Final Project

Introduction

To safeguard agricultural productivity and maintain soil health in Telangana, our research team is conducting an analysis of irrigation water quality across various mandals within the state. Leveraging data obtained from the "Telangana Open Data Portal", we aim to identify potential correlations between water quality parameters and their implications for irrigation practices. This initiative seeks to provide actionable insights for farmers, agricultural scientists, and policymakers to optimize water resource management strategies. From previous pre-processing of the data, we found that the following features might have a bigger impact on our models.

Data Overview:

- **Mandal**: Administrative division in Telangana where the water sample was collected, serving as the primary geographical unit for our analysis.
- **Village**: Specific villages within each mandal from which samples were collected, offering granular data points for our study.
- **RSC** (Residual Sodium Carbonate): Indicates the excess of carbonate and bicarbonate ions over calcium and magnesium ions in water, crucial for evaluating the risk of soil sodicity.
- **SAR** (Sodium Adsorption Ratio): Measures the sodium hazard to crops and soil, a key factor in assessing water's suitability for irrigation.
- **Na** (Sodium content): The concentration of sodium in the water, which influences both SAR and the overall salinity hazard.
- **E.C** (Electrical Conductivity): Reflects the water's salinity level, directly impacting plant water uptake and soil structure.
- **TDS** (Total Dissolved Solids): The overall concentration of dissolved substances in water, indicating potential salinity issues.
- **HCO3** (Bicarbonate level): Concentration of bicarbonate in water, affecting soil pH and the precipitation of calcium and magnesium.

- **pH**: The acidity or alkalinity of the water, with significant implications for nutrient availability and microbial activity in the soil.
- **Classification.1:** Safety classification of the water (Safe P.S., Marginal MR, Unsafe U.S.). This variable is our response variable and has been changed to a binary response by changing MR to U.S.

The primary objective of this study is to establish a comprehensive understanding of irrigation water quality within Telangana's agricultural landscapes. By analyzing the relationships between the above water quality indicators and their collective impact on soil and plant health, we aim to categorize water sources into suitability classes for irrigation. This classification will help in formulating guidelines for water use in agriculture, thereby mitigating the risks associated with inappropriate water sources.

Methods

Logistic Regression Model: (Harris Adnan, Stanley Okoye)

We evaluate a logistic regression model to classify groundwater data based on certain features. We want to then determine the accuracy of our model using logistic regression. Logistic Regression was used because our data is a binary response variable and it is only natural that Logistic Regression is used here because Logistic Regression is popular for working with tasks involving binary outcomes. So, not only do we determine the accuracy of our outcome, but we also evaluate its effectiveness in predicting a binary outcome with Classification.1. Furthermore, Logistic Regression is fast and computationally efficient. Groundwater dataset being massive, and Logistic Regression being suitable for large datasets, it's a natural fit. Logistic Regression is built to handle numerical and categorical variables, especially when it comes to preprocessing steps like one-hot encode, and we did use one-hot encoding and so our code converts the categorical data into a suitable format for modeling. So, with everything in mind, Logistic Regression just became the right choice to classify our dataset, giving a clear interpretation, optimized computation, and smooth integration.

Our approach was based on the knowledge we gained during our lectures and through the slides of Dr. Poliak. Some of these commands were taught to us in class, while some of these commands are based off of research, and through our experience with the Introduction to Artificial Intelligence class. Following are the detailed steps we took for our Logistic Regression model:

Preparing the Data:

We import the libraries needed. For data visualization we import `ggplot2`and for data partitioning and model evaluation we import `caret`. We convert "Classification.1" to binary form. A subset of columns, namely RSC, SAR, Na, E.C, TDS, HCO3, pH, Mandal, and Village from groundwater dataset are selected. Use na.omit() to remove rows with missing data. “Mandal” and “Village” being categorical variables, are converted to factors, because in R, factors represent categorical variables. One-hot encoding is applied to “Mandal” and “Village”. We use `model.matrix()` to create dummy variables, and have Classification.1 as a response variable. This facilitates their use in the regression model. The dummy variables and the response variable are combined to create the final dataset, for which we used “final_data”. The following is our model’s formula.

