Statistical Consulting Report

Addressing Class Imbalance in Random Forest Classifier used to Identify Potential Pancreatic Cancer Cases

Prepared by Lori Kolaczkowski

April 2021

Table of Contents

Appendix

Abstract

Pancreatic ductal adenocarcinoma (PDAC) wouldn't be one of the deadliest cancers if an accurate screening test, simple enough to be as routine as measuring blood pressure, could be administered regularly to patients without any symptoms of PDAC, because once symptoms appear, it's already too late. This statistical analysis enhances the performance of a classification model developed to execute such a screening test. The classifier reads measurements of components of urine which are strong indicators of pre-PDAC or early-stage PDAC, and classifies a patient as "no" (low risk for PDAC) or "yes" (elevated risk for PDAC). A classification of "yes" alerts doctors to the need for serious follow-up early enough to save the lives of those who may be silently developing the cancer. Similar screening tests exist but are less powerful and more invasive than a urine test, and thus are not given frequently enough. Class imbalance is identified as the cause of poor performance of the original classifier. To achieve performance goals required for approval and implementation of this test in real settings, a series of 8 models which correct for class imbalance (and 2 without this correction) are trained, tested, and compared. The prescribed model which meets the specified performance goals, is a gradient boosting machine (GBM) model using the under-sampling method to balance the response classes in the training set. A repeated stratified K-fold cross-validation (K=5, repeats = 10) selects the following hyperparameter values: n.trees = 50, interaction.depth = 3, shrinkage = 0.1, and n.minobsinnode = 10. To further optimize towards specified performance goals, the probability threshold is tuned to 0.366.

Lori Kolaczkowski
Statistical Consulting Report, April 2021

Statement of Research

PDAC, the most common form of pancreatic cancer, and one of the deadliest and most aggressive of all cancers, has a very low survival rate with only 9% of patients with this cancer surviving 5 years beyond diagnosis. This poor survival rate is due to lack of detection in the early stages of development, when surgery can be effective in eliminating the cancer, as the cancer does not present symptoms until advanced. To detect this cancer early enough to substantially improve survivability, useful biomarkers associated with high risk of developing the disease must be identified through research and measured in routine screening tests which are clinically practical and ethical enough for widespread use. Traditionally, such biomarkers have been sourced from plasma or serum, requiring invasive collection and thus limiting frequency of screening.

This report provides requested expertise in statistical analysis needed for a relevant research study led by the client. The study investigates the viability of a completely noninvasive PDAC risk screening test, by examining the performance of patient age, sex, and a panel including creatinine and 3 urinary protein biomarkers, LYVE1, REG1B, and TFF1, in predicting elevated risk of PDAC. High levels of creatinine are associated with acute pancreatitis, a known risk factor for PDAC. LYVE1 (lymphatic vessel endothelial hyaluronan receptor 1) is a protein present in lymphatic vessels when pancreatic cancer is invading. REG1B (regenerating family member 1 beta) is a glycoprotein seen in patients with pancreatitis. TFF1 (trefoil factor 1) is a gastrointestinal secretory peptide which is more highly expressed during development of a variety of cancer types. LYVE1, REG1B, and TFF1 have each been found to be upregulated in PDAC precursor lesions, making them good biomarkers for pre-PDAC conditions and early-stage PDAC. If viable in its performance, a urine sample screening test based on these biomarkers would not only provide an inexpensive and noninvasive alternative method for early PDAC detection that is more conducive to frequent screening, but would likely be more effective given larger volume specimens containing higher concentrations of the biomarkers than what can be found in the blood.

The goal of the study is to train a predictive performance-focused classifier on this panel data collected from patients with and without PDAC, and classify test data with performance such that the classifier can primarily be expected to detect elevated risk for PDAC in a patient with such risk at least 90% of the time, and can be expected to label a patient without such risk as having risk no more than 25% of the time (the former being a primary expectation).

Study Design

According to the client, the study is observational with limited knowledge of the data collection process. Patients with PDAC (n_1=44) and patients without PDAC (n_0=391), a total of 435 patients, were randomly selected to provide urine specimens. For each patient, biomarker levels were measured via enzyme-linked immunosorbent assays (ELISAs), creatinine was measured via ILab Aries analyzer, and these measurements were recorded along with age and sex of the patient. All variables were then creatinine-normalized. The urine specimens were collected and provided by various pancreatic cancer research centers and university laboratories across Europe, and they were proven to remain stable through the variable measurement process. The data includes the binary categorical response variable, diagnosis (“1” if the patient who produced the urine sample had PDAC, “0” if the patient who produced the urine sample did not have PDAC), and the predictor variables sex (“F” if female and “M” if male), age (years), creatinine (mmol/L), LYVE1 (pg/ml), REG1B (pg/ml), and TFF1 (pg/ml). All predictors are continuous, except for the binary categorical variable, sex. Use of a needed function in the caret package required changing the labels of the response levels from "1" to "yes", and from "0" to "no".

The classifier takes as input the predictor values for each sample, and outputs a class label for the patient corresponding to the levels of the categorical response variable, diagnosis ("no" = no cancer, "yes" = “PDAC”). In a clinical setting, the class labels would instead refer to the patient’s risk for PDAC ("no" = low risk, "yes" = elevated risk). Patients who are labeled as having elevated risk would receive more invasive and expensive follow-up work such as imaging or pancreatic biopsy.

Using novice-level knowledge of machine learning algorithms, the client executed a random forest classification procedure on the data in R using the caret package. After training and testing the classifier on this data, the client reported that the predictive performance was shown in the output to be 90% accurate, despite true PDAC cases being correctly classified as PDAC only 30% of the time.

Client Questions

01

Accuracy Metric

Why is the model output showing high accuracy, when it has such a high misclassification rate for cancer cases?

02

Low Performance

What is causing the classifier to perform poorly with respect to identifying cancer as cancer, and how do you recommend this problem be addressed in order to meet the goals of the study?

03

Importance of Predictor Variables

How can one get a sense of which variables are most important in predicting PDAC risk?

Statistical Method

The analysis conducted to answer the client's questions was carried out using the statistical software, R, and it serves to identify the cause of the client's undesirable results, develop a recommendation for changes to the client's analysis needed to achieve the study goals, and to demonstrate a way to determine variable importance. R code and output for the entire analysis are included in the appendix for the client to reference. It began with obtaining the client's data and examining the dataset itself as well as visualizations created in performing exploratory data analysis (EDA). As the client suggested, the dataset was indeed complete and thoroughly cleaned. However, in confirming that the client defined the diagnosis level "yes" (PDAC) to be the event of interest (sometimes termed the "success"), the order of the levels of diagnosis had to be rearranged in R to ensure the program followed the same definition.

Exploratory Data Analysis

Considering the client's use of a random forest model, and a gradient boosting machine model considered later, EDA was not extensive. Both are nonparametric procedures carrying no assumptions for use in classification other than using a representative sample with a categorical response variable. Both are robust to outliers, and neither assume independent observations.
Since the client showed interest in variable importance, the analysis would include generation of variable importance plots, which can be misleading if any predictors are highly correlated with one another. Thus, a correlation heatmap (see Figure 1 below) was produced to check correlation levels between the continuous variables.

