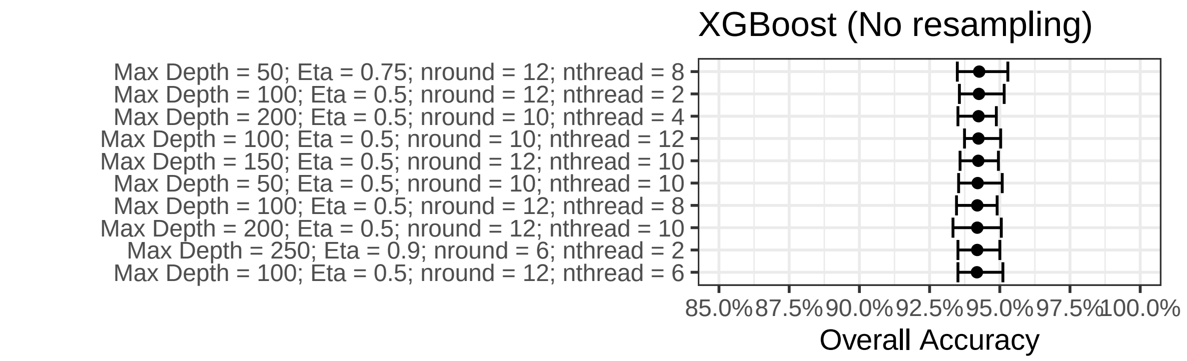
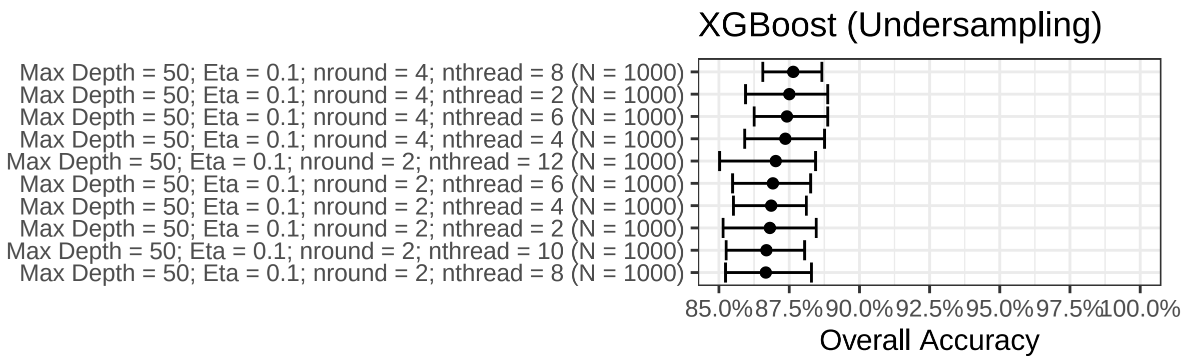
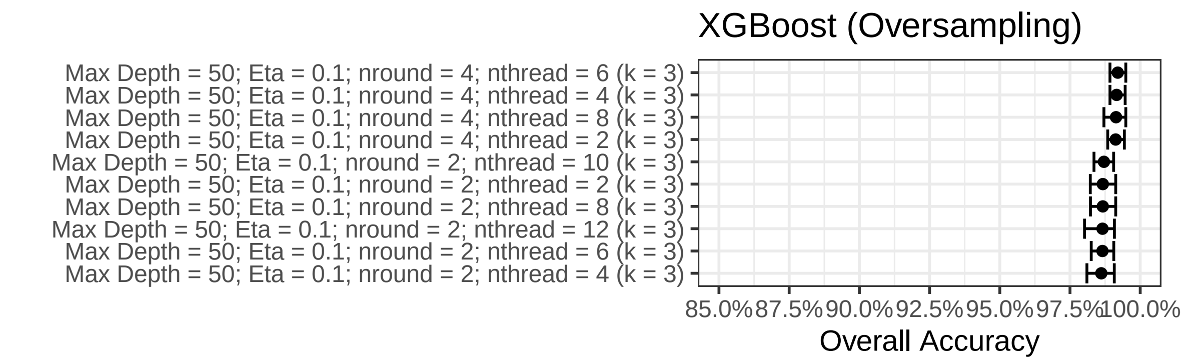
Classification Tuning

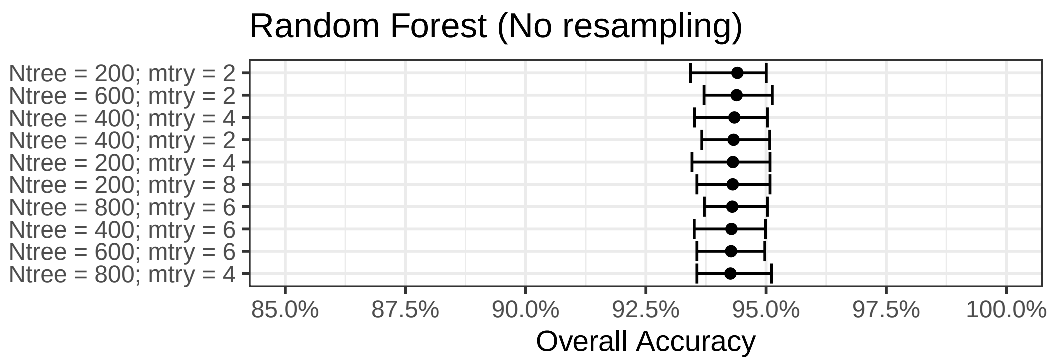
XGBoost

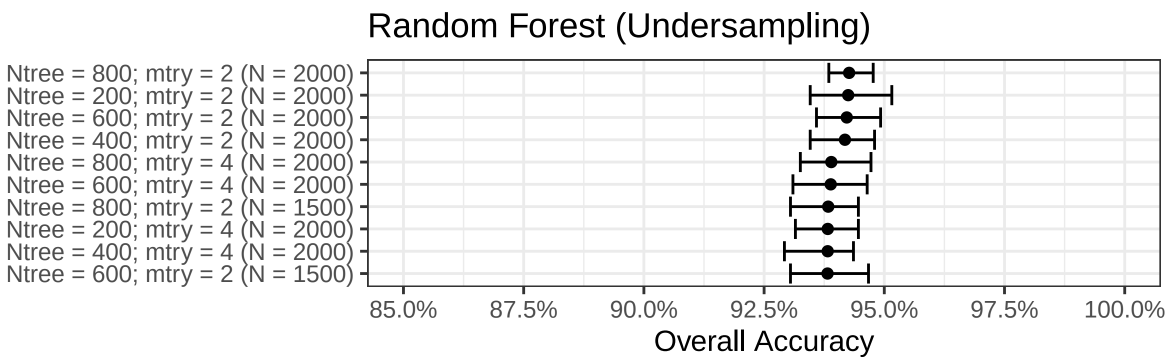


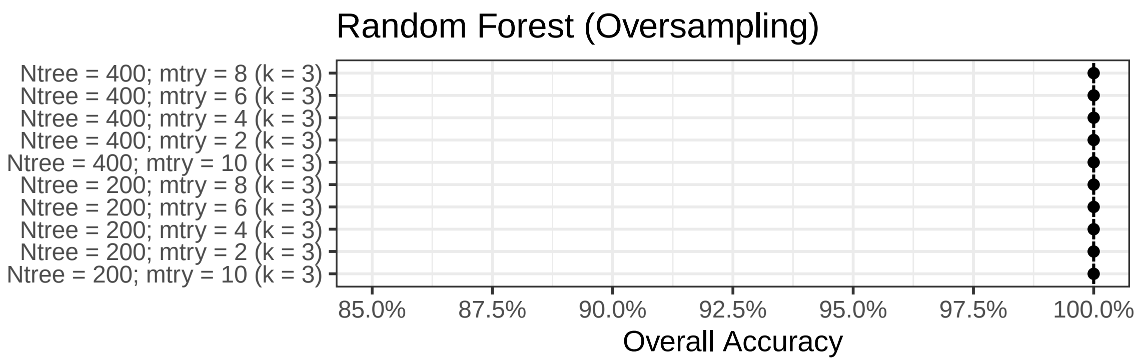




Random Forest







Code

```
# LOAD LIBRARIES
# ----------------------------------------------------------
library(tidyverse)
library(readr)
library(xgboost)
library(Matrix)
library(tictoc)
library(furrr)
library(smotefamily)

# IMPORT DATA, RELEVEL FACTOR COLUMNS
# ---------------------------------------
```

```r
winequality <- read_csv("data/winequality-all.csv") %>% mutate(type = factor(type,
    levels = c("red", "white")), type01 = as.numeric(ifelse(type ==
    "white", 0, 1)), qualityclass = factor(qualityclass, levels = c("Low",
    "Normal", "High")), under_class = ifelse(qualityclass ==
    "Normal", 0, 1), over_class = ifelse(qualityclass == "Low",
    0, ifelse(qualityclass == "Normal", 1, 2)))
colnames(winequality) <- make.names(names(winequality), unique = TRUE)
summary(winequality)

# RESAMPLING FUNCTIONS
# -----------------------------------------------------

# FUNCTION FOR UNDERSAMPLING
undersample <- function(train_df, nsample) {
    df_wine_0_ind <- which(train_df$under_class == 0)
    df_wine_1_ind <- which(train_df$under_class == 1)
    pick_0 <- sample(df_wine_0_ind, nsample)
    undersample_wine <- train_df[c(df_wine_1_ind, pick_0), ]  #Final Data frame
    undersample_wine <- undersample_wine %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)
    # table(undersample_wine$under_class) # have just to make
    # sure it's all balancing out how I think
    return(undersample_wine)
}

# FUNCTION FOR OVERSAMPLING
oversample <- function(train_df, k) {

    winequality_low <- filter(winequality, qualityclass %in%
        c("Low"))
    winequality_normal <- filter(winequality, qualityclass %in%
        c("Normal"))
    winequality_high <- filter(winequality, qualityclass %in%
        c("High"))

    wine <- sort(sample(nrow(winequality_normal), nrow(winequality_normal) *
        0.5))
    winequality_norm1 <- winequality_normal[wine, ]
    winequality_norm2 <- winequality_normal[-wine, ]

    wine_LN <- rbind(winequality_low, winequality_norm1)
    wine_HN <- rbind(winequality_high, winequality_norm2)
    SMOTEData1 <- SMOTE(wine_LN[, c("fixed.acidity", "volatile.acidity",
        "citric.acid", "residual.sugar", "chlorides", "free.sulfur.dioxide",
        "total.sulfur.dioxide", "density", "pH", "sulphates",