$$f(x) = \beta_0 + \beta_1 \cdot RSC + \beta_2 \cdot SAR + \beta_3 \cdot Na + \beta_4 \cdot E.C + \beta_5 \cdot TDS + \beta_6 \cdot HCO3 + \beta_7 \cdot pH + \beta_8 \cdot Mandal + \beta_9 \cdot Village$$

```r
library(dplyr)
```

```
Warning: package 'dplyr' was built under R version 4.1.3


Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(caret)
```

```
Warning: package 'caret' was built under R version 4.1.3

Loading required package: lattice
```

```r
groundwater_data <- read.csv("C:/Users/braya/OneDrive/Desktop/4322 Project/4322_project/gr

# Convert 'Classification.1' to binary
groundwater_data$Classification.1 <- ifelse(groundwater_data$Classification.1 == "P.S.", 1
# Select specific columns and remove missing values
selected_data <- groundwater_data %>%
  select(RSCmeqL, SAR, Na, E.C, TDS, HCO3, pH, mandal, village, Classification.1) %>%
  na.omit()


str(selected_data)  # Should show the expected structure
```

```
'data.frame':   972 obs. of  10 variables:
 $ RSCmeqL         : num  -3 -0.4 2 -0.2 -1.6 -1.8 -0.8 -0.4 -0.4 0 ...
 $ SAR             : num  1.9 0.79 3.31 2.61 1.87 2.65 1.12 1.35 0.65 1.17 ...
 $ Na              : int  85 14 118 104 84 125 40 44 15 36 ...
 $ E.C             : int  1065 122 945 1028 1110 1389 620 582 242 504 ...
 $ TDS             : num  682 78 605 658 710 889 397 372 155 323 ...
 $ HCO3            : int  230 40 340 290 300 330 200 180 80 180 ...
 $ pH              : num  8.04 8.21 7.99 8.01 8.11 7.84 8.14 8.02 7.74 7.89 ...
 $ mandal          : chr  "Adilabad" "Bazarhatnur" "Bela" "Bheempur" ...
 $ village         : chr  "Adilabad" "Bazarhatnur" "Chandpally" "Arli" ...
 $ Classification.1: num  1 1 0 1 1 1 1 1 1 1 ...
 - attr(*, "na.action")= 'omit' Named int [1:52] 108 110 111 119 120 125 129 147 239 258 ...
  ..- attr(*, "names")= chr [1:52] "108" "110" "111" "119" ...
```

```r
colnames(selected_data)  # Confirm the selected columns
```

```
 [1] "RSCmeqL"          "SAR"              "Na"               "E.C"
 [5] "TDS"              "HCO3"             "pH"               "mandal"
 [9] "village"          "Classification.1"
```

```r
# Convert categorical variables to factors
selected_data$mandal <- as.factor(selected_data$mandal)
selected_data$village <- as.factor(selected_data$village)

# One-hot encode categorical variables except the response variable
dummy_vars <- model.matrix(~ mandal + village, data = selected_data)

numerical_vars <- selected_data[, c("RSCmeqL", "SAR", "Na", "E.C", "TDS", "HCO3", "pH")]
response_var <- selected_data$Classification.1  # Response variable
final_data <- cbind(as.data.frame(dummy_vars), numerical_vars, Classification.1 = response
# Check that 'final_data' is a data frame
is.data.frame(final_data)  # Should return TRUE
```

```
[1] TRUE
```

Partitioning the Data:

We set our seed to 123 for reproducibility. We split "final_data" into training and testing sets using `createDataPartition`. This makes our model use 80% for training and the remaining 20% for testing with 10 different splits when placed in the for loop. The split is stratified based on the response variable, ensuring the equal distribution of classes in both sets.

Logistic Regression:

We build our Logistic Regression model using glm with a binomial family, and having classification.1 as our response variable, and the rest of the columns as predictors. To see information about coefficients, their significance, and other details we use the `summary(model)` command. In this case we only get to see the summary for "mandal" and "village".

Making Predictions

In order to make predictions on the test set, we use `predict()` command. Since Logistic Regression is popular for its usage to predict the probability of an outcome, we use the `Type = "response"` argument to indicate that predictions should be probabilities. Now we want to convert our predicted probabilities to binary class predictions, for which we used the command `ifelse(predictions > 0.5, 1, 0)`, using a threshold of 0.5. This indicates that if the probability is greater than 0.5, the predicted class is 1, otherwise, it's 0.

```
# List of Confusion Matrices
conf_mat = list()

# List of Accuracies
accuracies = list()

# Partition the data into training and testing sets
set.seed(123)
conf_mat <- list()
accuracies <- numeric(10)

for (i in 1:10) {
  trainIndex <- createDataPartition(final_data$Classification.1, p = 0.8, list = FALSE)
  trainData <- final_data[trainIndex, ]
  testData <- final_data[-trainIndex, ]
  if (i == 5){
    train = trainData
    test = testData
  }

  # Build a logistic regression model with 'Classification.1' as the response variable
  model <- glm(Classification.1 ~ ., data = trainData, family = binomial)

  # Make predictions on the test set
  predictions <- predict(model, newdata = testData, type = "response")
  predictedClass <- ifelse(predictions > 0.5, 1, 0)

  # Create a confusion matrix
  conf_mat[[i]] <- confusionMatrix(factor(predictedClass), factor(testData$Classification.

  # Calculate accuracy
  accuracies[i] <- conf_mat[[i]]$overall['Accuracy']
}
```

```
Warning: glm.fit: algorithm did not converge


Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading


Warning: glm.fit: algorithm did not converge


Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading


Warning: glm.fit: algorithm did not converge


Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading


Warning: glm.fit: algorithm did not converge


Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading
```

```
Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading

Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading

Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading

Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading

Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading

Warning: glm.fit: algorithm did not converge

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
prediction from a rank-deficient fit may be misleading
```

```r
best_model = glm(Classification.1 ~., data = train, family = binomial)

summary(best_model)

# Hard to interpret, so echo set to FALSE!!!
```

Accuracy

```
# Print accuracies
print(accuracies)
```

```
 [1] 0.5206186 0.8969072 0.8505155 0.9123711 0.9226804 0.8917526 0.8711340
 [8] 0.8814433 0.8865979 0.8195876
```

With the help of Dr. Poliak's lectures and slides, we create a confusion matrix to compare predicted class with the actual class in test set. The confusion matrix is important as it assists in the evaluation of the performance of our model. Now comes the main part, that is calculating the accuracy. To calculate accuracy we used the command `accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)`. In simple words, divide the sum of the diagonal elements in the Confusion Matrix by the total number of predictions. We get an accuracy ranging between 52.1% to 92.3%, indicating that the logistic regression model correctly predicts the outcome between 52.1% to 92.3% of the time.

Summary

We pre-process the data, apply logistic regression modeling, and evaluate accuracy to predict binary classifications for groundwater dataset. One-hot encoding technique was used for "Mandal" and "Village", data was partitioned, and confusion matrix was computed to analyze the model's performance.

