Classification and Resampling of Wine Quality on Imbalanced Data Statistical Learning (STAT 983)

Alison Kleffner, akleffner@huskers.unl.edu
Sarah Aurit, SarahAurit@creighton.edu
Emily Robinson, emily.robinson@huskers.unl.edu

April 20, 2021

Abstract

Purpose: The ability to assess the quality of wine is essential in the wine industry in order to market and sell a wine. Currently quality assessment is done by the use of human experts, which is costly and time consuming. We examined, through the use of data mining techniques, wine quality assessment through the use of readily available physiochemical wine properties. **Methods:** Tree-based ensemble methods were tuned to classify quality, which was designated as low, normal, and high. We employed the use of eXtreme Gradient Boosting and Random Forest tree-based ensemble methods. Additionally, due to the imbalanced nature of the quality class for which the majority of the data were classified as normal, undersampling and oversampling resampling techniques were used to evaluate model performance. **Results:** Utilizing cross-validation techniques, we saw immense benefits of oversampling the training data set. We consistently saw excellent performance and reached 100% classification rate of all quality classes with both classifiers. **Conclusions:** These results revealed that oversampling outperformed undersampling and that our two classifiers performed with similar accuracy with XGBoost having a slight advantage in classification of low and high quality when no resampling or undersampling was performed.

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, thus there is the potential for human bias to come into play with human testers [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]. Additionally, if human experts are used, the classification model can be used to speed up and improve their quality assessment [Cortez et al., 2009].
These techniques also have the advantage of identifying the importantance of the phsiochemical characteristics that have an impact on the quality of wine as determined by a sensory test.

There have been a few previous papers that have looked into the classification of wine. Cortez et al. [2009] and Gupta [2018] examined the data using regression methods with their response being a quality rating from 0 to 10, with 0 being the worst quality and 10 being the highest quality. The methods that they employed were multiple regression, neural networks, and support vector machines, and they both found that support vector machines produced the lowest misclassification percentage. In Hu et al [2016], a different approach to this problem was employed where instead of using quality categories from 0-10, they grouped the qualities into three different categories: low, normal, and high. They looked at only data relating to white wine. The classification methods that they employed were Adaptive Boosting and Random Forest. Additionally, due to most of the qualities being classified as "normal," they employed an oversampling technique, SMOTE in order to develop better classification on the categories with not as much data, in this case "low" and "high." They found that Random Forest had the lowest misclassification rate of the two at 5.4% [Hu et al., 2016].

The goal of this paper, following Hu et al. [2016] is to train a model that would work well to classify wines into three categories, which are: low quality, normal quality and high quality. We evaluated two classification techniques, eXtreme Gradient 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 observations. Additionally, previous papers on the white wine data set have also applied different versions of gradient boosting such as adaptive boost; therefore 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 first evaluated the classifier performance with no resampling. Since the quality class is highly imbalanced, we hypothesized no resampling 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, as this technique seemed to perform well in previous research [Hu et al., 2016]. The final resampling method we applied is a random undersampling method, where majority class data are randomly removed from the data set. We hypothesize random undersampling 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

2.1 About the Data

In this paper, we applied classification and resampling techniques to the "Wine Quality Data Set" found on the UCI Data Repository [UCI]. The data consist of information from samples of Vinho Verde, which is a product from the northwest region of Portugal. The data was collected from May, 2004 to February, 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 measurements 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, we classified the combined red and white wine data sets, and included an additional predictor variable categorizing the given observation as red or white wine. The response was measured as 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 quality 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 range of the High category is above that of the Normal and Low class.