        "alcohol", "type01")], wine_LN[, "over_class"], K = k,
        dup_size = 0)
    SMOTEData2 <- SMOTE(wine_HN[, c("fixed.acidity", "volatile.acidity",
        "citric.acid", "residual.sugar", "chlorides", "free.sulfur.dioxide",
        "total.sulfur.dioxide", "density", "pH", "sulphates",
        "alcohol", "type01")], wine_HN[, "over_class"], K = k,
        dup_size = 0)
    oversample_df1 <- SMOTEData1$data  #Final data frame
    oversample_df2 <- SMOTEData2$data  #Final data frame
    oversample_df <- rbind(oversample_df1, oversample_df2) %>%
        mutate(type01 = round(type01)) %>% mutate(type = as.factor(ifelse(type01 ==
        0, "white", "red")), qualityclass = ifelse(class == 0,
        "Low", ifelse(class == "1", "Normal", "High"))) %>% mutate(qualityclass = factor(qualityclass,
        levels = c("Low", "Normal", "High"))) %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)
    # table(oversample_df$class) have just to make sure it's all
    # balancing out how I think
    return(oversample_df)
}

# XGBOOST FUNCTIONS
# -----------------------------------------------------------------

# SET UP FUNCTION TO EVALUATE XGBOOST
xgbFunc <- function(df = winequality, samplingMethod = "none",
    nUndersample = 2000, kOversample = 5, trainPct = 0.7, max.depth,
    eta, nround = 2, nthread = 2, show.table = F) {

    require(Matrix)
    require(xgboost)

    # set up training/testing sets
    n <- nrow(df)
    train.index <- sample(seq(1, n), floor(n * trainPct), replace = F)

    # create dgCMatrix for modeling

    # training
    train.data <- df[train.index, ]  # Normal

    if (samplingMethod == "undersample") {
        train.data <- undersample(train.data, nUndersample)  # Undersample
    }

    if (samplingMethod == "oversample") {
        train.data <- oversample(train.data, kOversample)  # Oversample
    }

    train.data <- train.data %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)

    train.datamatrix <- sparse.model.matrix(qualityclass ~ .,
        data = train.data)[, -1]
    train.qualityclass <- train.data$qualityclass
    train.label <- as.integer(train.data$qualityclass) - 1  # label conversion
    xgb.train <- list(data = train.datamatrix, label = train.label)

    # testing
    test.data <- df[-train.index, ] %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)
    test.datamatrix <- sparse.model.matrix(qualityclass ~ .,
        data = test.data)[, -1]
    test.qualityclass <- test.data$qualityclass
    test.label <- as.integer(test.data$qualityclass) - 1  # label conversion