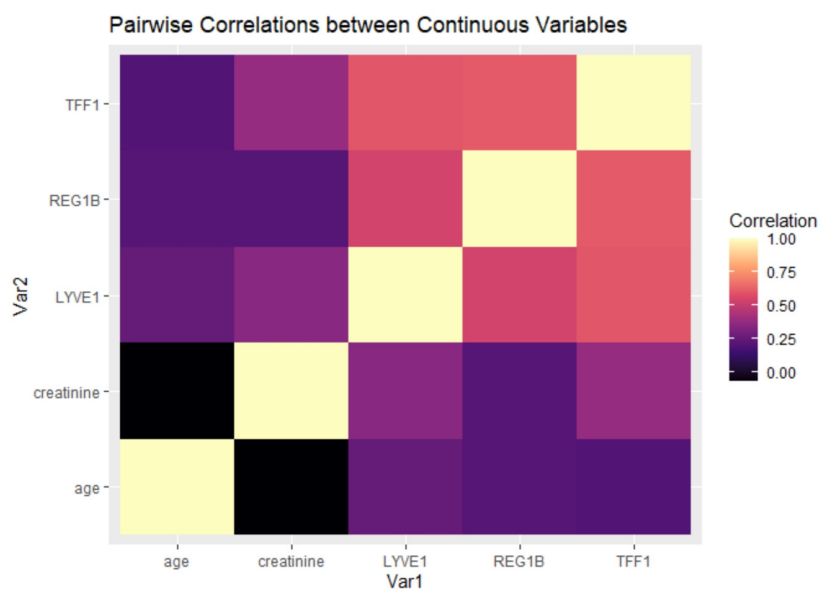


Figure 1. Heatmap showing pairwise correlations between continuous variables

Figure 1 shows no high correlation (> 0.70) was found between any unique pairs of variables. Instead, it shows low correlation between most of the continuous variables, and moderate correlation (the highest being 0.61) between all 3 pairs of biomarkers LYVE1, REG1B, and TFF1. And so, the variable importance plots to be generated later in the analysis were assumed to be trustworthy. Next, since the client's model is a classifier, it is important to see how well each class is represented in the data. To check the distribution of the response classes, a simple bar plot was created and is shown in Figure 2 below.

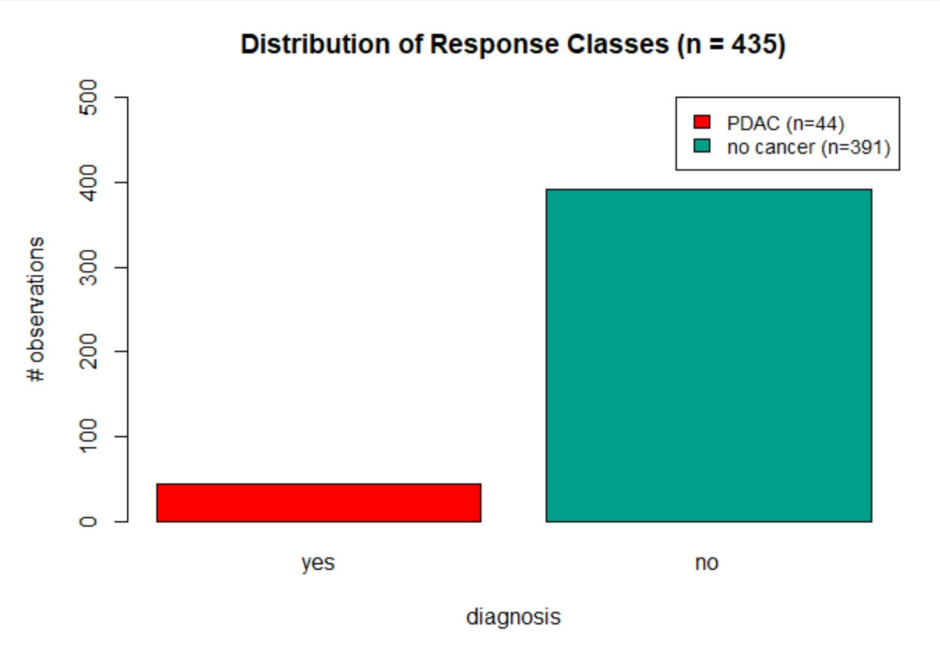


Figure 2. Bar plot showing imbalance among response classes

A moderate class imbalance is shown in Figure 2, for the response variable diagnosis. It is apparent that samples taken from patients without cancer outnumber samples taken from patients with PDAC, about 9:1. This certainly raised a red flag as class imbalance in the response variable often leads to poor predictions for the minority class if left unaddressed in the modeling process.

The Problem

In reviewing the client's methodology, it was found that this imbalance was indeed unaddressed. The response class imbalance, left unmitigated in the training of the classifier, provides reason for the classifier's poor performance in classifying the true cases of PDAC properly. Typical classification algorithms work to minimize error rate (maximize overall accuracy), without accounting for the class distribution. They are heavily biased towards the majority class. With only 44 samples representing PDAC, further reduced in the training set upon splitting the data into training and test sets, the algorithm tries to minimize error by classifying PDAC as non-cancer, resulting in a high misclassification rate within the PDAC class.

This leads to something called the Accuracy Paradox – the reason for the perceived discrepancy between accuracy value given in the output and misclassification rate in the minority class. The Accuracy Paradox occurs when a high proportion of the data is comprised of classes with negligible misclassification rates, while a small proportion of the data belongs to classes with high misclassification rates. Despite the classifier being terrible at predicting labels for the minority classes, overall, the classifier is great at making predictions, since most of the observations belong to the majority classes, and thus are classified correctly. This overall measure of classification performance is given in the testing output as "Accuracy", and is not the appropriate performance metric to use when the study is most concerned with classification performance of a minority class.

Before work can be done to mitigate the problems brought on by the class imbalance in the response, metrics are needed which will be most reliable in getting a good sense of how close the results are to the study goals. The study goals do not seek 90% overall accuracy, but rather a sensitivity of 0.90. These are completely different things. So, what is sensitivity and why should it be used as the performance measure for this study? Sensitivity is a confusion matrix metric which describes the true positive rate (in context of this study: proportion of times the classifier classifies a true cancer case to be cancer). Since the primary study goal is to detect PDAC when it is there at least 90% of the time, the goal is a sensitivity ≥ 0.9. But what else needs to be considered in measuring performance? The secondary goal of the study is to keep the proportion of non-cancer patients sent for unnecessary invasive and/or expensive follow-up tests below 0.25. This corresponds to aiming for a false positive rate of 0.25 or lower. The value for false positive rate is obtained by subtracting another confusion matrix metric, specificity, from 1.

Specificity is actually the true negative rate (proportion of times the classifier classifies a non-cancer case as a non-cancer case), and so subtracting it from 1 gives the false positive rate (proportion of times the classifier classifies a non-cancer case as a cancer case). Since 1-specificity measures how close the results are to reaching the secondary goal of the study, it should be the secondary performance measure.

Ways to Mitigate

The following are some ways to handle imbalance among response classes in building a classifier.

- Get more data for the minority class until it is close in size to the majority class. This is by far the most ideal approach, as balance is achieved with genuine data, and as a result, the best predictive performance will be achieved. If it can be done, no need to consider any of the following mitigation steps.
- In random sampling for the training and test portions of the data, stratify the random sampling by response class to ensure the training and test data hold the same ratio of response classes. Without stratifying the split, most of the minority class observations could end up in the test set, leaving the classifier virtually untrained on the minority class, and resulting in very poor predictions for that class. Similarly, use stratified splits of the training set for validation, e.g., stratified K-fold cross-validation.
- Try integrating different balancing methods (described on next page) for the training data into the model fitting and validation process.
- Repeat third bullet point for different classification algorithms appropriate for the goals of the study.

In the case of this study, the client has stated that collection of more data is unfortunately not possible. Therefore, performance would be enhanced by using stratified random sampling in forming the training and test datasets, using repeated stratified K-fold cross-validation (CV), and then applying a balancing method to the training data within each CV fold. There are 3 ways to balance the response classes when you cannot collect more genuine data – random under-sampling of the majority class, oversampling via resampling of the minority class or by generation of synthetic examples of the minority class, or a mix of under-sampling and oversampling.