Random Forest Model: (Brayan Gutierrez, Ulises Ramirez, Tie Wang)

To mitigate issues such as overfitting, high variance, and poor generalization to unseen data, commonly associated with a single decision tree, we opted for an ensemble method employing decision trees. While the bagging ensemble model addresses the shortcomings of individual decision trees, it introduces a unique challenge. In the bagging model, variables tend to be highly correlated due to the consistent selection of the most important variables in each decision tree. To address these challenges and leverage the robustness of random forest models, we chose this particular ensemble method. However, by adopting the random forest model, we sacrifice some interpretability inherent in a single decision tree, as well as computational resources, and introduce the need for hyperparameter tuning, which is less significant in a single decision tree. Similar to the previous model, our response variable is "Classification.1", with the remaining variables serving as predictors. Below is the formula for our random forest model:

$$\text{Classification.1} \sim \text{Mandal} + \text{Village} + \text{RSC} + \text{SAR} + \text{Na} + \text{E.C} + \text{TDS} + \text{HCO3} + \text{pH}$$

Firstly, we aimed to evaluate the performance of the base random forest with default hyperparameters on our dataset. We read the data, removed NA observations, and converted the response variable to a binary format. Additionally, as the Mandal and Village variables contained multiple categories and to ensure proper functionality of the model, we employed one-hot encoding using the `dummyVars()` function from the *caret* library.

```r
library(tree)
```

```
Warning: package 'tree' was built under R version 4.1.3
```

```r
library(randomForest)
```

```
Warning: package 'randomForest' was built under R version 4.1.3
```

```
randomForest 4.7-1.1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```

Attaching package: 'randomForest'
```

```
The following object is masked from 'package:ggplot2':

    margin
```

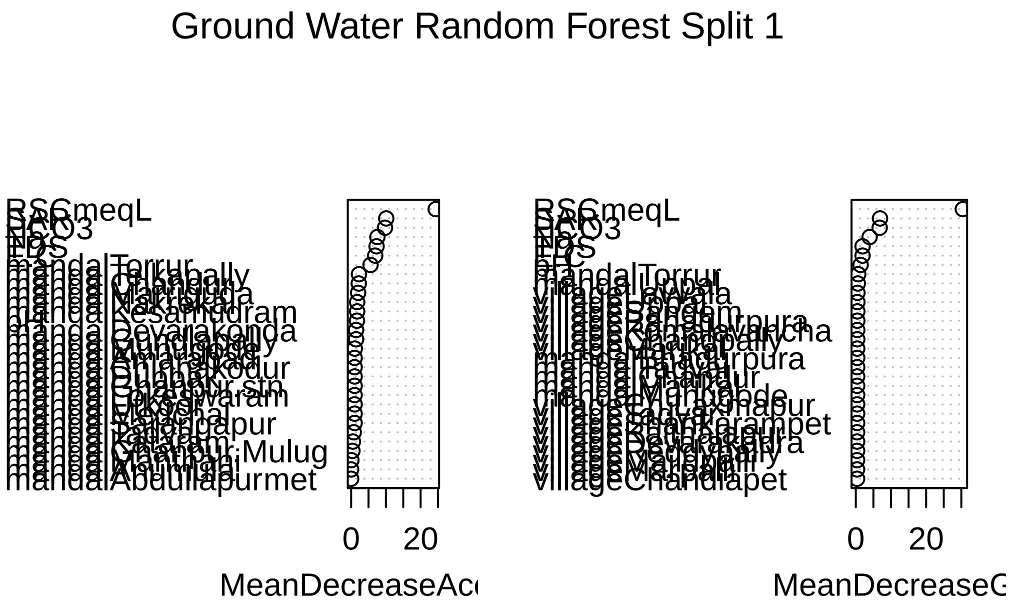
```
The following object is masked from 'package:dplyr':

    combine
```

```r
library(caret)

ground_water_quality_2022_post <- read.csv("ground_water_quality_2022_post.csv", header =

ground_water_quality_2022_post = na.omit(ground_water_quality_2022_post)

# Omitting CO3 and Season since they're the same
ground_water_quality_2022_post$CO3 = NULL
ground_water_quality_2022_post$season = NULL

# Selecting the most useful features
```

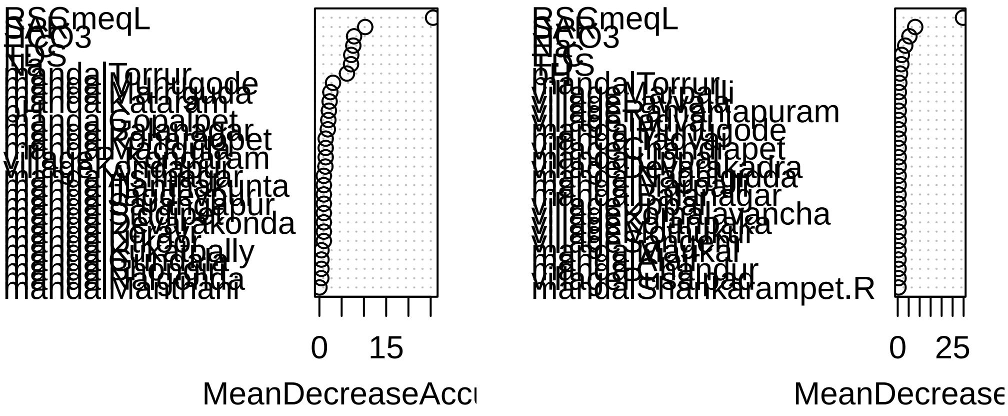
```
gw_df = ground_water_quality_2022_post[,c(23, 21, 16, 9, 10, 11, 8, 3, 4)]

# One-hot-encoding Mandal and Village
dummy = dummyVars(" ~ .", data = gw_df)
dum_df = data.frame(predict(dummy, newdata = gw_df))

# Changing Marginal Safe (MR) to Unsafe (U.S.)
Classification.1 = ground_water_quality_2022_post[,24]
final_gw_df = data.frame(dum_df, Classification.1)
final_gw_df$Classification.1 = gsub("MR", "U.S.", final_gw_df$Classification.1)
final_gw_df$Classification.1 = as.factor(final_gw_df$Classification.1)
```