    xgb.test <- list(data = test.datamatrix, label = test.label)


    # fit xgboost model
    xgb.fit <- xgboost(data = xgb.train$data, label = xgb.train$label,
        booster = "gbtree", max.depth = max.depth, eta = eta,
        nround = nround, nthread = nthread, objective = "multi:softprob",
        eval_metric = "merror", num_class = length(levels(train.qualityclass)),
        verbose = 0)

    # predict
    xgb.pred = predict(xgb.fit, xgb.test$data, reshape = T) %>%
        as.data.frame()
    colnames(xgb.pred) = levels(train.qualityclass)
    xgb.pred$prediction = apply(xgb.pred, 1, function(x) colnames(xgb.pred)[which.max(x)])
    xgb.pred$label = levels(train.qualityclass)[test.label +
        1]
    xgb.pred <- xgb.pred %>% mutate(prediction = factor(prediction,
        levels = c("Low", "Normal", "High")), label = factor(label,
        levels = c("Low", "Normal", "High")))

    # evaluated prediction
    accuracy.all <- mean(xgb.pred$prediction == xgb.pred$label)
    table <- with(xgb.pred, table(label, prediction))
    prop.table <- table/rowSums(table)
    accuracy.low <- prop.table[1, 1]
    accuracy.normal <- prop.table[2, 2]
    accuracy.high <- prop.table[3, 3]
    accuracy <- cbind(accuracy.all, accuracy.low, accuracy.normal,
        accuracy.high)

    if (show.table) {
        return(list(accuracy = accuracy, table = table, prop.table = prop.table))
    } else {
        return(accuracy)
    }

}

# XGBOOST MCMC FUNCTION WITH PARALLEL COMPUTING
xgbMCMC <- function(samplingMethod = "none", nUndersample = 2000,
    kOversample = 5, B = 5, trainPct = 0.7, max.depth, eta, nround = 2,
    nthread = 2) {
    require(furrr)

    # Create Parameter Grid
    mcmc.grid <- expand_grid(B = seq(1, B), samplingMethod = samplingMethod,
        nUndersample = nUndersample, kOversample = kOversample,
        trainPct = trainPct, max.depth = max.depth, eta = eta,
        nround = nround, nthread = nthread)

    # Obtain Accuracy
    accuracyList <- furrr::future_pmap(mcmc.grid[, -1], xgbFunc)
    xgbAccuracy <- matrix(unlist(accuracyList, use.names = TRUE),
        ncol = 4, nrow = nrow(mcmc.grid), byrow = T)
    colnames(xgbAccuracy) <- colnames(accuracyList[[1]])

    # Summarize Accuracy
    results <- cbind(mcmc.grid, xgbAccuracy) %>% pivot_longer(cols = c("accuracy.all",
        "accuracy.low", "accuracy.normal", "accuracy.high"),
        names_to = "accuracyGroup", values_to = "accuracy") %>%
        mutate(Method = "XGBoost") %>% dplyr::group_by(Method,