Balancing Method	Description
Under-sampling (Figure 3, upper left)	Randomly sample observations from the majority class to delete until it is even with the minority class in size.
Oversampling (Figure 3, top right)	Randomly sample observations (with replacement) of the minority class until it is even with the majority class, i.e., generate copies of original minority observations.
SMOTE *Synthetic Minority Oversampling TEchnique* (Figure 3, lower right)	Generate synthetic minority samples to augment the existing ones until even with the majority class, through randomly selecting minority class observations and, for each, interpolating new values between the selected observation and its minority neighbors.
ROSE *Random Over-Sampling Examples* (Figure 3, lower left)	Simulates a new balanced training set by producing synthetic samples for each class from their respective estimated distributions (yellow and purple ovals in Figure 3) based on the original data.

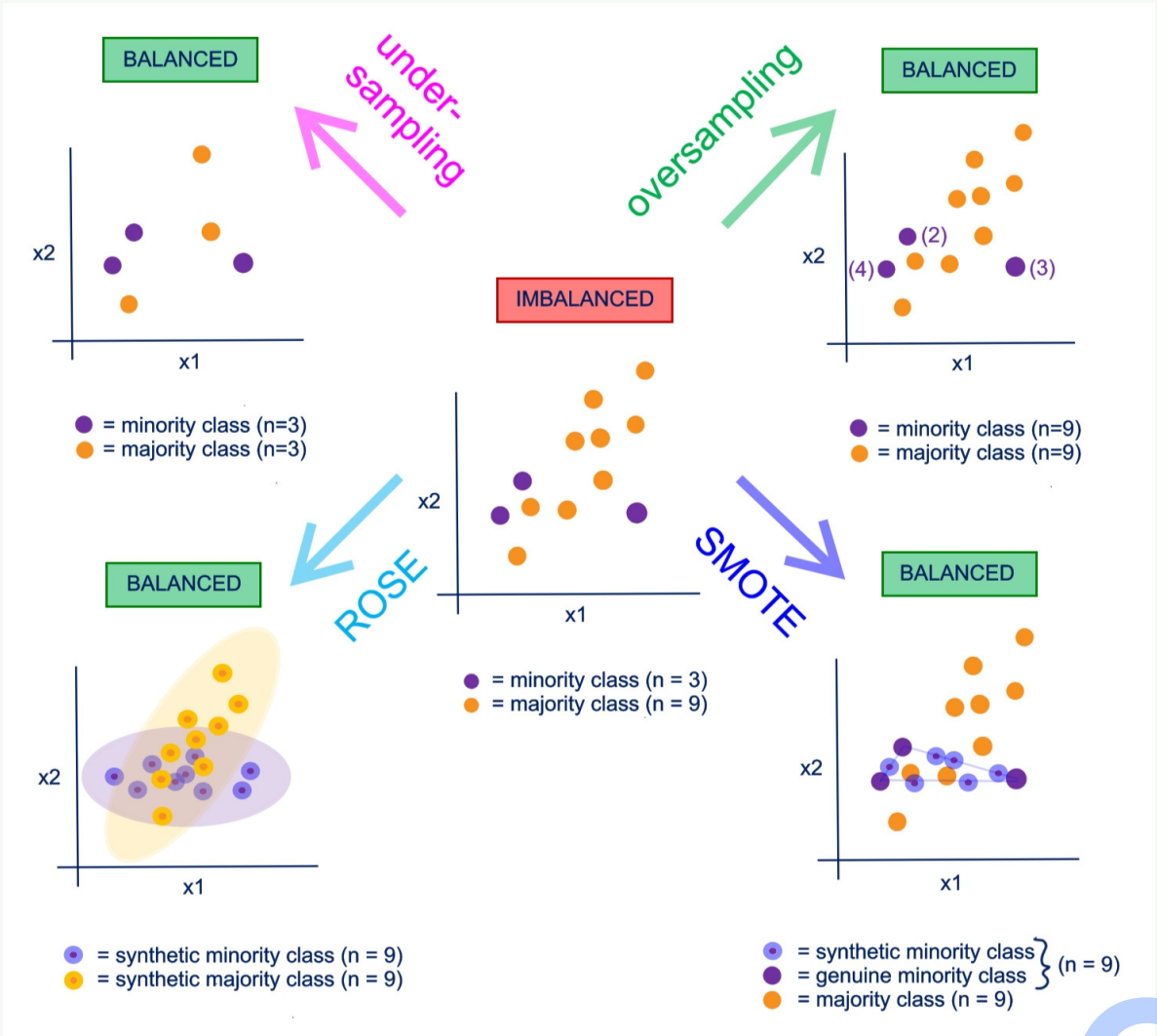


Figure 3. Visual comparison of methods for balancing classes of the training set

Analysis

The approach taken with the analysis was to try several models, using different combinations of algorithm type and balancing method type, since there is no one-size-fits-all solution. The approach with the best results changes from one dataset to the next, and with changing study goals. The classification performance of each were compared and the model(s) with highest sensitivity further tuned to optimize performance with respect to goals for sensitivity and 1-specificity.

The algorithms compared were limited to random forest (RF) and gradient boosting machine (GBM), since these generally yield the highest predictive performance among classification-appropriate algorithms, and loss of interpretability is not an issue with maximizing predictive performance being the sole concern of the study. There are more balancing techniques than under-sampling, oversampling (via minority resampling), SMOTE, and ROSE, but these were selected as balancing tools in this analysis due to their popularity in the machine learning community, and availability of packages in R for deployment. To adhere to the client's definition of PDAC as the event ("success"), this definition was maintained for every model. The primary performance metric chosen was sensitivity, with a goal of sensitivity greater or equal to 0.90, and the secondary metric chosen was 1-specificity, with a goal of 1-specificity no greater than 0.25. These metrics and limits were chosen to conform to the goals of the study as previously discussed. For this analysis, no variables were transformed since normalization and transformations are not needed for the nonparametric models used. No variables were removed since these models are unaffected by multicollinearity if it exists, and each variable helps to predict PDAC risk to some extent.

To begin, the client's RF model was reproduced for comparison with balanced models. The client's model did not stratify by class the random sampling used to split the original dataset into train (70%) and test (30%) sets, nor did they ensure random sampling for K-fold CV splits was stratified by class. No balancing techniques were employed in the client's model. Since the following aspects of the client's model were deemed appropriate and did not prevent achievement of client goals, they were used in all models for the sake of consistency (and using the same seed number for random processes as the client used in R): Use of K = 5 folds and 10 repeats in repeated K-fold CV, allowing the caret package to define the tuning grids for model hyperparameters, and use of "ROC" as the CV metric which leads to selection of hyperparameters that yield the highest area under the curve (AUC) for the ROC plot (High AUC is associated with highest predictive power in terms of sensitivity and specificity). Next, 4 balanced RF models were produced, each using a unique balancing method. All balanced models used a common training set (70%) and test set (30%) assembled via stratified (by class) random sampling of observations from the original dataset. For balanced models, repeated stratified K-fold CV was used, and the balancing method was deployed on the imbalanced training set within each CV fold, independently. This served to ensure model performance values were robust.

For each model, its test data was then fed to the model, and for each test observation, the probability of its class being "no" is generated by the model. These probabilities were subjected to the following rule to convert each of them to a class label (or class prediction), where tau (τ) = 0.5 is the default tuning parameter value (the probability threshold) of the classifier used in all models, initially, and later tuned to different values for selected models:

> If the probability that the observation = “no” is $\geq \tau$, classify the observation as “no” (low risk). Otherwise, classify the observation as “yes" (elevated risk).

This was repeated using the GBM algorithm in place of RF, generating 4 additional balanced models. For comparison’s sake a GBM model using the same unstratified training and test sets used in the client’s model was produced without use of any balancing technique. The 10 models examined in this analysis are listed below.