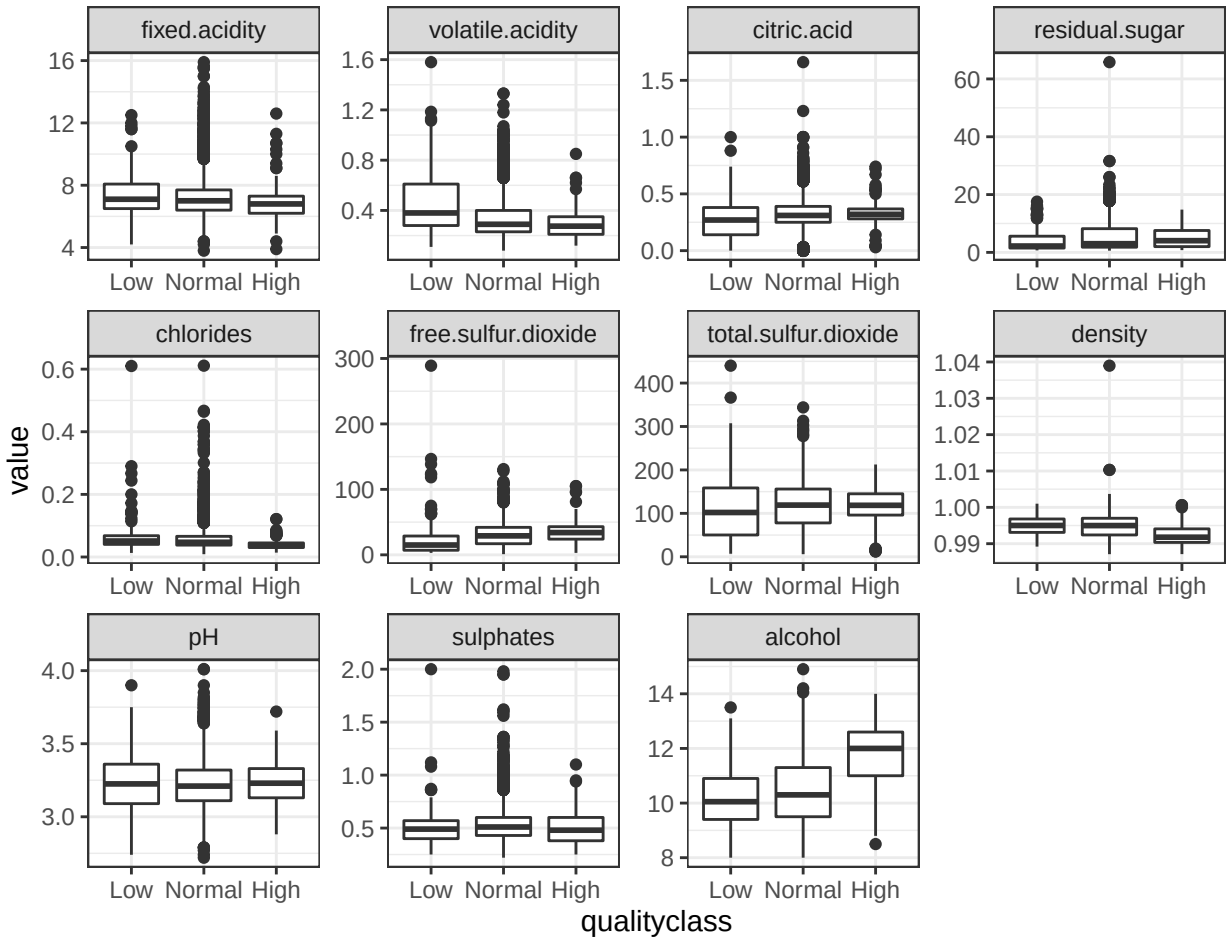


Figure 2: Distribution of the predictor variables.

Additionally, Table 1 provides physiochemical characteristics stratified by three quality classes. Continuous variables are presented as median and interquartile range (IQR) whereas discrete variables are presented as counts and proportions (%). Data were stratified by three classes of quality and comparisons were made with the Kruskal-Wallis and chi-square tests for continuous and discrete variables, respectively. We did not conduct post-hoc testing to further investigate pairwise differences. There was statistical evidence of a difference between the three classes of quality for all variables except for pH ($P < 0.001$, respectively) The low designation of wine quality was associated with the highest amount of red wine, chlorides, density, fixed acidity, and volatile acidity, and the lowest amount of alcohol, citric acid, free sulfur dioxide, residual sugar, sulphates, and total sulfur dioxide.

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.

Table 1: Basic Descriptive Statistics from the Wine Quality Dataset.