        accuracyGroup, samplingMethod, nUndersample, kOversample,
        max.depth, eta, nround, nthread) %>% summarise(B = n(),
        mean = mean(accuracy), lower = quantile(accuracy, probs = c(0.05)),
        upper = quantile(accuracy, probs = c(0.95))) %>% ungroup()

    return(results)

}

# RANDOM FOREST FUNCTIONS
# -------------------------------------------

# SET UP FUNCTION TO EVALUATE RANDOM FOREST
rfFunc <- function(df = winequality, samplingMethod = "none",
    nUndersample = 2000, kOversample = 5, trainPct = 0.7, importance = F,
    mtry = 4, ntree = 500, show.table = F) {

    require(randomForest)

    # set up training/testing sets
    n <- nrow(df)
    train.index <- sample(seq(1, n), floor(n * trainPct), replace = F)

    # training
    train.data <- df[train.index, ]  # Normal

    if (samplingMethod == "undersample") {
        train.data <- undersample(train.data, nUndersample)
    }

    if (samplingMethod == "oversample") {
        train.data <- oversample(train.data, kOversample)
    }
```

```r
    train.data <- train.data %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)

    # testing
    test.data <- df[-train.index, ] %>% dplyr::select(qualityclass,
        type, fixed.acidity, volatile.acidity, citric.acid, residual.sugar,
        chlorides, free.sulfur.dioxide, total.sulfur.dioxide,
        density, pH, sulphates, alcohol)

    # Fit that Random Forest!
    rf.fit <- randomForest(qualityclass ~ ., data = train.data,
        method = "class", ntree = ntree, mtry = mtry, importance = importance)

    # Get that Prediction!
    rf.pred = predict(rf.fit, newdata = test.data)

    # Evaluate Prediction
    accuracy.all <- mean(rf.pred == test.data$qualityclass)  #this is not working SJA
    table <- table(test.data$qualityclass, rf.pred)
    prop.table <- table/rowSums(table)
    accuracy.low <- prop.table[1, 1]
    accuracy.normal <- prop.table[2, 2]
    accuracy.high <- prop.table[3, 3]
    accuracy <- cbind(accuracy.all, accuracy.low, accuracy.normal,
        accuracy.high)

    if (show.table) {
        return(list(accuracy = accuracy, table = table, prop.table = prop.table))
    } else {
        return(accuracy)
    }

}

# RANDOM FOREST MCMC FUNCTION WITH PARALLEL COMPUTING
rfMCMC <- function(B = 5, samplingMethod = "none", nUndersample = 2000,
    kOversample = 5, trainPct = 0.7, ntree = 500, mtry = 4) {
    require(furrr)

    # Create Parameter Grid
    mcmc.grid <- expand_grid(B = seq(1, B), samplingMethod = samplingMethod,
        nUndersample = nUndersample, kOversample = kOversample,