RF/imbalanced (client model)	GBM/imbalanced
RF/balanced (under-sampling)	GBM/balanced (under-sampling)
RF/balanced (oversampling)	GBM/balanced (oversampling)
RF/balanced (SMOTE)	GBM/balanced (SMOTE)
RF/balanced (ROSE)	GBM/balanced (ROSE)

For each model, the predicted classes were then compared to the actual classes and a confusion matrix was produced. Confusion matrix metrics, sensitivity and specificity, for each model of a given algorithm type, were visualized in a single plot for ease of comparison and to render the results more easily digestible. The model with highest sensitivity within the RF group, and the model with the highest sensitivity within the GBM group were then compared with respect to sensitivity and 1-specificity upon tuning τ to a series of different values. Tuning τ further optimizes the results towards goal values. This was visualized with ROC plots in order to see which of the 2 tuned models were best for achieving the study goals. To accommodate the client’s interest in variable (feature) importance, feature importance plots
were made for the two competing models.

Results

As shown in the left panel of Figure 4 below, the under-sampling method (under) gave the highest sensitivity (0.923) out of all 5 GBM models. The right panel of the same figure shows the under-sampling method resulted in the highest sensitivity (0.846) among the 5 RF models. So already, before tuning τ, we can expect the GBM/under model to detect PDAC 92% of the time, exceeding the goal of 90% minimum. It also meets the goal of getting a false positive no more than than 25% of the time, with 1-specificity at 0.23. Although RF/under does not make the sensitivity goal, tuning tau will get it there. It can also further improve one or both metrics for the GBM/under model. The analysis goes forward considering only these 2 models, and tunes τ in order to optimize their performance in terms of the study goals.

Sensitivity ≥ 0.90	GOALS	1-Specificity ≤ 0.25

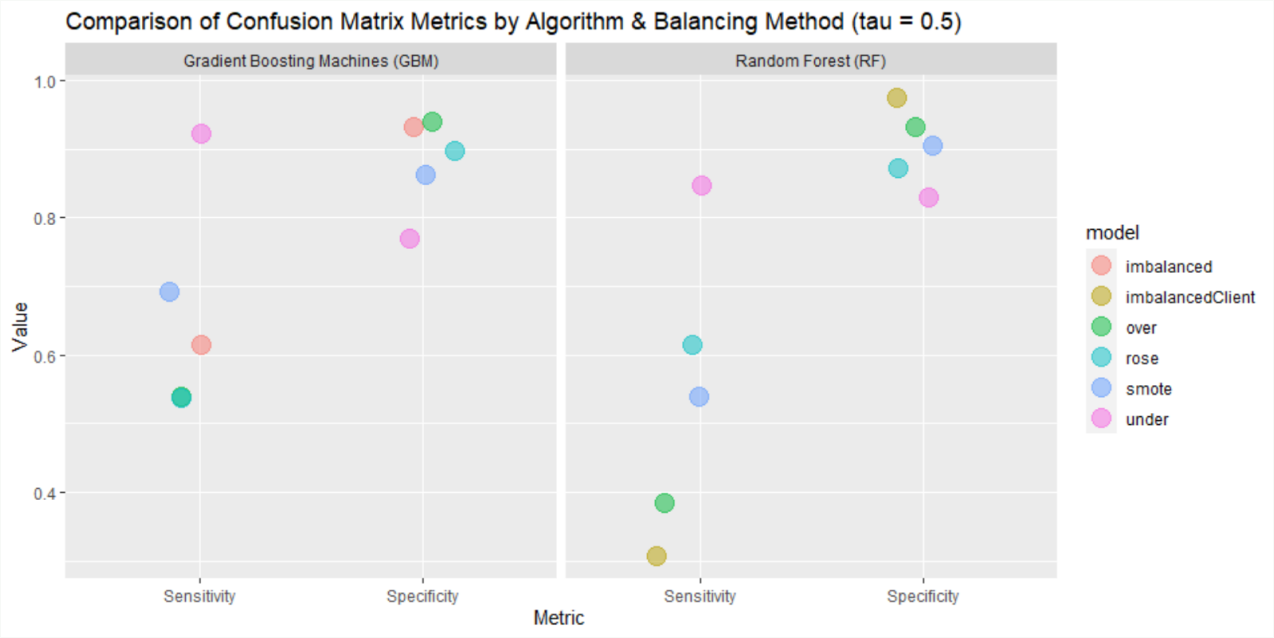


Figure 4. Pre-tuning comparison of the 10 models, by algorithm type, on goal metrics.

Relative feature importance for the GBM/under-sampling model is given on the left in Figure 5 below, with feature importance for the RF/under-sampling model on the right. In either model, it is clear LYVE1 and age are the strongest predictors for PDAC risk, with REG1B in 5th place, and sex being the weakest predictor (though it still adds value for prediction). The rank of TFF1 and creatinine as predictors depends on the model.

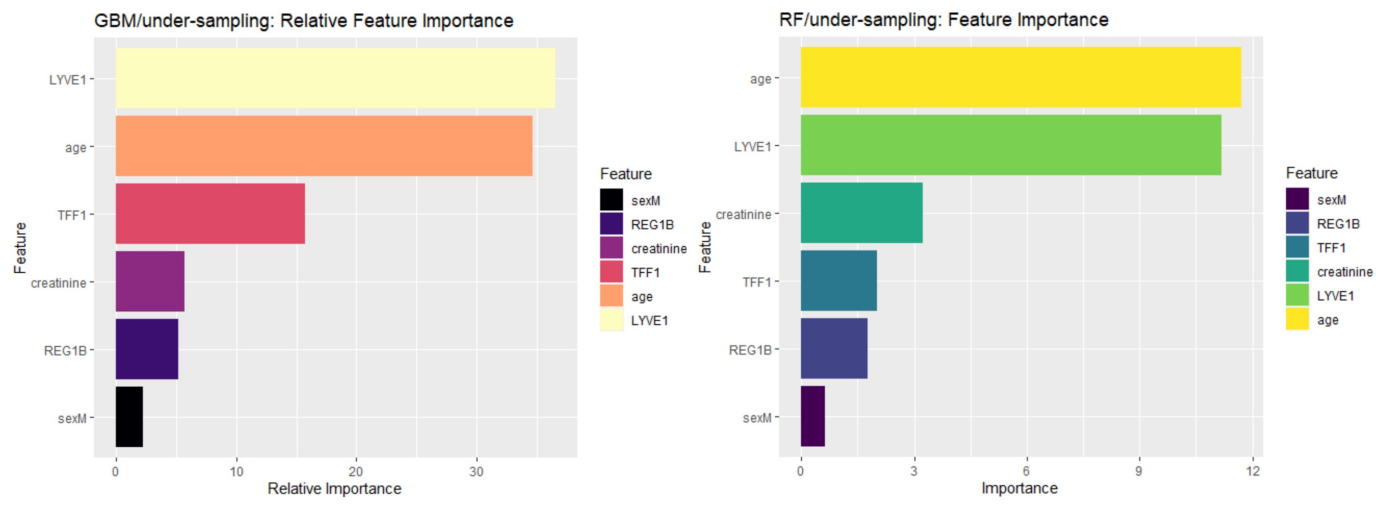


Figure 5. Feature Importance for RF/under-sampling and GBM/under-sampling

Returning to the model results on the previous page, the sensitivity of the RF/under model must be increased through tuning threshold values. The ROC curve for the RF/under model is shown below in Figure 6, and it shows the sensitivity and specificity achieved at different threshold values (the values immediately left of the "Spec" and "Sens" values). The threshold, τ, simply dictates how much evidence (probability) is required that a patient does not have the cancer in order to classify a patient as "no". In order to increase the percentage of time the classifier detects PDAC when a patient has it, τ has to be increased to require stronger evidence that a patient does not have cancer to be classified as such. Tuning τ upward from 0.5 to 0.536 or higher allows for the sensitivity goal of ≥ 0.90 to be achieved. However, for the 1-specificity goal of $\leq$ about 0.25 (or equivalently specificity $\geq$ about 0.75) to be essentially met simultaneously, τ cannot exceed 0.630. A τ of 0.536 is optimal for the RF/under model, since it offers a minimal value of 1 - specificity (0.171) while keeping sensitivity (0.923) in goal range.