	Low No. 246	Normal No. 6,053	High No. 198	P
Fixed Acidity	7.10 (6.50 - 8.07)	7.00 (6.40 - 7.70)	6.80 (6.20 - 7.30)	< 0.001
Volatile Acidity	0.38 (0.28 - 0.61)	0.29 (0.23 - 0.40)	0.28 (0.21 - 0.35)	< 0.001
CitricAcidity	0.27 (0.14 - 0.38)	0.31 (0.25 - 0.39)	0.32 (0.28 - 0.37)	< 0.001
Residual Sugar	2.20 (1.50 - 5.60)	3.00 (1.80 - 8.20)	4.05 (2.00 - 7.57)	< 0.001
Chlorides	0.05 (0.04 - 0.07)	0.05 (0.04 - 0.07)	0.04 (0.03 - 0.04)	< 0.001
Free Sulfur Dioxide	15.00 (7.00 - 28.75)	29.00 (17.00 - 42.00)	34.00 (24.00 - 43.00)	< 0.001
Total Sulfur Dioxide	102.00 (50.25 - 158.50)	119.00 (78.00 - 156.00)	118.50 (96.00 - 145.00)	0.3496
Density	1.00 (0.99 - 1.00)	0.99 (0.99 - 1.00)	0.99 (0.99 - 0.99)	< 0.001
pH	3.23 (3.09 - 3.36)	3.21 (3.11 - 3.32)	3.23 (3.13 - 3.33)	< 0.001
Sulphates	0.49 (0.40 - 0.57)	0.51 (0.43 - 0.60)	0.48 (0.38 - 0.60)	0.0025
Alcohol	10.05 (9.40 - 10.90)	10.30 (9.50 - 11.30)	12.00 (11.00 - 12.60)	< 0.001

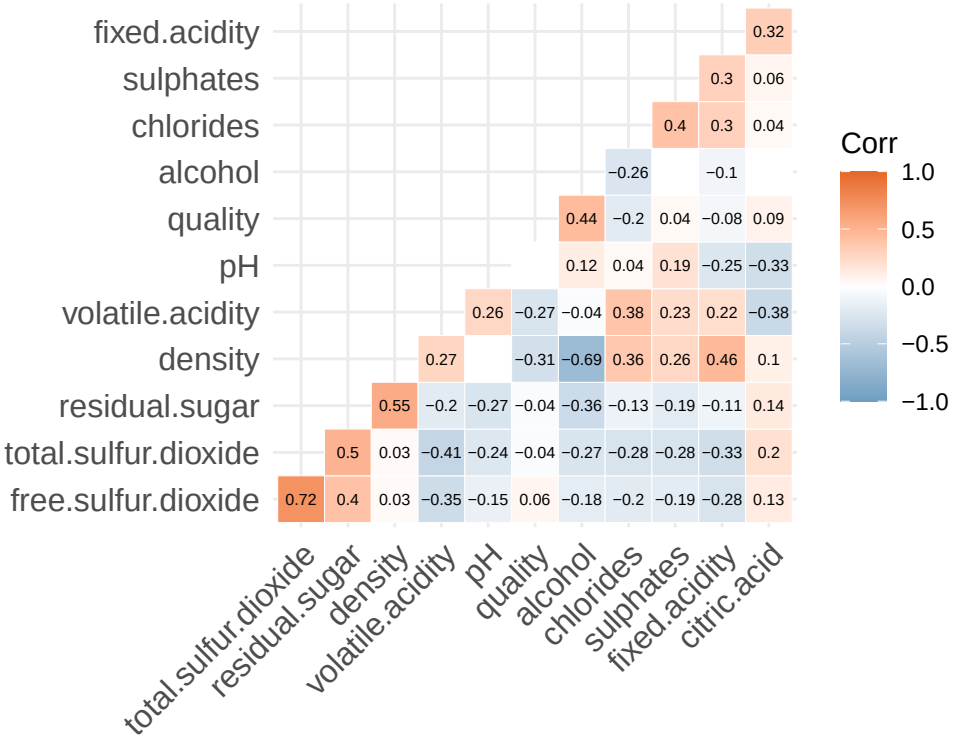


Figure 3: Correlation Plot

3 Statistical Methods

We applied two classification methods, Random Forest and eXtreme Gradient Boosting (XGBoost), 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. Classification rate is calculated as:

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

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

Monte Carlo Cross-Validation (MCMC) was used to select tuning parameters and evaluate model performance. The MCMC algorithm repeatedly splits the data into training and testing sets (B = 50 for model tuning; B = 100 for final model comparison) by randomly selecting the designated proportions (70% training; 30% testing) of the overall data set to each. The undersampling and oversampling techniques as 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; 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 (B = 50) to minimize the overall accuracy [Appendix A]; parallel computing with 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, we must first examine the imbalanced issue in our data (Figure 1) and develop our resampling techniques. This is because typical classifier algorithms assume a relatively balanced distribution across the classes, so with imbalanced data there tends to be bias towards the majority class [Sun et al., 2009]. 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 Quality" and "High Quality" were classified as "rare," and "Normal Quality" was classified as "not rare," the majority (not rare) class has 6053 observations, and the minority (rare) class has 444 observations, which is a ratio of about 13:1. Due to this imbalanced nature in the data, classifying algorithms will be biased towards classifying wines as "Normal," causing poor predictions for the "Low" and "High" quality classes. When creating the resampled data sets, the binary classification of "rare" and "not rare" is used due to the resampling algorithms relying on a binary class designation.

A method on how to lower the effects on working with an imbalanced data set is through resampling the original data set either by either 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 due to being able to predict to the minority class more accurately [Hoens and Chawla, 2013]. To see how valuable resampling was in our data set, we ran the classifying algorithms with the original imbalanced data set, a random undersampling method, and an a Synthetic Minority Over-Sampling Technique (SMOTE), which is an oversampling method. We initially evaluated the classifier performance with the original imbalanced data set (without resampling), and hypothesized that it would be associated with low performance, however, we felt it provided an appropriate baseline.

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 [Hoens and 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. On the other hand, potentially useful information is discarded, which may make the decision boundary between the majority and minority class less clear. We hypothesize this discarding of information may cause a decrease in overall prediction performance [Hoens and Chawla, 2013].

Next, we handled imbalanced data through resampling the original data set by oversampling the majority class using the SMOTE method, which is a technique that 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. That number is then added to the observation, which becomes the new synthetic data point that is then added to the data set [Chawla et al., 2002]. Through the use of a hperparameter grid search with MCMC, the number of neighbors chosen was k = 3 [Appendix A.2]. Oversampling tends 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 [Luengo et al., 2011]. This was ran using the SMOTE function in the `smotefamily` package in R [Siriseriwan, 2019].

3.2 Classification Methods

We evaluated two classifiers, Random Forest and eXtreme Boosting (XGBoost), both tree based ensemble methods. First, we utilized the Random Forest algorithm and the associated selection of independent variables for the Random Forest with the `randomForest` library in R [Liaw and Wiener, 2002]. In general, a Random Forest algorithm involves the generation of many classification trees, which are each grown through recursive partitioning 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 represent the final classification of data. After the growth phase, a leaf is potentially pruned back to a node to maximize the trade-off between complexity and rate of misclassification. A Random Forest incorporates many trees and subsequently aggregates the predictions made by each tree. We selected the number of trees and number of variables to be randomly sampled from to generate a split point at each node (mtry) by conducting a hyperparameter grid search with MCMC [Appendix A.1].

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.2].

4 Results

A comparison of the best classifiers is shown in Figure 4. With no resampling, our overall baseline classification rate for Random Forest was between 93.7% and 94.9% while the rare classes had baseline classification rates of 9% and 30.9% for Low and High Quality respectively. The XGBoost had a similar overall baseline classification rate between 93.2% and 94.9% with a slight improvement in accuracy of Low and High Quality classes over Random Forest with 14.5% and 33% baseline classification rates. When undersampling on the training data set is conducted, Random Forest performs similar to the baseline in terms of overall accuracy with a classification rate between 93.4% to 94.9%, but performs better in terms of Low Quality and High Quality classification rates at 15% and 33% respectively. However, with undersampling, XGBoost results in a slight decrease from the baseline with an overall classification rate between 92.3% and 94%. We see this decrease occur from a sacrifice in classification of Normal Quality wines at 97.8%, but an increase in the Low Quality, 24.1%, and High Quality, 34.8% classification rates. We saw immense benefits of oversampling the training data set reaching consistent performance of 100% classification rate of all quality classes with both classifiers over all MCMC splits. These results indicate that oversampling outperformed undersampling and that accuracy is similar between classifiers with a slight advantage in the XGBoost classifier over Random Forest for the rare quality classification when no resampling or random undersampling is applied.