To obtain the testing error for a classification random forest model, we would need to generate a confusion matrix for each test set. As we are testing the model 10 times with different 80% training and 20% testing splits, we decided to store all the confusion matrices for these splits in a list called "conf_mat". Additionally, we calculated and stored each model's out-of-bag score in a list named "OOB" before the loop. The model was trained on the 10 different training sets and then tested on 10 different testing sets within a for loop. We used a seed of 1 for reproducibility. Since the `cv.tree()` function only works for regression random forest models, we didn't use it here. Furthermore, we plotted the variable importance for each training/testing split. Below is the code for this process:

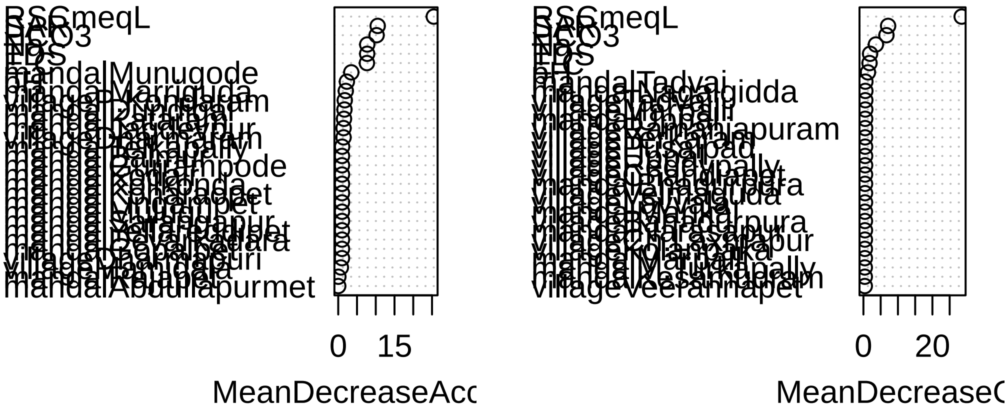
```
conf_mat = list()
OOB = rep(0,10)

set.seed(1)

for (i in 1:10) {

  # Splitting data
  sample = sample(1:nrow(final_gw_df), 0.8 * nrow(final_gw_df))
  training = final_gw_df[sample,]
  testing = final_gw_df[-sample,]

  # Random Forest Model
  rf_gw = randomForest(Classification.1 ~., data = training, proximity = T, importance = T

  # OOB of Models
  conf = rf_gw$confusion[,-ncol(rf_gw$confusion)]
  OOB[i] = 1 - (sum(diag(conf)) / sum(conf))
  # Importance Plots
  importance(rf_gw)
  varImpPlot(rf_gw, main = paste('Ground Water Random Forest Split', i, sep = ' '))

  test.pred = predict(rf_gw, newdata = testing)

  # Confusion Matrix
  conf_mat[[i]] = table(testing$Classification.1, test.pred)

}
```