The ROC curve for the GBM/under model is shown below in Figure 7. Tuning the probability threshold (τ) downward from 0.5 to 0.366 allows for the sensitivity goal of ≥ 0.90 to be held with a sensitivity of 0.923, while reducing false positive rate from 0.23 to 0.162. Therefore, a τ of 0.366 is optimal for the GBM/under model.

Though the two models, once tuned, have very close metrics, GBM/under actually has a very slightly lower value for 1 - specificity. The AUC (area under the curve) is a type of overall measure of performance (higher is better) and is included as secondary support for the final model choice.

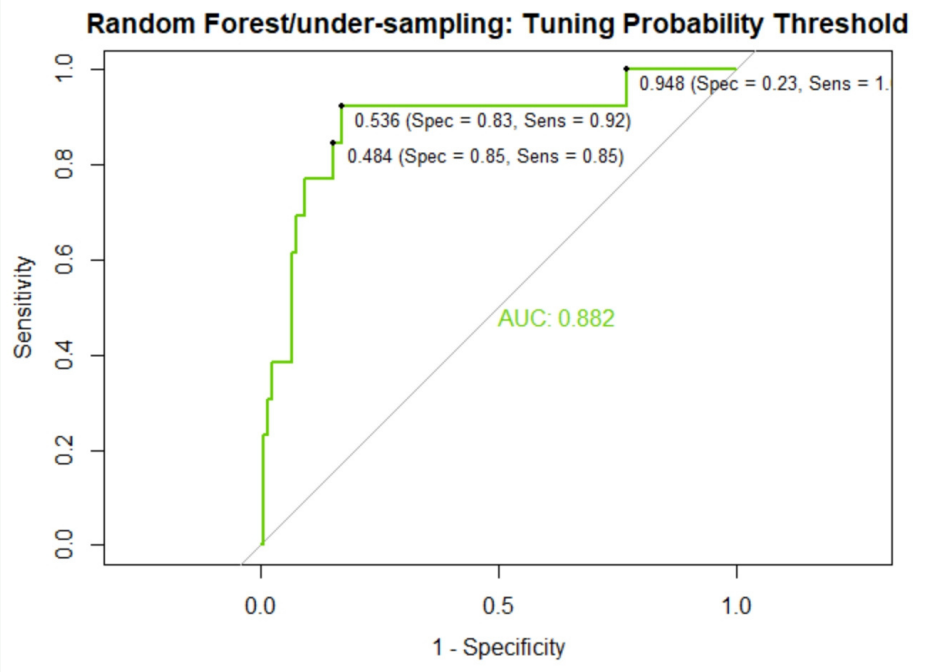


Figure 6. ROC plot for RF/under-sampling model

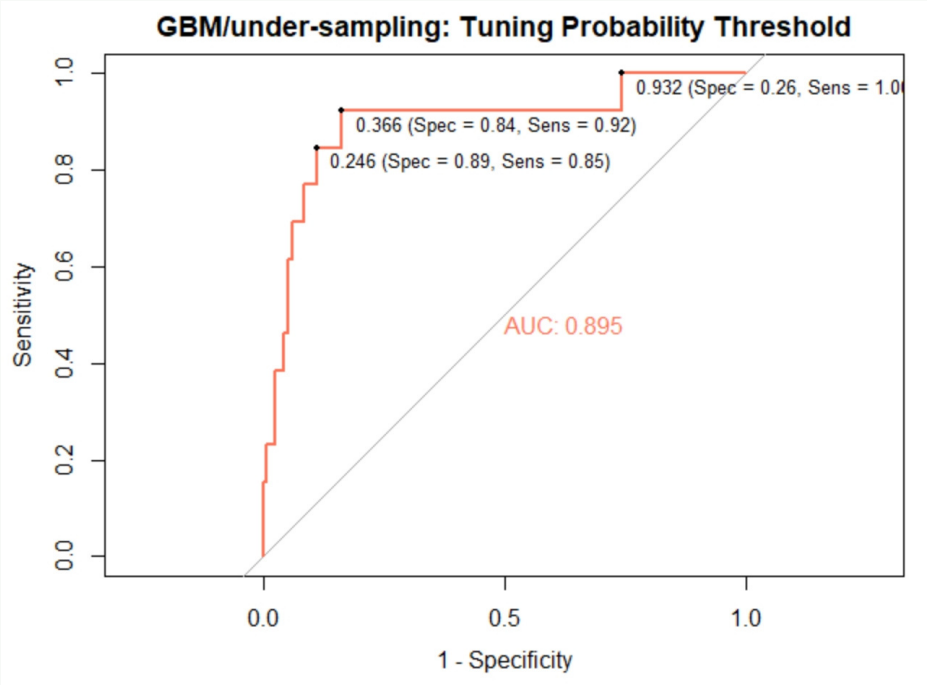


Figure 7. ROC plot for GBM/under-sampling model

Figure 10 below visually compares the sensitivity and specificity of the GBM/under model before tuning (under), and the same model after tuning τ to 0.366 (underTuned).

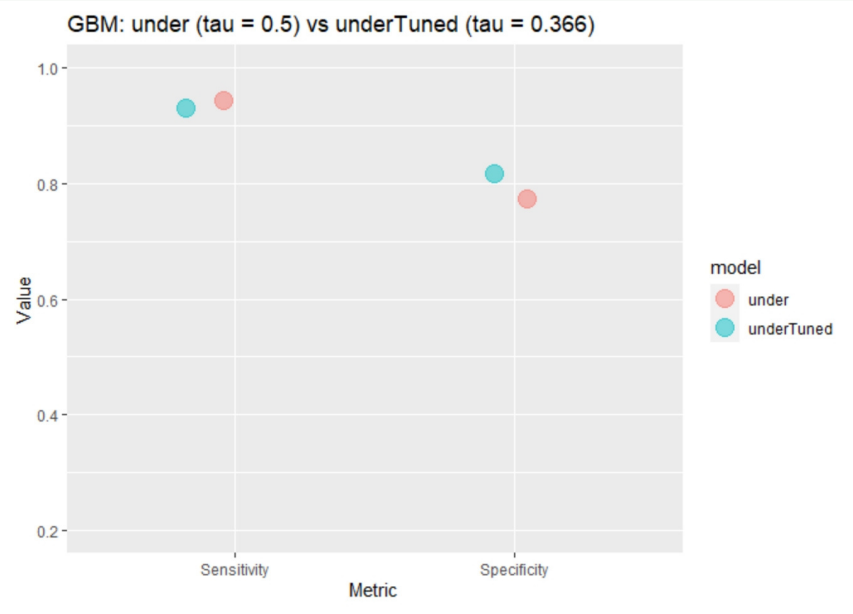


Figure 10. Plot of GBM/under-sampling before and after tuning

The GBM/under classification model with τ = 0.366 can be expected to detect cancer risk in patients 92% of the time when cancer risk is there, giving good reason to continue with more invasive and/or expensive procedures such as biopsy of the pancreas and imaging, which will catch presence of PDAC early enough to save many lives. And it comes with a relatively low cost in terms of false positives – only 16% of those without high cancer risk will be recommended to undergo further follow-up procedures.

Conclusions

Answers to the client's questions:

01 Accuracy Metric

The reported performance measure discrepancy was due to using a performance metric which is not appropriate for the goals of the study. Given the goals of this study, total accuracy should not be the chosen metric, but rather sensitivity should be the primary metric with 1-specificity as the secondary metric.

