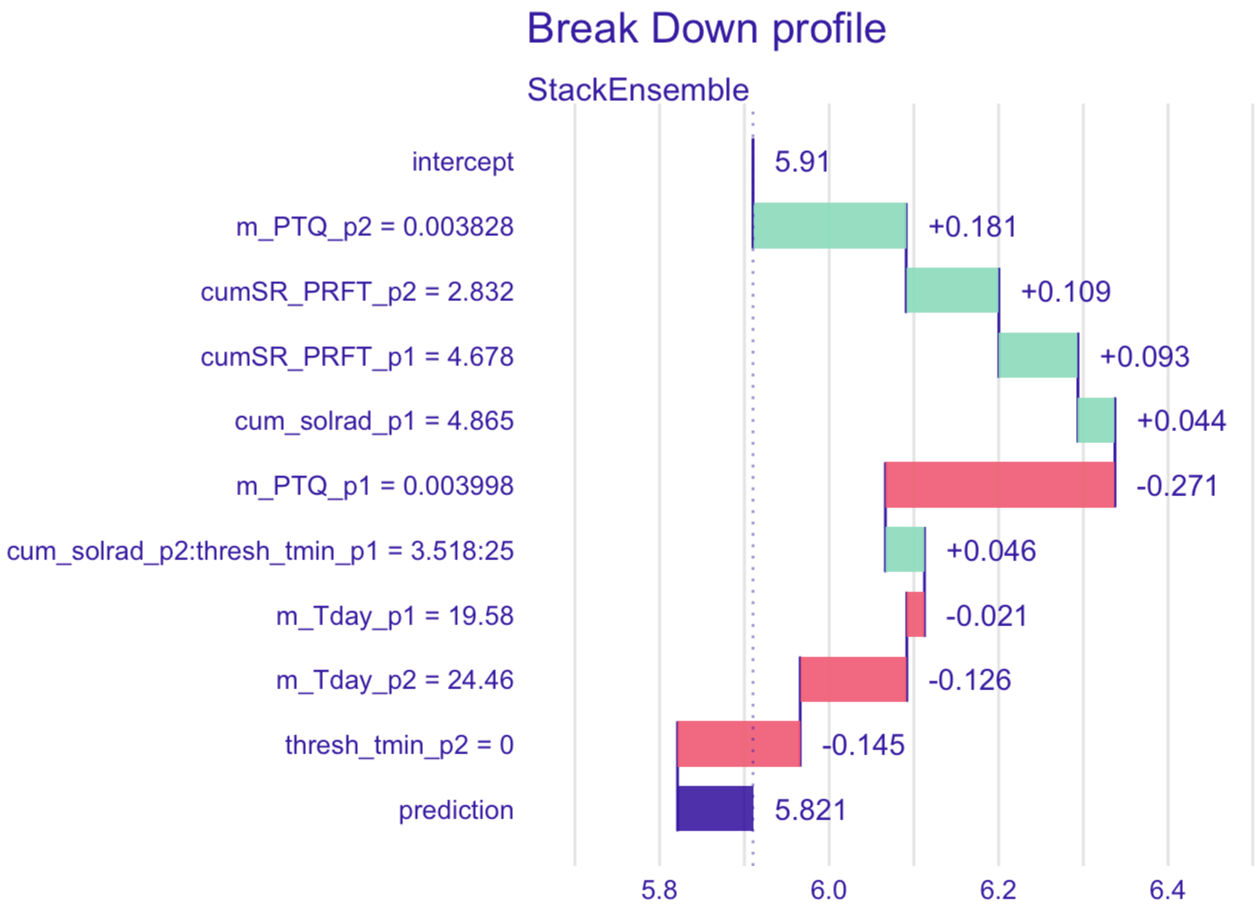


The profiles attempt to explain the effect of a feature on a predicted outcome while other variables are controlled for. When correlated variables are present, the accumulated local effects profile (blue line) is the most accurate representation of this; it shows the pure effect of this singular feature. For the accumulated local effects plot, one should not interpret the effect across intervals. Each interval is created with different

data instances and, therefore, the interpretation of the effect can only be local. These plots help highlight the relationship between an environmental explanatory variable and crop yield, allowing us to assess how individual variables are contributing to the larger trends.

