Classification and Resampling of Wine Quality on Imbalanced Data

by Alison Kleffner, Sarah Aurit, Emily Robinson

April 17, 2021

1 Introduction

Successful marketing campaigns and productive selling strategies are directly linked to communication about key indicators of quality; hence, objective measurements of quality are essential. Within the wine industry, there are two types of quality assessment: physiochemical and sensory tests. Sensory tests require a human expert to assess the quality of wine based on visual, taste, and smell [Hu et al., 2016]. Hiring human experts to conduct sensory tests can take time and be expensive [Gupta, 2018]. In addition, taste is the least understood of all human senses [Cortez et al., 2009]. Unlike sensory tests, laboratory tests for measuring the physiochemical characteristics of wine such as acidity and alcohol content do not require a human expert. The relationship between physiochemcial and sensory analysis is not well understood. Recently, research in the food industry has utilized statistical learning techniques to evaluate widely available characteristics of wine. This type of evaluation allows the automation of quality assessment processes by minimizing the need of human experts [Gupta, 2018]. These techniques also have the advantage of identifying important the phsiochemical characteristics that have an impact on the quality of wine as determined by a sensory test.

The goal of this paper is to train a model that would work well to classify wines into three categories, which are: poor quality, normal quality and high quality. It is desirable to classify wines using physicochemical properties since this does not involve human bias that would come into play with human tasters. We evaluated two classification techniques, eXtreme Boosting (XGBoost) and Random Forest. Given prior investigation of the white wine data showed the Random Forests technique performed well; we would like to test this method using both white and red wine. Previous papers on the wine data set have also applied different versions of gradient boosting such as adaptive boost; we will apply XGBoost, an alternative gradient boosting technique. The quality categorization poses a challenge of working with imbalanced classes as there are many more normal quality wines than low or high quality wines [Figure 1]. To address this, we will evaluate different resampling techniques in order to determine if resampling improves performance and to identify which resampling method is best. Accuracy of each classification model applied in conjunction with each resampling method was compared. Initially, we will first evaluated the classifier performance with no resampling. Since the data is very imbalanced, we hypothesized this to have low performance but provide a baseline. We will then apply a resampling method using SMOTE, which is an algorithm where the minority class (in this case low and high quality), will be oversampled. We hypothesize this method runs the risk of overfitting the mode. The final resampling method applied is a random under sampler, where majority class data are randomly removed from the data set. We hypothesize this risks losing valuable information in our model. Accuracy of each classification method and resampling technique was evaluated by the overall correct classification rate and each individual group (low quality, normal quality, and high quality) correct classification rate.

2 Data Exploration

Sarah put in table and explanation in this section?

2.1 About the Data

In this paper, we apply classification and resampling techniques to the "Wine Quality Data Set" found on the UCI Data Repository [UCI]. The data is made up of vinho verde, which is a product from the northwest region of Portugal. The data was collected from May 2004 to February of 2007, and is made up of 1599 red wines and 4898 white wines for a total of 6493 observations [Cortez et al., 2009]. For each of the wines in the dataset, 11 physiochemical variables were taken on it: fxed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol. Additionally, in our paper, we classified the combined red and white wine data sets, so included was a 12th predictor variable categorizing the given observation as red or white wine. Our response was a quality rating based on a sensory test carried out by at least three sommeliers, where a 0 was considered very bad and a 10 was excellent Gupta [2018]. Following Hu et al. [2016], we separated the wine into three classes: Low Quality (≤ 4), Normal (5-7), and High quality (≥ 8). These response values are imbalanced as can be seen in Figure 1, with most of the wines having a response of "Normal."

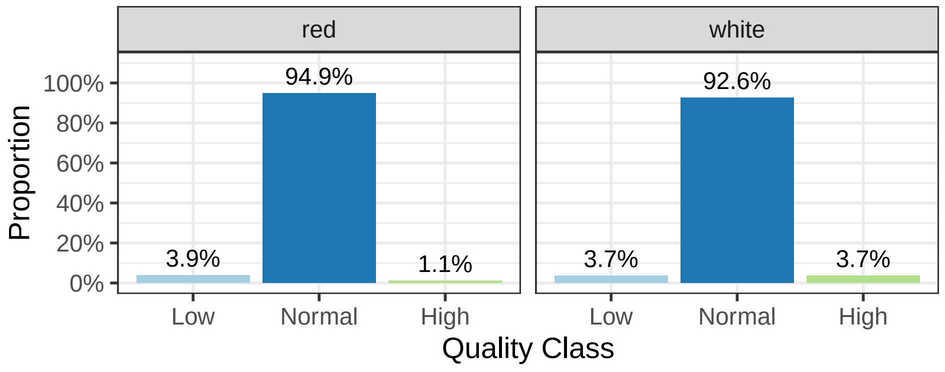


Figure 1: Wine quality class imbalance.