02 Low Performance

The initial poor performance of the random forest classifier in detecting cancer was a result of moderate class imbalance in the response variable. In integrating methods for balancing the response classes into the model training process as detailed in the methods section, the following classification procedure is most optimal in meeting the specified goals, and is thus recommended for use in the study:

Gradient Boosting Machine algorithm using under-sampling with classification cutoff $\tau = 0.366$.

(hyperparameters chosen by CV: n.trees = 50, interaction.depth = 3, shrinkage = 0.1, n.minobsinnode = 10)

Although this procedure meets the study goals, it could have shown improved results if the study had allowed for collection of enough genuine urine samples from subjects with PDAC. Still, the above approach is an effective alternative.

03 Importance of Predictor Variables (Features)

Feature importance can be produced in R from random forest models using varImp(model) from the caret package and from gradient boosting machine models using summary(model), and both can be plotted using ggplot2 as shown in the appendix under "Check variable importance...".
But be aware that high correlation between features can result in misleading feature importance plots. Correlations can be checked by making a correlation heat map as shown in the appendix in the exploratory data analysis portion.

Closing Statements

It can be done – even with moderate class imbalance, pancreatic cancer risk can be detected accurately with a simple noninvasive and inexpensive urine sample test by running the resulting data through the recommended classification procedure. And its predictive performance, since it meets goals for true positive rate and false positive rate, renders the procedure clinically practical enough to make implementation of the urine sample screening test a feasible and attractive option in real settings.

Many people die from PDAC because it must be caught very early to beat it, but it instead advances undetected for lack of strong enough evidence to warrant the invasive and/or costly tests which are needed to find the cancer. Run through an appropriately crafted classifier, the biomarkers measured in this study's urine sample panel are strong enough predictors of PDAC to provide the means to routinely and accurately screen patients for follow-up need, long before PDAC can become advanced enough to present the first signs and symptoms that it's already too late.

As other urinary proteins and peptides have shown promise as biomarkers for other cancers including lung cancer, colon cancer, and ovarian cancer, this study can inform similar studies in other cancer areas in which such proteins are explored as panel candidates for simple and accurate routine screening tests. This study should also serve as an example of some ways to handle class imbalance in classification procedures, and to encourage sampling designs which allow for balanced classes in categorical responses when classifiers are to be trained on them, regardless of the research area.

Appendix

R Code and Output for Client's Reference

Analysis Script & Output: Addressing Class Imbalance in Random Forest Classifier used to Identify Potential Pancreatic Cancer Cases

Lori Kolaczkowski (kolaczkowskil@stat.tamu.edu)

4/9/2021

```
#load packages
library(ggplot2) #for some plots
library(viridis) #for plot color
library(wesanderson) #for plot color
library(reshape2) #for heat map (melting correlation)
library(caret) #for all the ML algorithms
library(DMwR) #for smote
library(ROSE) #for rose
library(dplyr) #for data manipulation needed for confusion matrix plots
library(tidyr) #for data manipulation needed for confusion matrix plots
library(pROC) #for ROC plots
library(knitr) #for ROC tables
```

```
#read in the data
data <- read.csv("D:/STAT 684 - Consulting/Project/pancCancer.csv", header = TRUE)

#check it out (un-comment next line to print the data)
#data
str(data)
```

```
## 'data.frame':    435 obs. of  7 variables:
##  $ age       : int  33 81 51 61 62 53 70 58 59 56 ...
##  $ sex       : chr  "F" "F" "M" "M" ...
##  $ diagnosis : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ creatinine: num  1.832 0.973 0.78 0.701 0.215 ...
##  $ LYVE1     : num  0.89322 2.03758 0.14559 0.0028 0.00086 ...
##  $ REG1B     : num  52.9 94.5 102.4 60.6 65.5 ...
##  $ TFF1      : num  654.3 209.5 461.1 142.9 41.1 ...
```

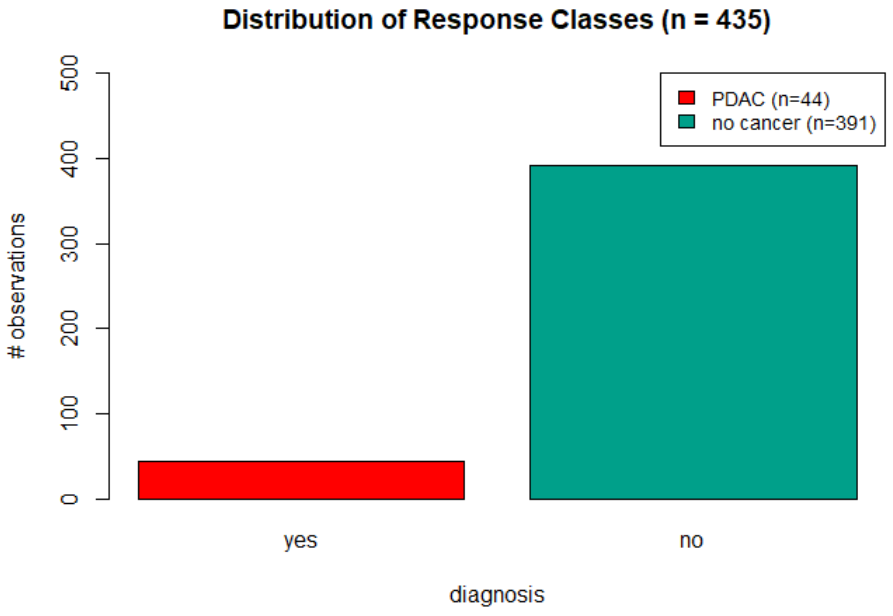
```
#convert categorical variables to factors
data$sex <- as.factor(data$sex)
data$diagnosis <- as.factor(data$diagnosis)

#reorder levels of diagnosis so that "1" shows as 1 and "0" shows as 2
#by default caret will call the 1st level the event (or success)
data$diagnosis <- factor(data$diagnosis, levels = c("1","0"))

#now change the response level labels so that they are valid R variable names.
#This is required for caret's classProbs = TRUE to be used
levels(data$diagnosis) = c("yes", "no")
```

```
#check the data
str(data)
```

```
## 'data.frame':    435 obs. of  7 variables:
##  $ age       : int  33 81 51 61 62 53 70 58 59 56 ...
##  $ sex       : Factor w/ 2 levels "F","M": 1 1 2 2 2 2 2 1 1 1 ...
##  $ diagnosis : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
##  $ creatinine: num  1.832 0.973 0.78 0.701 0.215 ...
##  $ LYVE1     : num  0.89322 2.03758 0.14559 0.0028 0.00086 ...
##  $ REG1B     : num  52.9 94.5 102.4 60.6 65.5 ...
##  $ TFF1      : num  654.3 209.5 461.1 142.9 41.1 ...
```

First, always run through some exploratory data analysis.