Break Down Plots:

```
bd_StackEnsemble <- predict_parts(explainer = explainer_StackEnsemble,
                                  new_observation = x_model[64,],
                                  type = "break_down_interactions", loss_function = MAE_
loss)
plot(bd_StackEnsemble)
```



Break down plots examine the contribution of each explanatory variable for a single observation. The green bars indicate that, for this observation, the value of the variable contributed positively to the final prediction while the pink bars indicate a negative contribution from the variable. This allows us to examine locations individually and pinpoint which features are negatively affecting crop growth.

Plot Spatial Variability of Contribution Risk Factors:

```
# select one observation from each location (unique lat/lon combo)
locs <- data.frame(x_lat,x_lon)
locs <- unique(locs)
print(nrow(locs))
```

```
## [1] 422
```

```
chosen_idx <- rep(NA,nrow(locs))
for(i in 1:nrow(locs)){
  indices <- which(x_lon == locs$x_lon[i] & x_lat == locs$x_lat[i])
  chosen_idx[i] <- sample(indices,1)
}

x_sample <- x_model[chosen_idx,]
x_lon_sample <- x_lon[chosen_idx]
x_lat_sample <- x_lat[chosen_idx]
```

```
x_tot <- data.frame(x_sample, x_lon_sample, x_lat_sample)
x_tot_sampled <- sample_n(x_tot,75)
x_lon_sample <- x_tot_sampled$x_lon_sample
x_lat_sample <- x_tot_sampled$x_lat_sample
x_sample <- x_tot_sampled[,-c(11,12)]
```

Here I am sampling one observation from each location and then sampling 75 locations in total to plot. In the formal analysis, I examine the trend of a variable’s contribution over time and look at averages over different time periods.

```r
contribution_df <- list() # dataframe of variable contributions for each observation

for(i in 1:nrow(x_sample)){

  if(i %% 10 == 0){
    print(i)
  }

  observation <- x_sample[i,]
  bd_StackEnsemble <- predict_parts(explainer = explainer_StackEnsemble,
                                    new_observation = observation,
                                    type = "break_down_interactions", loss_function = MAE_
loss)

  contribution <- rep(NA,length(colnames(x_model))) # create variable to store contribut
ions, reset for each observation
  j <- 1
  for(name in colnames(x_model)){
    idx <- which(bd_StackEnsemble$variable_name == name)

    if(length(idx) == 0){
      contribution[j] <- 0
      j <- j + 1
    }

    else{
      contribution[j] <- bd_StackEnsemble$contribution[idx]
      j <- j + 1
    }

  }

  contribution_df[[i]] <- contribution

}
```

```
## [1] 10
## [1] 20
## [1] 30
## [1] 40
## [1] 50
## [1] 60
## [1] 70
```

```r
c <- data.frame(contribution_df)
colnames(c) <- NULL
c <- t(c)
colnames(c) <- colnames(x_model)
```

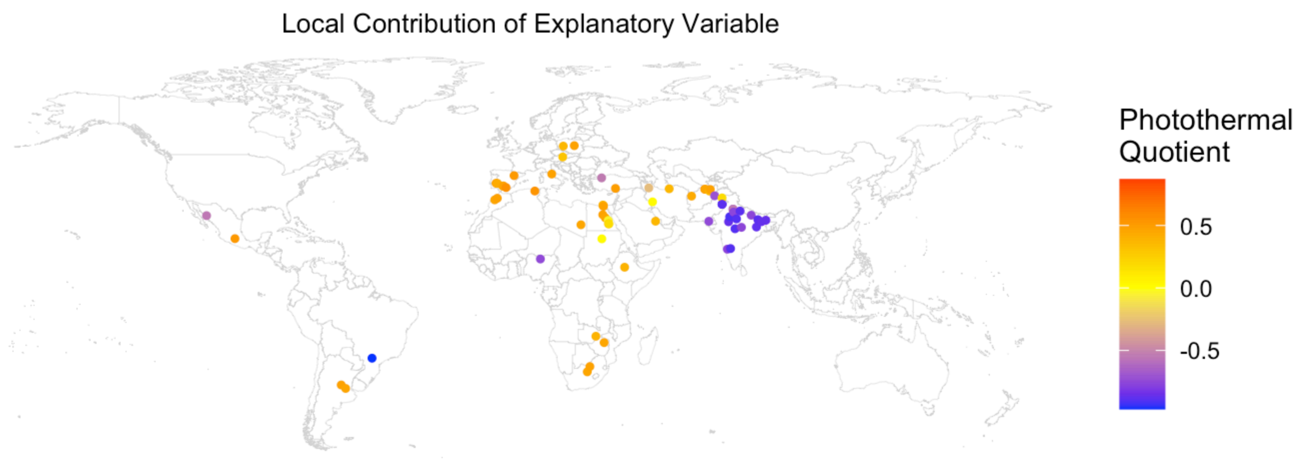
```
map_dat <- data.frame(x_lat_sample,x_lon_sample,c)
colnames(map_dat)[c(1,2)] <- c("lat","lon")
map_dat <- aggregate(.~lat+lon,data = map_dat,mean)
```

```
lower <- min(map_dat[,c(3:12)])
upper <- max(map_dat[,c(3:12)])
```