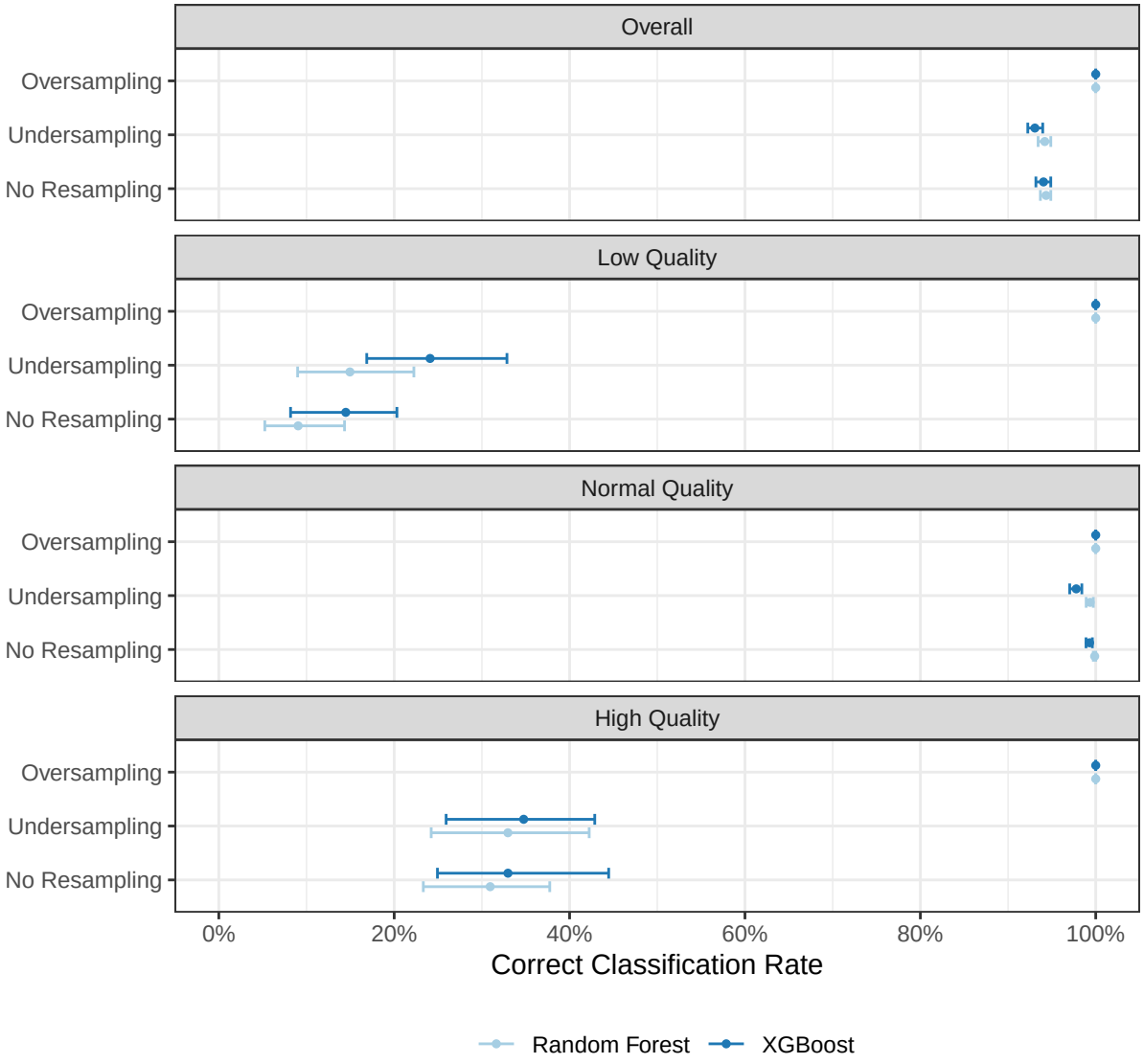


Figure 4: MCMC accuracy results for each classifier applied to each resampling technique.

5 Discussion and Conclusion

As previously noted, due to the imbalanced nature of the quality class, classifying algorithms were biased towards categorizing wines as Normal quality, and causing poor predictions for the Low and High quality classes. By employing resampling methods, we increased the classification of the minority classes. While random undersampling showed improvement in the classification rates of the low and high classes, oversampling using the SMOTE algorithm outperformed both the undersampling method and no resampling. After tuning our classifiers, we reached a consistent perfect quality classification for both red and white wines. Both classifiers, Random Forest and XGBoost, performed similar in overall performance. We see a slight advantage of XGBoost in the classification of the rare classes when undersampling of the training data or no resampling of the training data is applied. This advantage arises from the boosting technique of reassigning higher weights to observations misclassified at each boosting iteration. While XGBoost indicated a slight advantage, tuning the classifier was more intensive than the tuning required to implement the Random Forest classifier. Our results support previous research of strong performance from ensemble based methods. We saw our best classification rates when applying the SMOTE oversampling algorithm to the training data set. Future research in alternate undersampling methods is necessary to provide a comparison against the random undersampling method used in this paper. Additional model validation should be conducted on wine from other vineyards and regions in order to implement the use of a tuned model across the wine industry for quality classification.