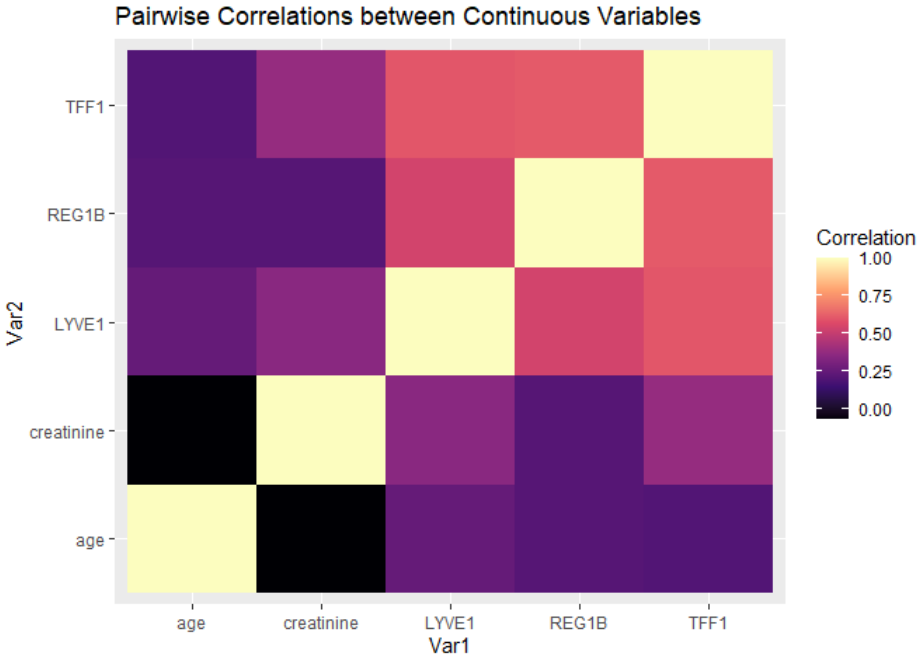
```
###### check distribution of predictors/features

# check distribution of response/supervisor
barplot(table(data$diagnosis),
        ylab = "# observations",
        xlab = "diagnosis",
        ylim = c(0,500),
        main = "Distribution of Response Classes (n = 435)",
        col = wes_palette("Darjeeling1", 5, type = "continuous"))
legend('topright', legend=c("PDAC (n=44)", "no cancer (n=391)"), cex=0.9,
       fill = wes_palette("Darjeeling1", 5, type = "continuous"))
```




```
###### check correlation between all pairs of features with a correlation heatmap
# make and melt the correlation matrix for numeric variables
feats <- data[,-c(2,3)]
cormat <- round(cor(feats),2)
melted_cormat <- melt(cormat)
melted_cormat
```

```
##          Var1       Var2 value
## 1         age        age  1.00
## 2  creatinine        age -0.07
## 3       LYVE1        age  0.25
## 4       REG1B        age  0.21
## 5        TFF1        age  0.20
## 6         age creatinine -0.07
## 7  creatinine creatinine  1.00
## 8       LYVE1 creatinine  0.35
## 9       REG1B creatinine  0.21
## 10       TFF1 creatinine  0.38
## 11        age      LYVE1  0.25
## 12 creatinine      LYVE1  0.35
## 13      LYVE1      LYVE1  1.00
## 14      REG1B      LYVE1  0.54
## 15       TFF1      LYVE1  0.60
## 16        age      REG1B  0.21
## 17 creatinine      REG1B  0.21
## 18      LYVE1      REG1B  0.54
## 19      REG1B      REG1B  1.00
## 20       TFF1      REG1B  0.61
## 21        age       TFF1  0.20
## 22 creatinine       TFF1  0.38
## 23      LYVE1       TFF1  0.60
## 24      REG1B       TFF1  0.61
## 25       TFF1       TFF1  1.00
```

```
# make the correlation heatmap using viridis palettes
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  scale_fill_viridis_c(option="magma", name="Correlation") +
  labs(title="Pairwise Correlations between Continuous Variables")
```



Original random forest model:

No stratified train/test split, K-fold CV is not stratified, and no sampling to correct imbalance

```
# data is split into train and test sets using random sampling (not stratified by response class),
#so that 70% of the data will be used to train the model
set.seed(777)
these_client <- sample(1:nrow(data), floor(nrow(data)*0.7))
train_client <- data[these_client, ]
test_client <- data[-these_client, ]

# repeated K-fold cross-validation object is created (K=5, repeats=10)
cv_client <- trainControl(method = "repeatedcv",
                  number = 5,
                  repeats = 10,
                  verboseIter = FALSE,
                  classProbs = TRUE,
                  summaryFunction = twoClassSummary)

# RF model is fit implementing above CV object
set.seed(777)
modelrf_client <- caret::train(diagnosis ~ .,
                           data = train_client,
                           method = "rf",
                           trControl = cv_client,
                           metric = "ROC")

final_client <- data.frame(actual = test_client$diagnosis,
                       predict(modelrf_client, newdata = test_client, type = "prob"))
final_client$predict <- ifelse(final_client$no > 0.5, "no", "yes")
final_client$predict <- as.factor(final_client$predict) #predictions must also be a factor
cmRF_imbalancedClient <- confusionMatrix(final_client$predict, test_client$diagnosis)
```

```
## Warning in confusionMatrix.default(final_client$predict, test_client$diagnosis):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
cmRF_imbalancedClient
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   4   3
##        no    9 115
##
##                Accuracy : 0.9084
##                  95% CI : (0.8455, 0.9518)
##     No Information Rate : 0.9008
##     P-Value [Acc > NIR] : 0.4573
##
##                   Kappa : 0.3552
##
##  Mcnemar's Test P-Value : 0.1489
##
```

```
##             Sensitivity : 0.30769
##             Specificity : 0.97458
##          Pos Pred Value : 0.57143
##          Neg Pred Value : 0.92742
##              Prevalence : 0.09924
##          Detection Rate : 0.03053
##    Detection Prevalence : 0.05344
##       Balanced Accuracy : 0.64113
##
##        'Positive' Class : yes
##
```

```
modelrf_client
```

```
## Random Forest
##
## 304 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 243, 243, 243, 243, 244, 243, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8784138  0.3685714  0.9839125
##   4     0.8661055  0.3585714  0.9787946
##   6     0.8609318  0.3495238  0.9751178
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Mitigation Suggestions

Let's be sure to stratify the train/test split

```
# createDataPartition carries out random sampling of observations, stratified by response class.
#70% of the data will be used to train the model
set.seed(777)
these <- createDataPartition(data$diagnosis, p = 0.7, list = FALSE)
train <- data[these, ]
test  <- data[-these, ]
```

Now let's fit a series of random forest models - each implementing a different sampling method (undersampling, oversampling, SMOTE, ROSE) to achieve class balance in the response variable, diagnosis. K-fold cross-validation must also use stratified splits (cvIndex takes care of this).

RF using under-sampling

```
#this is to ensure the sampling which forms the folds is stratified by class
cvIndex <- createFolds(factor(train$diagnosis), 5, returnTrain = T)

cv_us <- trainControl(index = cvIndex,
```

```
                     method = "repeatedcv",
                     number = 5,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling="down",
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary)

set.seed(777)
modelrf_us <- caret::train(diagnosis ~ .,
                           data = train,
                           method = "rf",
                           trControl = cv_us,
                          metric = "ROC")

final_us <- data.frame(actual = test$diagnosis,
                       predict(modelrf_us, newdata = test, type = "prob"))
final_us$predict <- ifelse(final_us$no > 0.5, "no", "yes")
final_us$predict <- as.factor(final_us$predict)
cmRF_under <- confusionMatrix(final_us$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_us$predict, test$diagnosis): Levels are
## not in the same order for reference and data. Refactoring data to match.
```

```
cmRF_under
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes no
##        yes  11 20
##        no    2 97
##
##               Accuracy : 0.8308
##                 95% CI : (0.7551, 0.8908)
##    No Information Rate : 0.9
##    P-Value [Acc > NIR] : 0.9950566
##
##                  Kappa : 0.418
##
##  Mcnemar's Test P-Value : 0.0002896
##
##             Sensitivity : 0.84615
##             Specificity : 0.82906
##          Pos Pred Value : 0.35484
##          Neg Pred Value : 0.97980
##              Prevalence : 0.10000
##          Detection Rate : 0.08462
##    Detection Prevalence : 0.23846
##       Balanced Accuracy : 0.83761
##
##        'Positive' Class : yes
##
```

```
modelrf_us
```

```
## Random Forest
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8605972  0.7190476  0.8140741
##   4     0.8415713  0.8142857  0.7698990
##   6     0.8761199  0.8142857  0.7740067
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