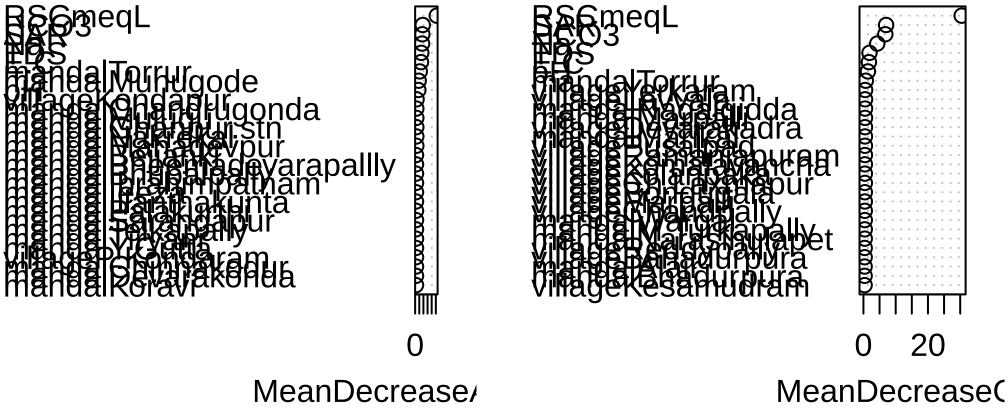
Ground Water Random Forest Split 2



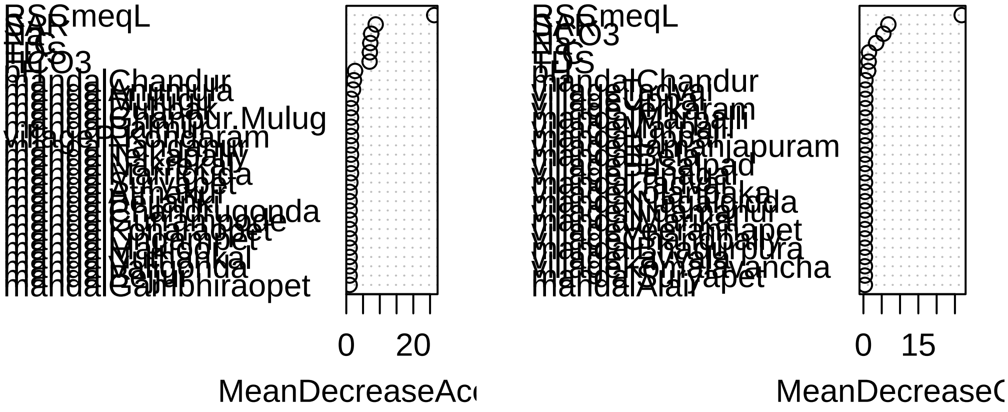
Ground Water Random Forest Split 3



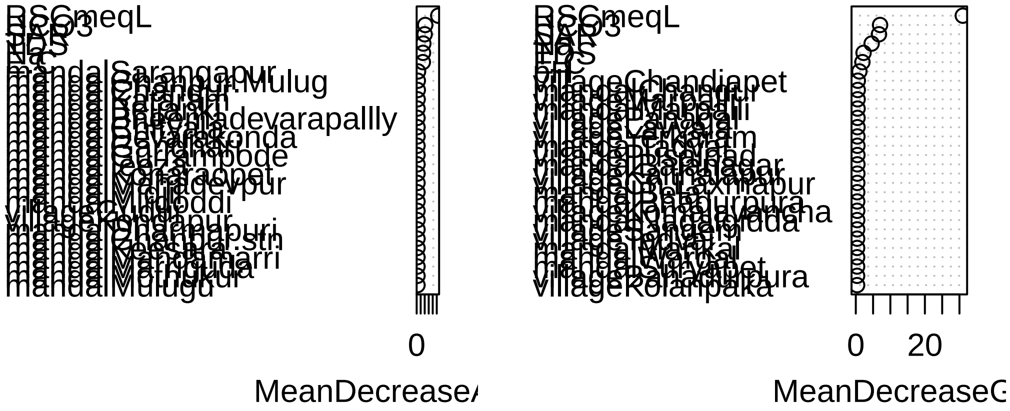
Ground Water Random Forest Split 4



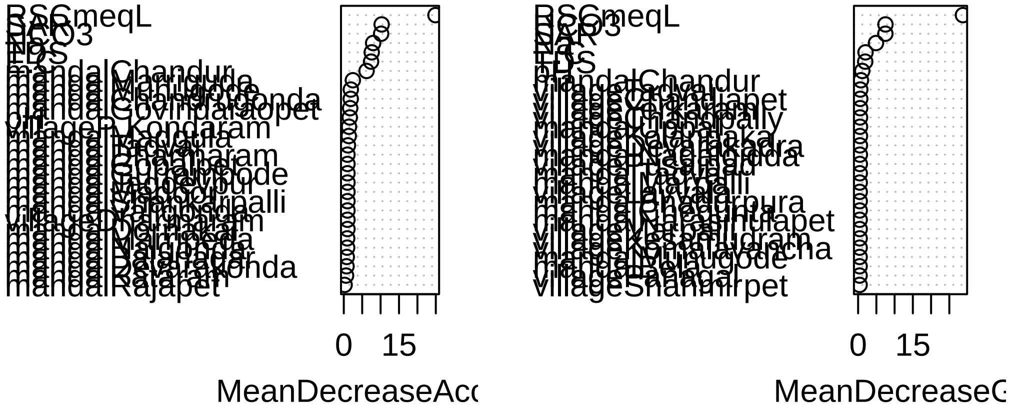
Ground Water Random Forest Split 5



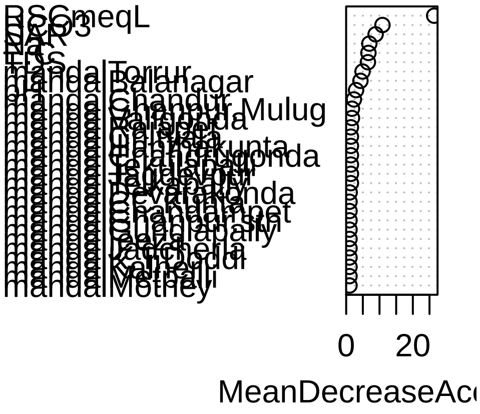
Ground Water Random Forest Split 6



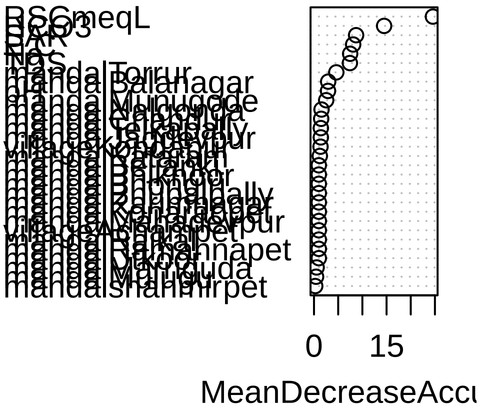
Ground Water Random Forest Split 7

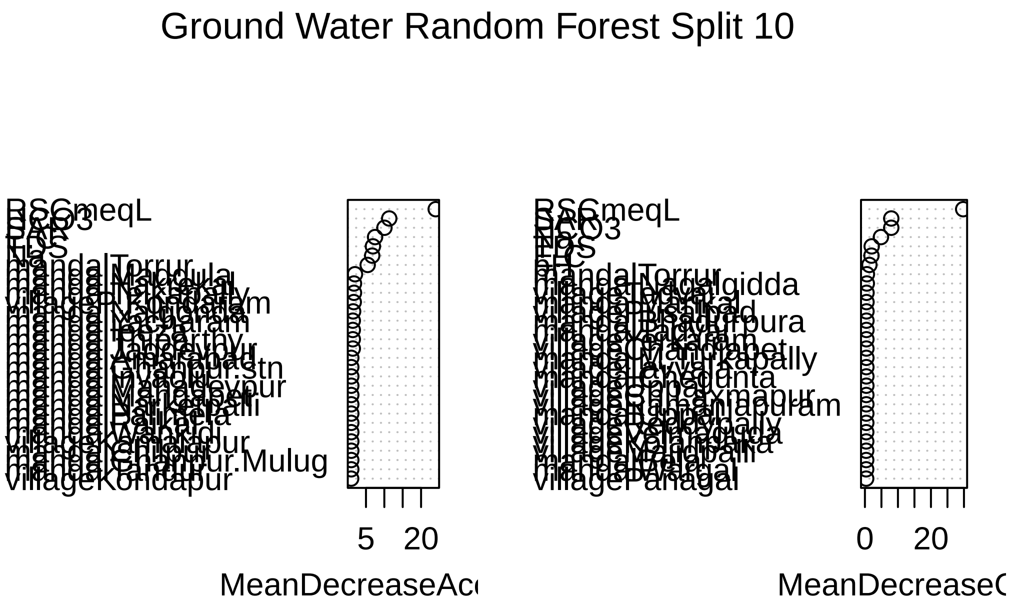


Ground Water Random Forest Split 8



Ground Water Random Forest Split 9





We observed that the three most important variables in each split were RSC, SAR, and HCO3, with SAR and HCO3 competing for the second most important variable across the different splits. Now that we have saved the confusion matrices for each training/testing split, we can utilize them to calculate the testing error for each split.

```
# Test Error Calculations
test_errors = rep(0, 10)
accuracies = rep(0, 10)

for (i in 1:10) {
    TP = conf_mat[[i]][2, 2]  # True Positives
    FP = conf_mat[[i]][1, 2]  # False Positives
    FN = conf_mat[[i]][2, 1]  # False Negatives
    TN = conf_mat[[i]][1, 1]  # True Negatives

    # Calculating test error rate
    test_errors[i] = (FP + FN) / sum(conf_mat[[i]])

    # Calculating Accuracies
    accuracies[i] = (TP + TN) / sum(conf_mat[[i]])
}
```

```
scores = list(test_error = test_errors, OOB_score = OOB, accuracy = accuracies)

scores
```

```
$test_error
 [1] 0.024390244 0.006097561 0.006097561 0.006097561 0.006097561 0.000000000
 [7] 0.012195122 0.000000000 0.012195122 0.012195122

$OOB_score
 [1] 0.015313936 0.006125574 0.015313936 0.007656968 0.009188361 0.007656968
 [7] 0.015313936 0.006125574 0.016845329 0.021439510

$accuracy
 [1] 0.9756098 0.9939024 0.9939024 0.9939024 0.9939024 1.0000000 0.9878049
 [8] 1.0000000 0.9878049 0.9878049
```

```
(avg_test_error = sum(test_errors) / 10)
```

```
[1] 0.008536585
```

We discovered that the base random forest model, evaluated across 10 different training/testing splits, exhibited test errors ranging from 0% to 2.4%, with an average testing error of 0.8%. Comparing these testing errors with the out-of-bag (OOB) scores, we observed minimal disparity, indicating that the base random forest model performs admirably with unseen data, affirming its ability to generalize well beyond the training set. This suggests a robust learning of underlying patterns within the data. However, it's worth noting the limitation posed by the fewer U.S. observations in our dataset, which might affect the model's performance on U.S. data. With this promising result, we proceeded to fine-tune the hyperparameters of the random forest model on our dataset.