Supplementary Material

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

Course Reflection

We have learned multiple lessons from this course and this project! From the paper specifically, we made a point of using this opportunity to learn some new technical skills on top of methods. For example, we were able to utilize some collaboration tools, like GitHub, to make working on the project as a group more seamless. Additionally, through this project and the homework assignments, we were able to improve our R coding skills, especially with R Markdown. Furthermore, the project also reiterated the importance of cross validation and the tuning of the models, as we saw increase in our performance when we spent some time working on the finding the best tuning parameters. Having a proposal due weeks before the final project due date was also helpful in pacing us through this project, as time management is an important skill. As for the course, we really appreciated how well prepared you were for each lecture and that we received timely feedback (like going through some of the theory homework questions in class). We also like how all of the material of the course felt accessible, which made it easy to follow along and learn. This also helped with the desire to learn, as the methods were described in an understandable way. We enjoyed the heavy emphasis on the importance of interpretation and communication, as the use of complex methods is of no use if readers are not able to understand it. This practice with communication will help in future careers. The only suggestion we had was to potentially spend some more time on coding in class, however the homework as very helpful with this aspect. Overall we really enjoyed the course and have learned many valuable things from it!

References

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

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

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.

T. R. Hoens and N. V. Chawla. Imbalanced datasets: from sampling to classifiers. *Imbalanced learning: Foundations, algorithms, and applications*, pages 43–59, 2013.

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.

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL https://CRAN.R-project.org/doc/Rnews/.

J. Luengo, A. Fernández, S. García, and F. Herrera. Addressing data complexity for imbalanced data sets: analysis of smote-based oversampling and evolutionary undersampling. *Soft Computing*, 15(10):1909–1936, 2011.

W. Siriseriwan. *smotefamily: A Collection of Oversampling Techniques for Class Imbalance Problem Based on SMOTE*, 2019. URL https://CRAN.R-project.org/package=smotefamily. R package version 1.3.1.

Y. Sun, A. K. Wong, and M. S. Kamel. Classification of imbalanced data: A review. *International journal of pattern recognition and artificial intelligence*, 23(04):687–719, 2009.