2.2 Exploring the Data

In Figure 2, boxplots of the 11 physiochemical variables are given. As can be see in this plot, for most of the predictor variables there are points that would be considered outliers, which are represented by black dots. We did not consider removing outliers, so no data points were removed for our final analysis. Looking at the boxplots, one can use these to give an idea of which of these predictor variables may be helpful in helping to determine the classification of the wines. For example, for alcohol, the Interquartile of the High category is above that of the Normal and Low class. Figure 3, gives the correlation plot for the 11 physiochemical variables, with also the 0-10 scale for the quality of the wines. Correlations that are not considered to be statistically significant are not shown. Most of the correlations seem to be on the lower end of the spectrum, so multicollinearity does not seem to be a huge issue that needs to be addressed among the predictor variables.It also shows that all of the predictor variables have a statistically significant correlation with the response of quality, so they should be helpful for classification.

3 Statistical Methods

We applied two classification methods, eXtreme Gradient Boositng (XGBoost) and Random Forest, on the original training set (no resampling), the undersampled training set, and the oversampled training set. Accuracy of each classification method and resampling technique was evaluated by the overall correct classification rate and each individual group (low quality, normal quality, and high quality) correct classification rate. Monte Carlo Cross-Validation (MCMC) was used to select tuning parameters and evaluate model performance. Classification rate is