Model Optimization Results:

The model optimization process involved an exhaustive search for the optimal combination of hyperparameters to maximize prediction accuracy while preventing overfitting. This was achieved through grid search, where various values of hyperparameters such as the number of estimators (trees), maximum depth, minimum samples split, minimum samples leaf, and maximum features were explored. Each hyperparameter combination was evaluated using cross-validation, which split the data into training and validation sets to assess model performance.

The optimization process was repeated across 10 different 80/20 splits of the dataset to ensure robustness and generalizability of the results. This approach allowed for a comprehensive exploration of the hyperparameter space and provided insight into the sensitivity of the model to different parameter settings. By leveraging multiple splits, the final model selection was based on the mean cross-validation score across all splits, ensuring that the chosen model performed well on diverse subsets of the data.

Impact of Hyperparameters on Model Performance:

The hyperparameters explored during grid search play a crucial role in determining the performance of the Random Forest model. The number of estimators (trees) determines the complexity of the model and its ability to capture intricate relationships within the data. Increasing the number of trees can improve predictive accuracy up to a certain point, beyond which additional trees may lead to diminishing returns or increased computational costs.

The maximum depth of the trees controls the depth of individual decision trees in the Random Forest ensemble. Deeper trees can capture complex interactions but may also be prone to overfitting, especially with small datasets. By limiting the maximum depth, the model can generalize better to unseen data while still capturing important patterns in the training data.

Similarly, the minimum samples split and minimum samples leaf parameters regulate the minimum number of samples required to split an internal node and the minimum number of samples required to be a leaf node, respectively. These parameters help control the granularity of the decision tree structure and prevent overfitting by imposing constraints on tree growth.