        trainPct = trainPct, ntree = ntree, mtry = mtry)

    # Obtain Accuracy
    accuracyList <- furrr::future_pmap(mcmc.grid[, -1], rfFunc)
    rfAccuracy <- matrix(unlist(accuracyList, use.names = TRUE),
        ncol = 4, nrow = nrow(mcmc.grid), byrow = T)
    colnames(rfAccuracy) <- colnames(accuracyList[[1]])

    # Summarize Accuracy
    results <- cbind(mcmc.grid, rfAccuracy) %>% pivot_longer(cols = c("accuracy.all",
        "accuracy.low", "accuracy.normal", "accuracy.high"),
        names_to = "accuracyGroup", values_to = "accuracy") %>%
        mutate(Method = "Random Forest") %>% dplyr::group_by(Method,
        accuracyGroup, samplingMethod, nUndersample, kOversample,
        ntree, mtry) %>% summarise(B = n(), mean = mean(accuracy),
        lower = quantile(accuracy, probs = c(0.05)), upper = quantile(accuracy,
            probs = c(0.95))) %>% ungroup()

    return(results)

}


# HYPERPARAMETER GRID SEARCH
# ------------------------------------------
setB = 50

# XGBOOST

tic()
xgbMCMC.none.gridsearch <- xgbMCMC(samplingMethod = "none", nUndersample = NA,
    kOversample = NA, trainPct = 0.7, B = setB, max.depth = seq(50,
        250, 50), eta = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95),
    nround = seq(2, 12, 2), nthread = seq(2, 12, 2))
toc()

xgbMCMC.none.gridsearch <- xgbMCMC.none.gridsearch %>% filter(accuracyGroup ==
    "accuracy.all") %>% arrange(-mean)
xgbMCMC.none.gridsearch
# write.csv(xgbMCMC.none.gridsearch, file =
# 'reports/xgbMCMC.none.gridsearch.csv', row.names = F, na =
# '')

xgbMCMC.none.gridsearch.plot <- xgbMCMC.none.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Max Depth = ", max.depth, "; Eta = ",
    eta, "; nround = ", nround, "; nthread = ", nthread, sep = "")) %>%
    ggplot(aes(x = mean, y = reorder(label, mean))) + geom_point() +
    geom_errorbar(aes(xmin = lower, xmax = upper)) + theme_bw() +
    scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("XGBoost (No resampling)")
xgbMCMC.none.gridsearch.plot

tic()
xgbMCMC.undersample.gridsearch <- xgbMCMC(samplingMethod = "undersample",
    nUndersample = seq(1000, 2000, 500), kOversample = NA, trainPct = 0.7,
    B = setB, max.depth = seq(50, 250, 50), eta = c(0.5, 0.1,
        0.25, 0.5, 0.75, 0.9, 0.95), nround = seq(2, 12, 2),
    nthread = seq(2, 12, 2))
toc()
xgbMCMC.undersample.gridsearch <- xgbMCMC.undersample.gridsearch %>%
    select(accuracyGroup == "accuracy.all") %>% arrange(-mean)
xgbMCMC.undersample.gridsearch
```

```r