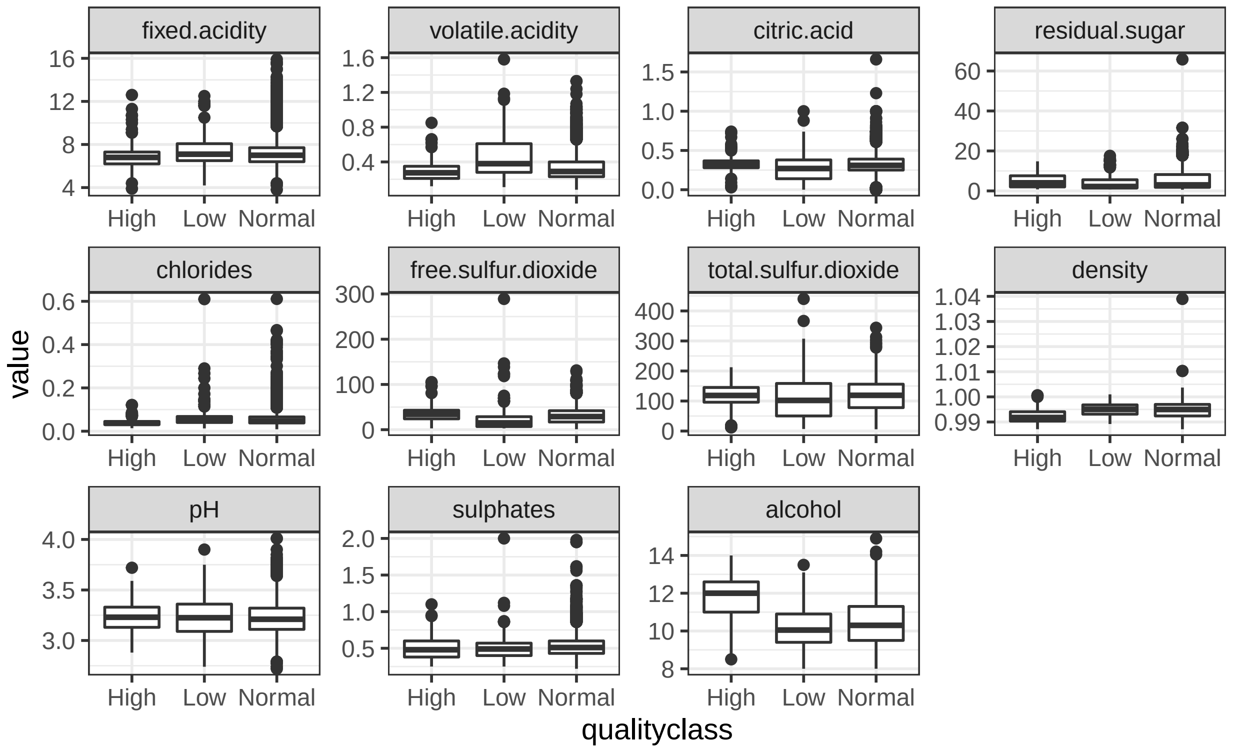
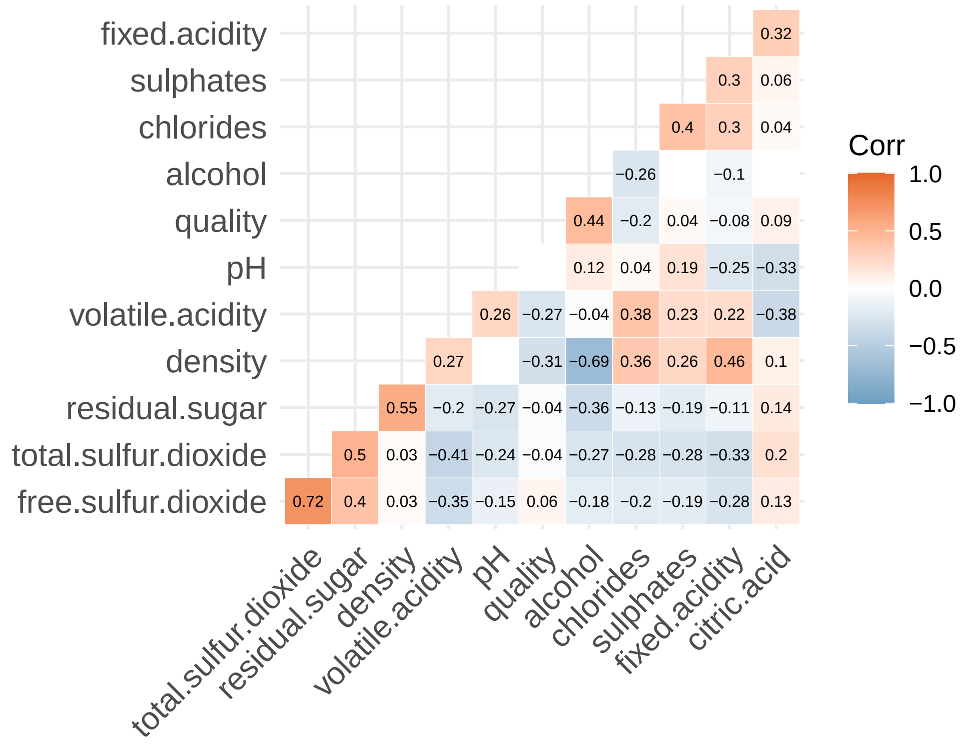


Figure 2: Distribution of the Predictor Variables



	total.sulfur.dioxide	residual.sugar	density	volatile.acidity	pH	quality	alcohol	chlorides	sulphates	fixed.acidity	citric.acid
fixed.acidity											0.32
sulphates										0.3	0.06
chlorides									0.4	0.3	0.04
alcohol								−0.26		−0.1	
quality							0.44	−0.2	0.04	−0.08	0.09
pH							0.12	0.04	0.19	−0.25	−0.33
volatile.acidity					0.26	−0.27	−0.04	0.38	0.23	0.22	−0.38
density				0.27		−0.31	−0.69	0.36	0.26	0.46	0.1
residual.sugar			0.55	−0.2	−0.27	−0.04	−0.36	−0.13	−0.19	−0.11	0.14
total.sulfur.dioxide		0.5	0.03	−0.41	−0.24	−0.04	−0.27	−0.28	−0.28	−0.33	0.2
free.sulfur.dioxide	0.72	0.4	0.03	−0.35	−0.15	0.06	−0.18	−0.2	−0.19	−0.28	0.13

Figure 3: Correlation Plot

calculated as:

$$\text{overall accuracy} = \frac{1}{n}\sum_{i=1}^{n} I_{[\widehat{class_i}=class_i]}$$

$$\text{accuracy for group k} = \frac{1}{n_k}\sum_{j=1}^{n_k} I_{[\widehat{class_j}=class_j]}.$$

The MCMC algorithm repeatedly splits the data into training and testing sets (B = 50) by randomly selecting the designated proportions (70% training; 30% testing) of the overall data set to each. The undersampling and oversampling techniques described below are then applied to the training set to obtain a final resampled training data set. For each split, the final resampled training data set is used to build the model and then accuracy is evaluated on the corresponding testing data set. The mean, 5th-quantile, and 95th-quantile of the correct classification rates are used to evaluate performance over the B splits. Both classification methods were tuned by conducting a hyperparameter grid search with MCMC to minimize the overall accuracy [Appendix A] and parallel computing through the furrr package in R was conducted to minimize computing time [Vaughan and Dancho, 2021].