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

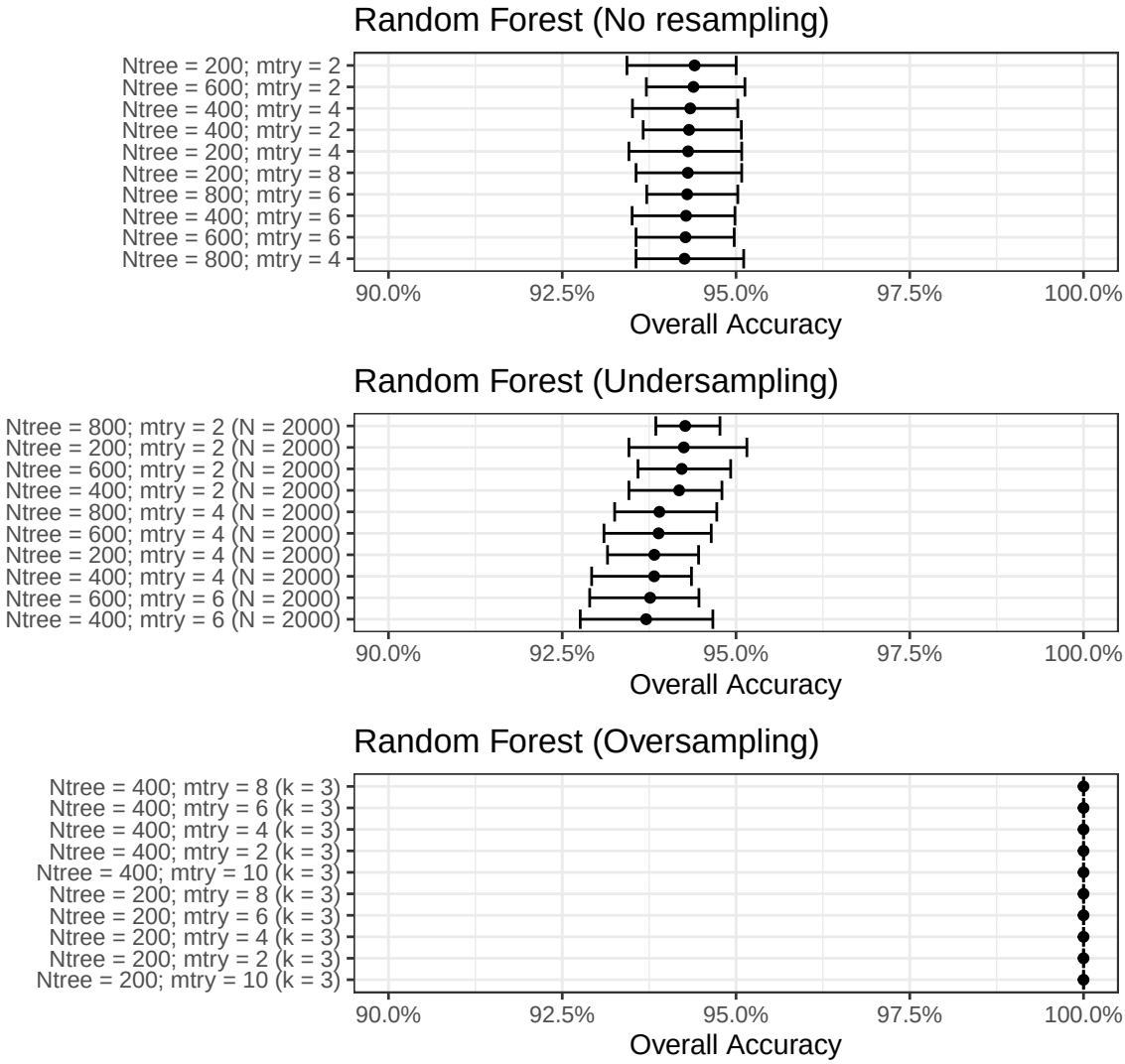
Classifiers were tuned on the original training set (no resampling), the undersampled training set, and the oversampled training set by maximizing the overall classification rate:

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

A hyperparameter grid search with Monte Carlo Cross-Validation (MCMC) was used to select tuning parameters (70% training; 30% testing; B = 50). The mean, 5th-quantile, and 95th-quantile of the overall correct classification rates are used to evaluate performance on the testing data over the B splits.