The choice of maximum features determines the number of features considered for each split in the decision trees. By randomly selecting a subset of features, Random Forest mitigates the risk of feature dominance and enhances model diversity, leading to improved generalization performance.

```r
# Define parameter grid for grid search
param_grid <- expand.grid(
  n_estimators = c(50, 100, 200),
  max_depth = c(5, 10, 15),
  min_samples_split = c(2, 5, 10),
  min_samples_leaf = c(1, 3, 5),
  max_features = c("sqrt", "log2"),  # Ensure max_features is specified as characters
  stringsAsFactors = FALSE  # Ensure strings are not converted to factors
)

# Initialize empty list to store results
results <- list()
```

Robustness and Generalizability of the Model:

By performing grid search and cross-validation across multiple data splits, the optimized Random Forest model demonstrates robustness and generalizability. The model's ability to consistently perform well across diverse subsets of the data indicates its stability and reliability.

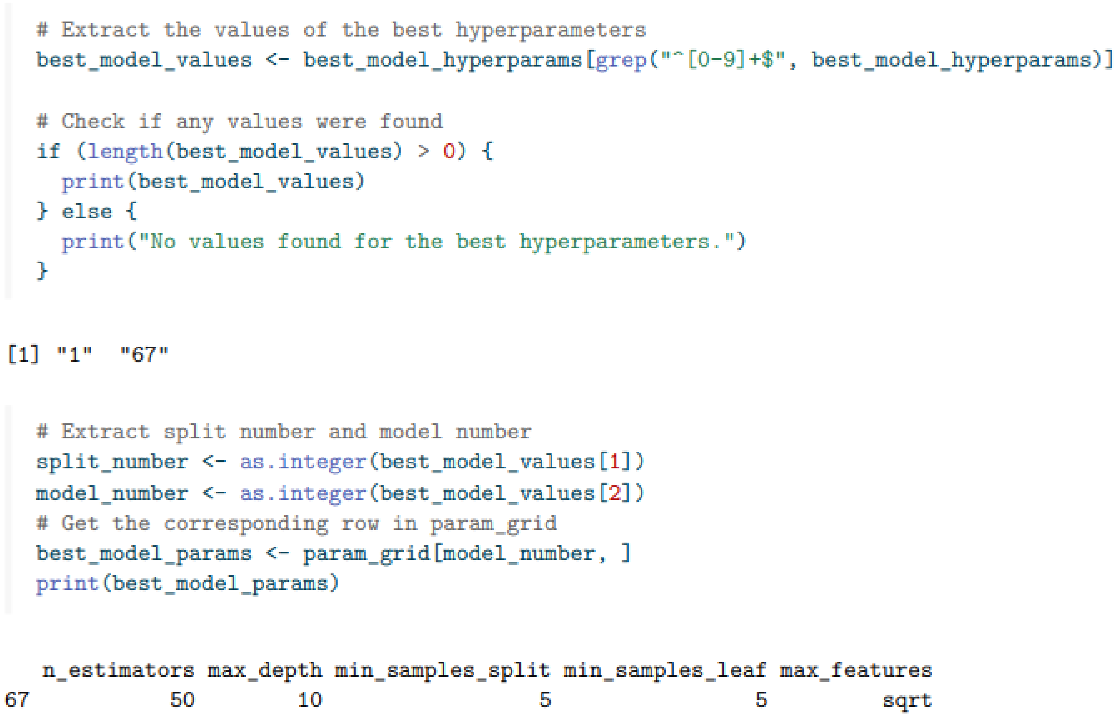
This ensures that the selected hyperparameters are not overly tailored to a specific subset of the data but rather capture underlying patterns that generalize to unseen instances. Due to long run time for the code below, code chunk evaluation was set to FALSE.

```r
# Perform 10 different 80/20 splits and grid search on each split
for (split in 1:10) {
  # Set seed for reproducibility
  set.seed(1)

  # Split data into training and testing sets (80/20 split)
  sample <- sample(1:nrow(final_gw_df), 0.8 * nrow(final_gw_df))
  training <- final_gw_df[sample, ]
  testing <- final_gw_df[-sample, ]

  # Initialize cross-validation folds for grid search
  folds <- createFolds(training$Classification.1, k = 5, list = TRUE, returnTrain = FALSE)

  # Perform grid search
  for (i in 1:nrow(param_grid)) {
    params <- param_grid[i, ]

    # Initialize vector to store cross-validation scores
    cv_scores <- c()

    # Perform cross-validation
    for (fold in folds) {
      # Split training data into training and validation sets
      train_data <- training[-fold, ]
      val_data <- training[fold, ]

      # Determine the value of mtry based on max_features
      if (params$max_features == "sqrt") {
        mtry_value <- sqrt(ncol(train_data) - 1)  # Subtract 1 for the target variable
      } else if (params$max_features == "log2") {
        mtry_value <- log2(ncol(train_data) - 1)
      } else {
        mtry_value <- params$max_features
      }

      # Fit random forest model
      rf_gw <- randomForest(Classification.1 ~ ., data = train_data,
                            ntree = params$n_estimators,
                            mtry = round(mtry_value),  # Round to nearest integer
                            maxdepth = params$max_depth,
                            nodesize = params$min_samples_leaf)

      # Make predictions on validation set
      val_pred <- predict(rf_gw, newdata = val_data)

      # Calculate accuracy
      accuracy <- sum(val_pred == val_data$Classification.1) / length(val_data$Classificat

      # Append accuracy to vector
      cv_scores <- c(cv_scores, accuracy)
    }

    # Store mean cross-validation score in results list
    results[[paste("Split", split, "Model", i, sep = "_")]] <- mean(cv_scores)
  }
}


#print(results)
results_df <- data.frame(Model = names(results), Score = unlist(results))
# Find the row with the highest score
best_model <- results_df[which.max(results_df$Score), ]
print(best_model)
best_model_name <- best_model$Model
# Find the corresponding parameters in the param_grid
best_model_params <- param_grid[grep(best_model_name, rownames(param_grid)), ]
print(best_model_params)


best_model_name <- best_model$Model
best_model_hyperparams <- unlist(strsplit(best_model_name, "_"))
# Extract the values of the best hyperparameters
best_model_values <- best_model_hyperparams[grep("^[0-9]+$", best_model_hyperparams)]

# Check if any values were found
if (length(best_model_values) > 0) {
  print(best_model_values)
} else {
  print("No values found for the best hyperparameters.")
```

```
}


# Extract split number and model number
split_number <- as.integer(best_model_values[1])
model_number <- as.integer(best_model_values[2])
# Get the corresponding row in param_grid
best_model_params <- param_grid[model_number, ]
print(best_model_params)
```

```
print(getwd())
```

```
[1] "C:/Users/braya/OneDrive/Desktop/4322 Project/4322_project"
```



```
# Extract the values of the best hyperparameters
best_model_values <- best_model_hyperparams[grep("^[0-9]+$", best_model_hyperparams)]

# Check if any values were found
if (length(best_model_values) > 0) {
  print(best_model_values)
} else {
  print("No values found for the best hyperparameters.")
}
```

```
[1] "1"  "67"
```

```
# Extract split number and model number
split_number <- as.integer(best_model_values[1])
model_number <- as.integer(best_model_values[2])
# Get the corresponding row in param_grid
best_model_params <- param_grid[model_number, ]
print(best_model_params)
```

```
   n_estimators max_depth min_samples_split min_samples_leaf max_features
67           50        10                 5                5         sqrt
```