3.1 Resampling Techniques

Before we are able to consider the different classification methods, the imbalanced issue in our data must be examined. This is because typical classifier algorithms assume a relatively balanced distribution, so with imbalanced data they tend to be biased towards the majority class [Yanminsun, 2011]. Classification rules for the minority classes tend to be undiscovered or ignored, so the minority class is misclassified more often than the majority class. If we were to look at our data set from a binary standpoint where “Low” and “High” were classified as rare, and “Normal” was classified as not rare, the majority class has 6053 observations, and the minority class has 444 observations, which is a ratio of about 13:1. So due to this imbalanced nature in the data, classifying algorithms will be biased towards classifying wines as Normal, and causing poor predictions for the Low and High classes. When creating the resampled data sets, the binary classification of “rare” and “not rare” is used due to the resampling algorithms needing a binary class.

A method on how to deal with the imbalanced issue in the data is through resampling the original data set either by oversampling the majority class, or undersampling the minority class. The goal with these methods is to create a data set that has close to a balanced class distribution, so the classifying algorithms will have better predictions, so it will predict to the minority class more accurately [Chawla, 2013]. The oversampling method involves replicating the minority class and creating synthetic data, or creating new instances using heuristics. This method tends to have an overfitting problem, where it’ll predict the minority class more than it should. The undersampling method removes instances from the majority class randomly or using some heuristics. Since data is being removed, potentially valuable information is not considered [Hu et al, 2016]. To see how valuable resampling is in our data set, we ran the classifying algorithms with just using the original imbalanced data set, randomly undersampling method, and SMOTE, which is an oversampling method.

The undersampling method that we considered is random undersampling. In this method, instances of the majority class are discarded at random until reaches balanced with the minority class [Chawla, 2013]. For example, say there are 1000 observations in the majority class, and 100 in the minority class, observations in the majority class will be randomly discarded until this class is also 100. The benefit of this method is that since the data frame is being reduced, it is less costly. However, potentially useful information is discarded, which may make the decision boundary between the majority and minority class less clear, causing a decrease in prediction performance [Chawla, 2013].

SMOTE, which stands for Synthetic Minority Over-Sampling Technique, is an oversampling technique which uses interpolation of the minority class to create synthetic data. The process begins by finding the k nearest neighbors of each observation of the minority class based on some distance measure. Then a point between the minority class observation and one of its nearest neighbors is randomly picked by first finding the difference between the observation and its nearest neighbor. This difference is then multiplied by a random number between 0 and 1. This is then added to the observation, which becomes the new synthetic data point that is then added to the data set [Chawla, 2002]. In Hu et al, the used k=5, so that is what we used to create our resampled data set [2016]. Oversampling tend to have an overfitting problem since now the minority class extends into the majority space, however this generally poses less of an issue with SMOTE [Luego, 2010]. This was ran using the SMOTE function in the smotefamily package in R.

3.2 Classification Methods

We evaluated two classification techniques, which included Random Forest and eXtreme Boosting (XGBoost) methodology, both tree based ensemble methods. Prior investigation of the white wine data showed the Random Forests technique performed well; we wanted to test this method using both white and red wine. Previous papers on the wine data set have also applied different versions of gradient boosting such as adaptive boost; we utilized an alternative gradient boosting technique XGBoost.

The first classficiation technique utilized the Random Forest algorithm and the associated selection of independent variables for the Random Forest with the randomForest and tuneRF functions. In general, a Random Forest algorithm involves the creation of many classification trees, which are each grown through recursive partioning of independent variable space. During the growth phase for each classification tree, split points of variable space are investigated to maximize the reduction of heterogeneity, which is measured with the Gini impurity index (a reflection of the probability of a variable being wrongly chosen through a stochastic process). Each tree is then grown to a phase where node(s) represent a delineation of data per split point of each variable, and leaves are the terminal points that represenet the final classification of data. After the growth phase, a leaf is potentially pruned back to a node to maximize the tradeoff between complexity and rate of misclassification. A random forest incorporates many trees and subsequently aggregates the predictions made by each tree. We initially selected 500 trees and four variables that could be randomly sampled from to generate a split point at each node. Finally, a stepwise function was created to iteratively conduct the Random Forest algorithm for the purposes of selection of the optimal number of variables to sample from.