# write.csv(xgbMCMC.undersample.gridsearch, file =
# 'reports/xgbMCMC.undersample.gridsearch.csv', row.names =
# F, na = '')

xgbMCMC.undersample.gridsearch.plot <- xgbMCMC.undersample.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Max Depth = ", max.depth, "; Eta = ",
    eta, "; nround = ", nround, "; nthread = ", nthread, " (N = ",
    nundersample, ")", sep = ""))
ggplot(aes(x = mean, y = reorder(label, mean))) + geom_point() +
    geom_errorbar(aes(xmin = lower, xmax = upper)) + theme_bw() +
    scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("XGBoost (Undersampling)")
xgbMCMC.undersample.gridsearch.plot

tic()
xgbMCMC.oversample.gridsearch <- xgbMCMC(samplingMethod = "oversample",
    nUndersample = NA, kOversample = seq(3, 7, 2), trainPct = 0.7,
    B = setB, max.depth = seq(50, 250, 50), eta = c(0.5, 0.1,
        0.25, 0.5, 0.75, 0.9, 0.95), nround = seq(2, 12, 2),
    nthread = seq(2, 12, 2))
toc()
xgbMCMC.oversample.gridsearch <- xgbMCMC.oversample.gridsearch %>%
    select(accuracyGroup == "accuracy.all") %>% arrange(-mean)
xgbMCMC.oversample.gridsearch
# write.csv(xgbMCMC.oversample.gridsearch, file =
# 'reports/xgbMCMC.oversample.gridsearch.csv', row.names = F,
# na = '')

xgbMCMC.oversample.gridsearch.plot <- xgbMCMC.oversample.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Max Depth = ", max.depth, "; Eta = ",
    eta, "; nround = ", nround, "; nthread = ", nthread, " (k = ",
    kOversample, ")", sep = ""))
ggplot(aes(x = mean, y = reorder(label, mean))) + geom_point() +
    geom_errorbar(aes(xmin = lower, xmax = upper)) + theme_bw() +
    scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("XGBoost (Oversampling)")
xgbMCMC.oversample.gridsearch.plot

# RANDOM FOREST

# No resampling
tic()
rfMCMC.none.gridsearch <- rfMCMC(samplingMethod = "none", nUndersample = NA,
    kOversample = NA, trainPct = 0.7, B = setB, ntree = seq(200,
        800, 200), mtry = seq(2, 10, 2)  # Default should be 4
)
toc()

rfMCMC.none.gridsearch <- rfMCMC.none.gridsearch %>% filter(accuracyGroup ==
    "accuracy.all") %>% arrange(-mean)
rfMCMC.none.gridsearch
# write.csv(rfMCMC.none.gridsearch, file =
# 'reports/rfMCMC.none.gridsearch.csv', row.names = F, na =
# '')

rfMCMC.none.gridsearch.plot <- rfMCMC.none.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Ntree = ", ntree, "; mtry = ",
    mtry, sep = "")) %>% ggplot(aes(x = mean, y = reorder(label,
    mean))) + geom_point() + geom_errorbar(aes(xmin = lower,
    xmax = upper)) + theme_bw() + scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("Random Forest (No resampling)")