Figure 1: Hyperparameter Tuning Results

The results of the model evaluation indicate that the model labeled as "Split_1_Model_67" achieved a mean cross-validation score of approximately 0.9908 for Split 1 of the data. This score suggests that the model performed exceptionally well on the validation data for this particular split. Further analysis reveals that this model was trained with a specific combination of hyperparameters, which can be found below the Score.

In conclusion, the machine learning model developed using Random Forests and optimized through cross-validation demonstrates high predictive accuracy for the task of groundwater quality prediction. The results of the evaluation process provide valuable insights into the performance of the model and offer guidance for future model development and refinement.

Future Directions:

While the optimized Random Forest model shows promising results in predicting ground water quality, there are opportunities for further refinement and enhancement. Future research may explore advanced techniques for hyperparameter tuning, such as Bayesian optimization or genetic algorithms, to more efficiently navigate the hyperparameter space. Additionally, incorporating domain knowledge and additional features, such as meteorological data or land use information, could further improve the model's predictive capabilities and provide valuable insights for environmental monitoring and management. Finally, ongoing validation and evaluation of the model on independent datasets are essential to ensure its applicability and reliability in real-world scenarios.

Results

Logistic Regression:

Due to the one-hot-encoding of our categorical variables, Mandal and Village, the interpretability of the summary of our best logistic regression model, which scored 92.3% accuracy, is difficult. Thus, we were not able to easily determine which features were significant and their respective coefficients. As a result, we were not able to address our objective of identifying correlations between water quality parameters using logistic regression. However, we were able to see that this model performed adequately in predicting unseen data. The model using the first training/testing split performed quite poorly with an accuracy of 52%, but in general performed well with the other splits. With this information, we can say that our logistic regression model may be used in predicting water safety, but it's not really the best if it's the only model used.

Random Forest:

In general, we observed that the random forest model, whether base or tuned, performed well on the testing data. This indicates that our models are robust and generalizable. Starting with the base model, among the 10 different splits, we observed testing errors between 0% to 2.4% and accuracies between 97.6% to 100%. This suggests that our variables positively contributed to our base model's classification of the test observations, with RSC, SAR, and HCO3 being the most important across all models. It is safe to assume that these top variables, along with the other important variables, will be present during the tuning process of the random forest model. This addresses our objective of identifying correlations between water quality parameters and their implications for soil health. By tuning the model, we were able to find the optimal hyperparameters that minimized the model's test error and maximized accuracy on unseen data. We observed that the parameters that led to the best results were 50 estimators, a max depth of 10, a minimum sample split of 5, a minimum leaf sample of 5, and a maximum number of features of $\sqrt{p}$ where p is the total number of features. With these hyperparameters, and the previously identified water quality parameters, we ensure that our model would have a classification accuracy of approximately 99%.

Conclusion

To recap, we were not able to determine the structure of our optimal logistic model due to its lack of interpretability. However, in general, the models tended to perform well with unseen data, with the first model being the outlier, scoring an accuracy of 52%. Conversely, we were able to identify the most important variables used in the random forest models, which showed the correlation between water quality parameters. Additionally, through hyperparameter tuning, we determined the best parameters for our random forest model, resulting in a classification accuracy of approximately 99%. Consequently, we concluded that the random forest model was the best choice for our water safety classification problem.

Bibliography

Telangana Ground Water Department - Post monsoon Water Quality Data. Telangana Open Data Portal. (2022). https://data.telangana.gov.in/dataset/telangana-ground-water-department-post-monsoon-water-quality-data

Poliak, C. (2024). *Logistic Regression* [PowerPoint slides]. University of Houston Department of Mathematics.

Poliak, C. (2024). *Lecture 18 Tree Based Methods: Random Forest* [PowerPoint slides]. University of Houston Department of Mathematics.