We also applied eXtreme Gradient Boosting (XGBoost) using the `xgboost` function with the multiclass classification using the softmax objective function with three classes maximizing the multiclass misclassfication error evaluation metric in the xgboost library [Chen et al., 2021]. This method combines the tree-based model approach by implementing recursive partitioning and the boosting algorithm which repeatedly optimizes classification methods on the training set. In repeated optimization, a weak classifier is fit on the original data set with each observation having equal weight, the weight is then calculated for the current model based on the error rate and observations are assigned new weights used to fit the next weak classifier. This process is repeated for a final boosted classifier given by the weighted sum of our weak classifiers. XGBoost allows for a variety of evaluation metrics providing a benefit over other boosting methods. Tuning parameters for maximum tree depth, step size shrinkage to prevent overfitting (eta), maximum number of boosting iterations, and number of threads used for parallel computing were selected by conducting a hyperparameter grid search with MCMC [Appendix A.1].

4 Results

Go back and check these classification rates.

The hyperparameter grid search led to final best fit models for Random Forest and XGBoost with tuning parameters shown in Tables 1 and 2 respectively. A comparison of the best classifiers is shown in Figure 4. With no resampling, the overall classification rate for Random Forest was between 93.7% and 95% while the rare classes had classification rates of 12% and 33% for low and high quality respectively. The XGBoost has a slight sacrifice in accruacy with a 91.7% to 93.4% overalll classification rate only 9.3% and 7.7% classification rates for the low and high quality classes. When undersampling on the training data set is conducted, we see a slight decrease in overall accuracy with Random Forest perfomring at a 93.2% to 94.8% classification rate and XGBoost performing at a 90.1% to 92.4% classification rate. We see this decrease in the normal quality classificaiton rate, but an increase in the rare quality classification rates. The low quality accuracy for Random forest increased to 20.5% while the high quality accuracy remained similar to that of no resampling at 34.8%. With the XGBoost classifier, undersampling increased the low quality accuracy to 21.3% and high quality accuracy to 26.8%. We saw imense benefits of oversampling the training data set reaching a 100% classification rate of all quality classes with the Random Forest classifier. This gain in accuracy from oversampling was also seen in the XGBoost classifier with overall classification rates between 97.3% to 98.4% and the rare classes reaching 95.2% and 95.5% classification rates for the low and high quality classes.