rfMCMC.none.gridsearch.plot

# Undersample
tic()
rfMCMC.undersample.gridsearch <- rfMCMC(samplingMethod = "undersample",
    nUndersample = seq(1000, 2000, 500), kOversample = NA, trainPct = 0.7,
    B = setB, ntree = seq(200, 800, 200), mtry = seq(2, 10, 2)  # Default should be 4
)
toc()

rfMCMC.undersample.gridsearch <- rfMCMC.undersample.gridsearch %>%
    filter(accuracyGroup == "accuracy.all") %>% arrange(-mean)
rfMCMC.undersample.gridsearch
# write.csv(rfMCMC.undersample.gridsearch, file =
# 'reports/rfMCMC.undersample.gridsearch.csv', row.names = F,
# na = '')

rfMCMC.undersample.gridsearch.plot <- rfMCMC.undersample.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Ntree = ", ntree, "; mtry = ",
    mtry, " (N = ", nUndersample, ")", sep = "")) %>% ggplot(aes(x = mean,
    y = reorder(label, mean))) + geom_point() + geom_errorbar(aes(xmin = lower,
    xmax = upper)) + theme_bw() + scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("Random Forest (Undersampling)")
rfMCMC.undersample.gridsearch.plot

# Oversample
tic()
rfMCMC.oversample.gridsearch <- rfMCMC(samplingMethod = "oversample",
    nUndersample = NA, kOversample = seq(3, 7, 2), trainPct = 0.7,
    B = setB, ntree = seq(200, 800, 200), mtry = seq(2, 10, 2)  # Default should be 4
)
toc()

rfMCMC.oversample.gridsearch <- rfMCMC.oversample.gridsearch %>%
    filter(accuracyGroup == "accuracy.all") %>% arrange(-mean)
rfMCMC.oversample.gridsearch
# write.csv(rfMCMC.oversample.gridsearch, file =
# 'reports/rfMCMC.oversample.gridsearch.csv', row.names = F,
# na = '')

rfMCMC.oversample.gridsearch.plot <- rfMCMC.oversample.gridsearch[c(0:10),
    ] %>% mutate(label = paste("Ntree = ", ntree, "; mtry = ",
    mtry, " (k = ", kOversample, ")", sep = "")) %>% ggplot(aes(x = mean,
    y = reorder(label, mean))) + geom_point() + geom_errorbar(aes(xmin = lower,
    xmax = upper)) + theme_bw() + scale_y_discrete("") + scale_x_continuous("Overall Accuracy",
    labels = scales::percent) + ggtitle("Random Forest (Oversampling)")
rfMCMC.oversample.gridsearch.plot

# FINAL RESULTS
# ----------------------------------------------

# XGBOOST MCMC RESULTS
setB = 100

tic()
xgbMCMC.none.results <- xgbMCMC(samplingMethod = "none", nUndersample = NA,
    kOversample = NA, trainPct = 0.7, B = setB, max.depth = 50,
    eta = 0.75, nround = 12, nthread = 8)
toc()

tic()
xgbMCMC.undersample.results <- xgbMCMC(samplingMethod = "undersample",
    nUndersample = 1000, kOversample = NA, trainPct = 0.7, B = setB,
    max.depth = 50, eta = 0.1, nround = 2, nthread = 2)