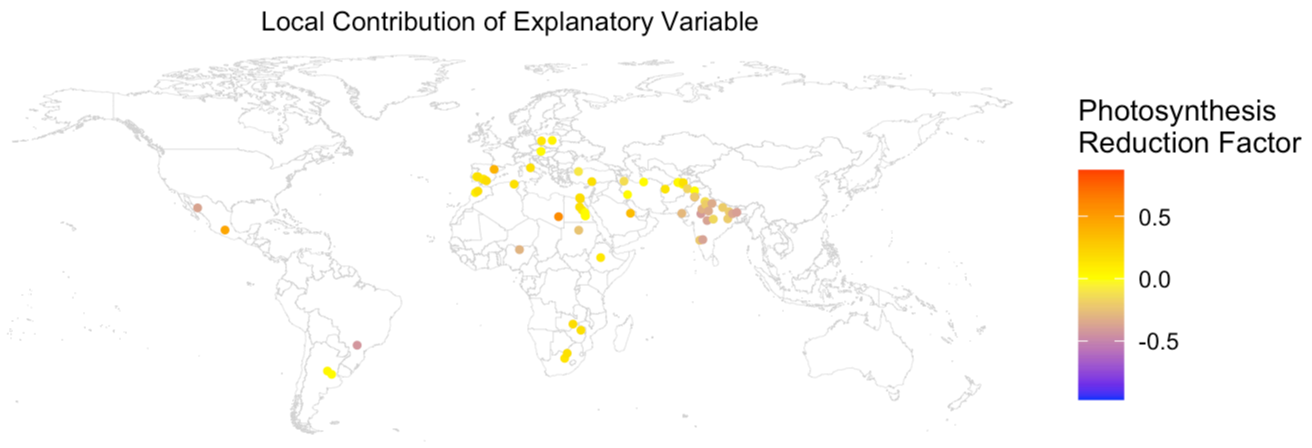
Plotting Variable Contribution

```
world <- map_data("world")
world <- world[world$region != "Antarctica",]

plot <- ggplot() +
  geom_map(data = world, map = world, aes(x = long, y = lat, map_id = region), fill = "w
hite", color = "lightgrey", size = 0.1) +
  geom_point(data = map_dat, aes(x = lon, y = lat, color = m_PTQ_p2), size = 0.9) +
  scale_color_gradient2(low = "blue", mid = "yellow", high = "red",limits = c(lower,uppe
r), name = "Photothermal \nQuotient") +
  theme_void() + coord_fixed() + ggtitle("Local Contribution of Explanatory Variable") +
theme(plot.title = element_text(hjust = 0.5, size = 10))
plot
```



```
plot2 <- ggplot() +
  geom_map(data = world, map = world, aes(x = long, y = lat, map_id = region), fill = "w
hite", color = "lightgrey", size = 0.1) +
  geom_point(data = map_dat, aes(x = lon, y = lat, color = cumSR_PRFT_p2), size = 0.9) +
  scale_color_gradient2(low = "blue", mid = "yellow", high = "red",limits = c(lower,uppe
r), name = "Photosynthesis \nReduction Factor") +
  theme_void() + coord_fixed() + ggtitle("Local Contribution of Explanatory Variable") +
theme(plot.title = element_text(hjust = 0.5, size = 10))
plot2
```



These plots show the variable's contribution in each location so that we can dissect the spatial variation a variable's influence.