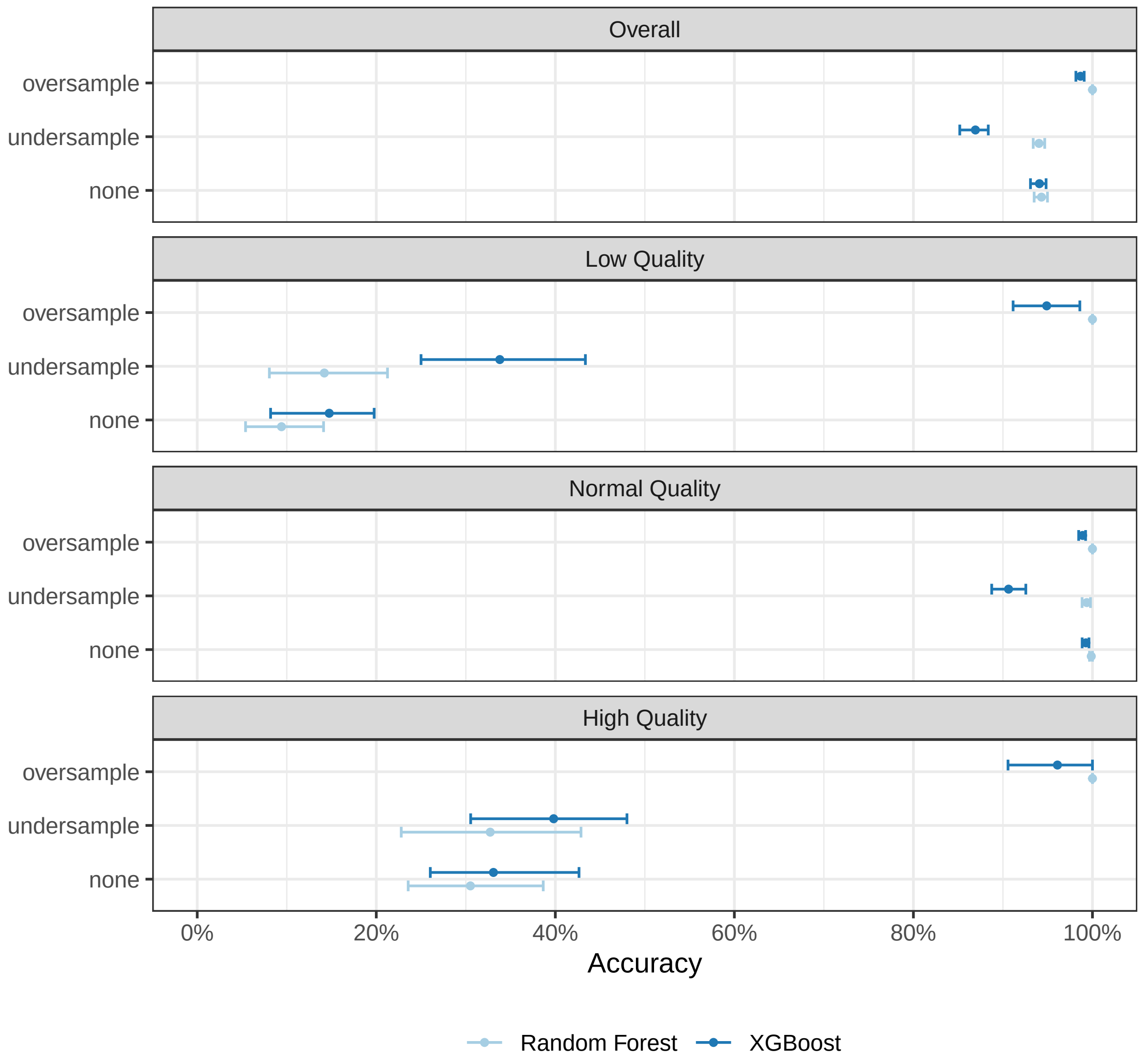


Figure 4: MCMC accuracy results for each classifier applied to each resampling technique.

Table 1: Final tuning parameters for Random Forest classifier as determined by the MCMC hyperparameter grid search.

Resampling	N. Trees	Mtry
No Resampling	200	2
Undersampling (n = 2000)	800	2
Oversampling (k = 3)	200	2

Table 2: Final tuning parameters for XGBoost classifier as determined by the MCMC hyperparameter grid search.

Resampling	Max Depth	Eta	Rounds	Threads
No Resampling	50	0.75	12	8
Undersampling (n = 1000)	50	0.10	2	2
Oversampling (k = 3)	50	0.10	2	2

5 Discussion and Conclusion

Add discusssion about the project here. Beginning a bulletted list of possible discussion points.

- Random Forest rocks!
- WOW, look at that oversampling method!
- Undersampling really helped increase the classification rate for rare classes.
- Tuning of XGBoost is not fun.

Supplementary Material

- **Data:** Data used was from the UCI Data Repository [UCI].
- **Code:** Access the final analysis code on GitHub or in Appendix B.

Course Reflection

Discuss overall course lessons in this section. Beginning a bulleted list of possible discussion topics.

- Importance of cross validation.
- Interpretation and communication is key!
- Overall great job describing the methods in an understandable way, maybe a little more coding done in class? The homework was helpful for this!
- Instructor was well prepared for class and provided timely feedback.
- Overall really enjoyed the course!

References

Uci machine learning repository: Wine quality data set. URL https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, and Y. Li. *xgboost: Extreme Gradient Boosting*, 2021. URL https://CRAN.R-project.org/package=xgboost. R package version 1.3.2.1.

P. Cortez, J. Teixeira, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Using data mining for wine quality assessment. In *International Conference on Discovery Science*, pages 66–79. Springer, 2009.