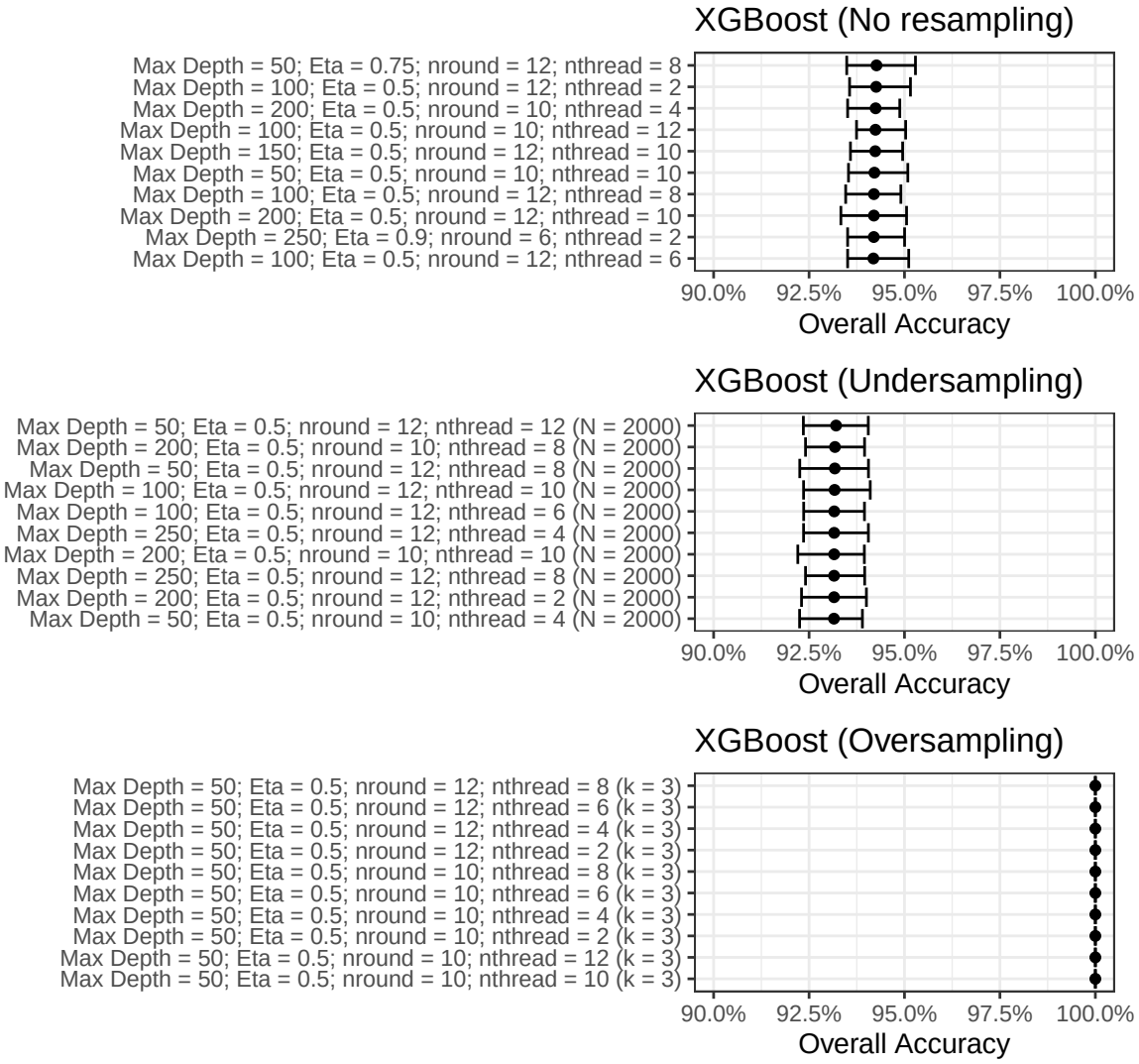
A.1 Random Forest

The Random Forest classifier was tuned by conducting a grid search over the number of trees (ntrees: 200 - 800 by 200) and number of variables randomly selected for splitting at each node (mtry: 2 - 10 by 2). The undersampling technique was conducted with 2000 randomly selected observations from the normal class. The number of nearest neighbors for the oversampling technique was selected by including this parameter in the grid search (k = 3 - 7 by 2). Figure **??** below shows the top 10 parameter combinations for Random Forest.



A.2 XGBoost

The XGBoost classifier was tuned by conducting a grid search over maximum tree depth (max.depth: 50 - 250 by 50), step size shrinkage (eta: 0.05, 1, 0.25, 0.5, 0.75, 0.9, 0.95), maximum number of boosting iterations (nround: 2 - 12 by 2), and the number of threads used in parallel computing (nthread: 2 - 12 by 2). The undersampling technique was conducted with 2000 randomly selected observations from the normal class. The number of nearest neighbors for the oversampling technique was selected by including this parameter in the grid search (k = 3 - 7 by 2). Figure **??** below shows the top 10 parameter combinations for XGBoost.



B Code

```
# LOAD LIBRARIES
# ------------------------------------------------------------
library(tidyverse)
library(readr)
library(xgboost)
library(Matrix)
library(tictoc)
library(furrr)
library(smotefamily)
# IMPORT DATA, RELEVEL FACTOR COLUMNS
# --------------------------------------
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)
    }

    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(0.05, 0.1,
        0.25, 0.5, 0.75, 0.9, 0.95), nround = seq(2, 12, 2),
    nthread = seq(2, 12, 2))
toc()
xgbMCMC.undersample.gridsearch <- xgbMCMC.undersample.gridsearch %>%
    select(accuracyGroup == "accuracy.all") %>% arrange(-mean)
xgbMCMC.undersample.gridsearch
# 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(0.05, 0.1,
        0.25, 0.5, 0.75, 0.9, 0.95), nround = seq(2, 12, 2),
    nthread = seq(2, 12, 2))
toc()
xgbMCMC.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 = accuracyGroup.labs
    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")
```