toc()

tic()
xgbMCMC.oversample.results <- xgbMCMC(samplingMethod = "oversample",
    nUndersample = NA, kOversample = 3, trainPct = 0.7, B = setB,
    max.depth = 50, eta = 0.1, nround = 2, nthread = 2)
toc()


xgbMCMC.results <- rbind(xgbMCMC.none.results, xgbMCMC.undersample.results,
    xgbMCMC.oversample.results) %>% mutate(label = paste("Max Depth = ",
    max.depth, "; Eta = ", eta, sep = "")) %>% select(-max.depth,
    -eta, -nround, -nthread)


# RANDOM FORST MCMC RESULTS
tic()
rfMCMC.none.results <- rfMCMC(B = setB, samplingMethod = "none",
    nUndersample = NA, kOversample = NA, trainPct = 0.7, ntree = 200,
    mtry = 2)
toc()

tic()
rfMCMC.undersample.results <- rfMCMC(B = setB, samplingMethod = "undersample",
    nUndersample = 2000, kOversample = NA, trainPct = 0.7, ntree = 800,
    mtry = 2)
toc()

tic()
rfMCMC.oversample.results <- rfMCMC(B = setB, samplingMethod = "oversample",
    nUndersample = NA, kOversample = 3, trainPct = 0.7, ntree = 200,
    mtry = 3)
toc()
```

```
rfMCMC.results <- rbind(rfMCMC.none.results, rfMCMC.undersample.results,
    rfMCMC.oversample.results) %>% mutate(label = paste("N Trees = ",
    ntree, "; Mtry = ", mtry, sep = "")) %>% select(-ntree, -mtry)

# COMBINE XGBOOST AND RANDOM FOREST RESULTS & PLOT
MCMC.results <- rbind(xgbMCMC.results, rfMCMC.results)

write.csv(MCMC.results, "reports/MCMC.results.csv", row.names = F,
    na = "")
MCMC.results <- read_csv("reports/MCMC.results.csv")

# new facet labels
accuracyGroup.labs <- c("Overall", "Low Quality", "Normal Quality",
    "High Quality")
names(accuracyGroup.labs) <- c("accuracy.all", "accuracy.low",
    "accuracy.normal", "accuracy.high")

# plot accuracy
MCMC.results %>% mutate(samplingMethod = factor(samplingMethod,
    levels = c("none", "undersample", "oversample")), accuracyGroup = factor(accuracyGroup,
    levels = c("accuracy.all", "accuracy.low", "accuracy.normal",
        "accuracy.high"))) %>% ggplot(aes(x = mean, y = samplingMethod,
    group = Method, color = Method)) + geom_point(size = 1, position = position_dodge(width = 0.5)) +
    geom_errorbar(aes(xmin = lower, xmax = upper), position = position_dodge(width = 0.5),
        width = 0.4) + facet_wrap(~accuracyGroup, ncol = 1, labeller = labeller(accuracyGroup = accurac
    theme_bw() + theme(aspect.ratio = 0.2) + scale_y_discrete("Resampling Method") +
    scale_x_continuous("Accuracy", limits = c(0, 1), breaks = seq(0,
        1, 0.2), labels = scales::percent) + scale_color_brewer("",
    palette = "Paired")
```