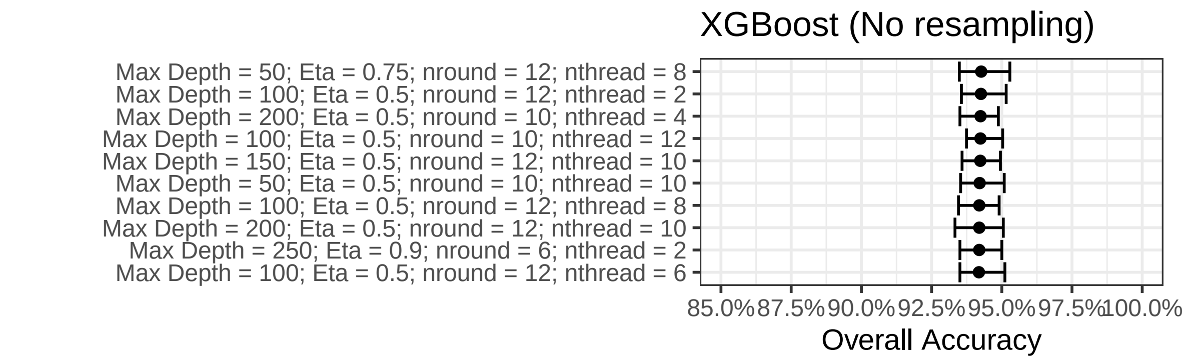
Y. Gupta. Selection of important features and predicting wine quality using machine learning techniques. *Procedia Computer Science*, 125:305–312, 2018.

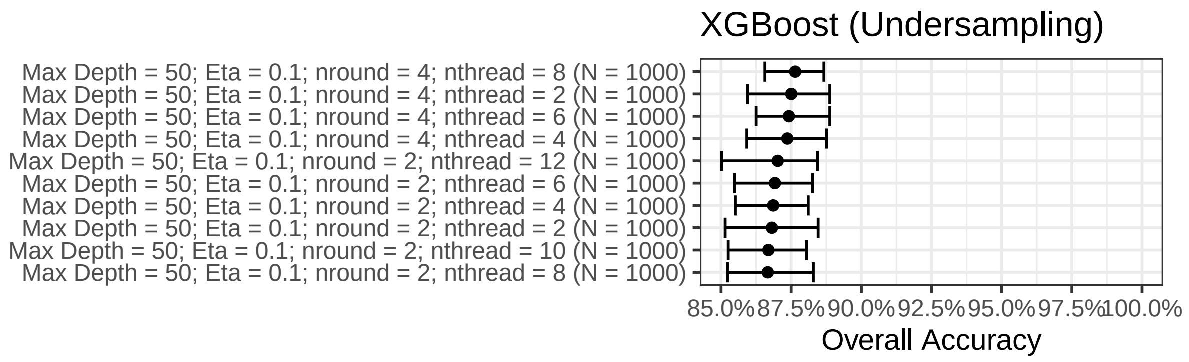
G. Hu, T. Xi, F. Mohammed, and H. Miao. Classification of wine quality with imbalanced data. In *2016 IEEE International Conference on Industrial Technology (ICIT)*, pages 1712–1217. IEEE, 2016.

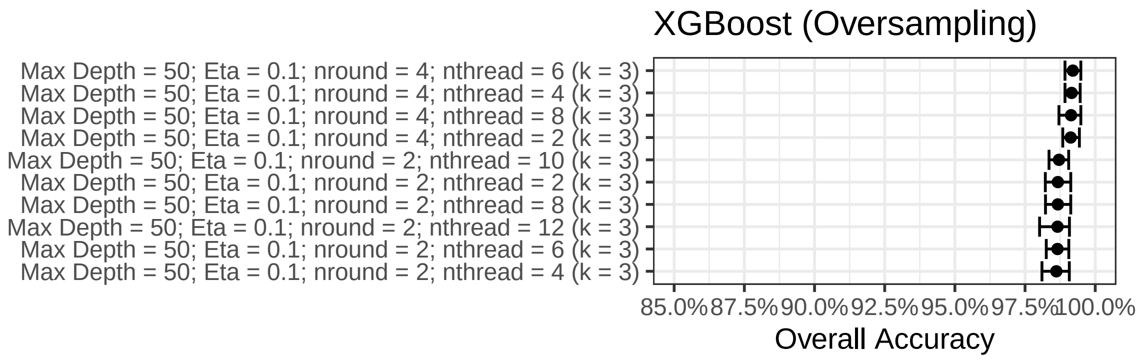
D. Vaughan and M. Dancho. *furrr: Apply Mapping Functions in Parallel using Futures*, 2021. URL https://CRAN.R-project.org/package=furrr. R package version 0.2.2.

A Classification Tuning

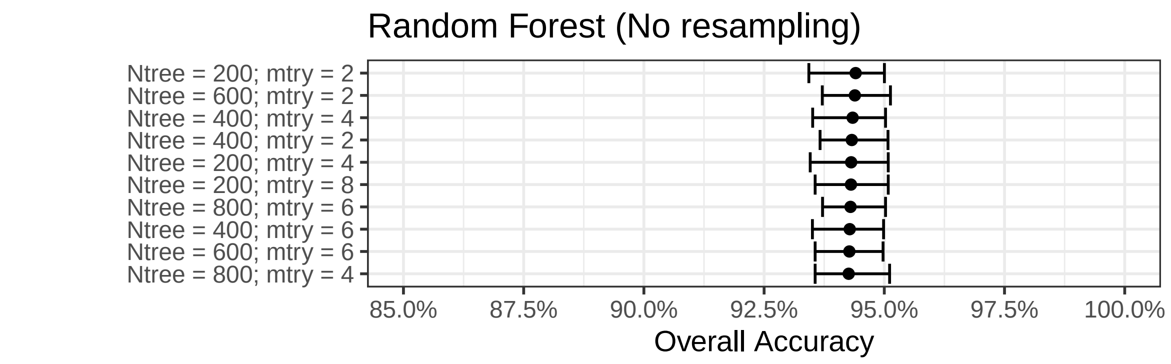
A.1 XGBoost

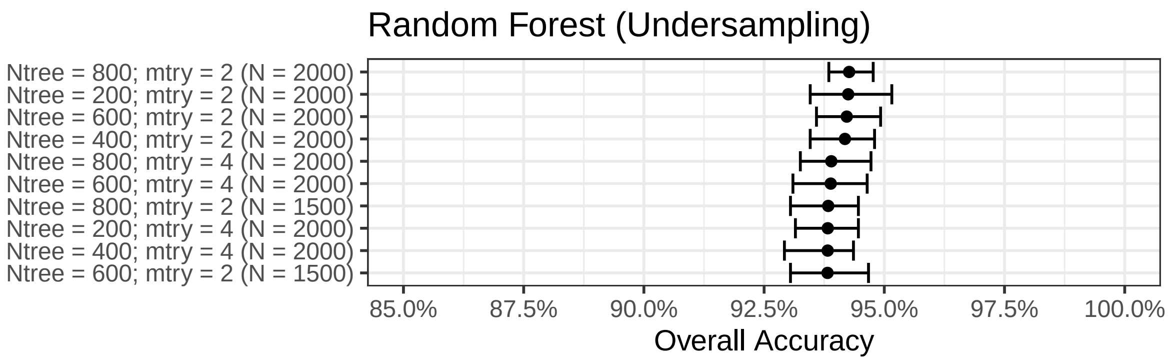


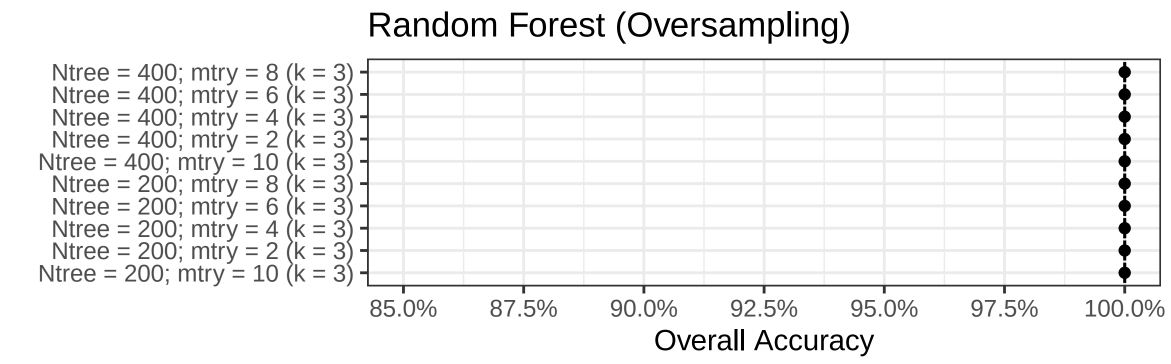




A.2 Random Forest







B 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
```

```r
    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 = accuracyGro
    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")
```