RF using oversampling

```
cv_os <- trainControl(index = cvIndex,
                      method = "repeatedcv",
                      number = 5,
                      repeats = 10,
                      verboseIter = FALSE,
                      sampling="up",
                      classProbs = TRUE,
                      summaryFunction = twoClassSummary)

set.seed(777)
modelrf_os <- caret::train(diagnosis ~ .,
                           data = train,
                           method = "rf",
                           trControl = cv_os,
                           metric = "ROC")

final_os <- data.frame(actual = test$diagnosis,
                       predict(modelrf_os, newdata = test, type = "prob"))
final_os$predict <- ifelse(final_os$no > 0.5, "no", "yes")
final_os$predict <- as.factor(final_os$predict)
cmRF_over <- confusionMatrix(final_os$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_os$predict, test$diagnosis): Levels are
## not in the same order for reference and data. Refactoring data to match.
```

```
cmRF_over
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   5   8
```

```
##        no    8 109
##
##               Accuracy : 0.8769
##                 95% CI : (0.8078, 0.928)
##    No Information Rate : 0.9
##    P-Value [Acc > NIR] : 0.847
##
##                  Kappa : 0.3162
##
##  Mcnemar's Test P-Value : 1.000
##
##             Sensitivity : 0.38462
##             Specificity : 0.93162
##          Pos Pred Value : 0.38462
##          Neg Pred Value : 0.93162
##              Prevalence : 0.10000
##          Detection Rate : 0.03846
##    Detection Prevalence : 0.10000
##       Balanced Accuracy : 0.65812
##
##        'Positive' Class : yes
##
```

```
modelrf_os
```

```
## Random Forest
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8725894  0.3619048  0.9672054
##   4     0.8574571  0.4238095  0.9525253
##   6     0.8656894  0.4619048  0.9452525
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

RF using SMOTE

```
cv_smote <- trainControl(index = cvIndex,
                         method = "repeatedcv",
                         number = 5,
                         repeats = 10,
                         verboseIter = FALSE,
                         sampling="smote",
                         classProbs = TRUE,
                         summaryFunction = twoClassSummary)
```

```
set.seed(777)
modelrf_smote <- caret::train(diagnosis ~ .,
                          data = train,
                          method = "rf",
                          trControl = cv_smote,
                          metric = "ROC")

final_smote <- data.frame(actual = test$diagnosis,
                         predict(modelrf_smote, newdata = test, type = "prob"))

final_smote$predict <- ifelse(final_smote$no > 0.5, "no", "yes")
final_smote$predict <- as.factor(final_smote$predict)
cmRF_smote <- confusionMatrix(final_smote$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_smote$predict, test$diagnosis): Levels
## are not in the same order for reference and data. Refactoring data to match.
```

```
cmRF_smote
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   7  11
##        no    6 106
##
##               Accuracy : 0.8692
##                 95% CI : (0.7989, 0.9219)
##    No Information Rate : 0.9
##    P-Value [Acc > NIR] : 0.9021
##
##                  Kappa : 0.3796
##
##  Mcnemar's Test P-Value : 0.3320
##
##             Sensitivity : 0.53846
##             Specificity : 0.90598
##          Pos Pred Value : 0.38889
##          Neg Pred Value : 0.94643
##              Prevalence : 0.10000
##          Detection Rate : 0.05385
##    Detection Prevalence : 0.13846
##       Balanced Accuracy : 0.72222
##
##        'Positive' Class : yes
##
```

```
modelrf_smote
```

```
## Random Forest
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8784560  0.6523810  0.8687542
##   4     0.8963043  0.6857143  0.8614141
##   6     0.8626888  0.7523810  0.8759596
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

RF using ROSE

```
cv_rose <- trainControl(index = cvIndex,
                        method = "repeatedcv",
                        number = 5,
                        repeats = 10,
                        verboseIter = FALSE,
                        sampling="rose",
                        classProbs = TRUE,
                        summaryFunction = twoClassSummary)

set.seed(777)
modelrf_rose <- caret::train(diagnosis ~ .,
                             data = train,
                             method = "rf",
                             trControl = cv_rose,
                            metric = "ROC")

final_rose <- data.frame(actual = test$diagnosis,
                         predict(modelrf_rose, newdata = test, type = "prob"))
final_rose$predict <- ifelse(final_rose$no > 0.5, "no", "yes")
final_rose$predict <- as.factor(final_rose$predict)
cmRF_rose <- confusionMatrix(final_rose$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_rose$predict, test$diagnosis): Levels
## are not in the same order for reference and data. Refactoring data to match.
```

```
cmRF_rose
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   8  15
##        no    5 102
##
##               Accuracy : 0.8462
##                 95% CI : (0.7724, 0.9034)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 0.98107
##
```

```
##                  Kappa : 0.3631
##
##  Mcnemar's Test P-Value : 0.04417
##
##             Sensitivity : 0.61538
##             Specificity : 0.87179
##          Pos Pred Value : 0.34783
##          Neg Pred Value : 0.95327
##              Prevalence : 0.10000
##          Detection Rate : 0.06154
##    Detection Prevalence : 0.17692
##       Balanced Accuracy : 0.74359
##
##        'Positive' Class : yes
##
```

```
modelrf_rose
```

```
## Random Forest
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using ROSE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.8621180  0.6809524  0.8651852
##   4     0.8730319  0.6142857  0.8832323
##   6     0.8673577  0.6238095  0.8540741
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.
```

Now, for comparison, we will make the above models, but using a gradient boosting machines (GBM) algorithm, to see how the random forest (RF) model with highest sensitivity compares to the GBM model with the highest sensitivity. Once all models are trained and tested, we will compare their performances based on confusion matrix metrics, in a plot.

GBM using under-sampling

```
# Using cv_us as previously defined
set.seed(777)
modelgbm_us <- caret::train(diagnosis ~ .,
                            data = train,
                            method = "gbm",
```

```
                          verbose = FALSE,
                          trControl = cv_us,
                          metric = "ROC")

final_usGBM <- data.frame(actual = test$diagnosis,
                          predict(modelgbm_us, newdata = test, type = "prob"))
final_usGBM$predict <- ifelse(final_usGBM$no > 0.5, "no", "yes")
final_usGBM$predict <- as.factor(final_usGBM$predict)
cmGBM_under <- confusionMatrix(final_usGBM$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_usGBM$predict, test$diagnosis): Levels
## are not in the same order for reference and data. Refactoring data to match.
```

```
cmGBM_under
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes no
##        yes  12 27
##        no    1 90
##
##                Accuracy : 0.7846
##                  95% CI : (0.704, 0.8519)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3665
##
##  Mcnemar's Test P-Value : 2.306e-06
##
##             Sensitivity : 0.92308
##             Specificity : 0.76923
##          Pos Pred Value : 0.30769
##          Neg Pred Value : 0.98901
##              Prevalence : 0.10000
##          Detection Rate : 0.09231
##    Detection Prevalence : 0.30000
##       Balanced Accuracy : 0.84615
##
##        'Positive' Class : yes
##
```

```
modelgbm_us
```

```
## Stochastic Gradient Boosting
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using down-sampling
```

```
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.8462787  0.7523810  0.7702357
##   1                  100      0.8262434  0.7476190  0.7775758
##   1                  150      0.8200401  0.6476190  0.7993939
##   2                   50      0.8378018  0.8476190  0.7371044
##   2                  100      0.8306670  0.8095238  0.7370370
##   2                  150      0.8223793  0.8142857  0.7444444
##   3                   50      0.8728523  0.8761905  0.7521212
##   3                  100      0.8448982  0.8428571  0.7411448
##   3                  150      0.8375469  0.8761905  0.7449158
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

GBM using oversampling

```
# Using cv_os as previously defined
set.seed(777)
modelgbm_os <- caret::train(diagnosis ~ .,
                            data = train,
                            method = "gbm",
                            verbose = FALSE,
                            trControl = cv_os,
                            metric = "ROC")

final_osGBM <- data.frame(actual = test$diagnosis,
                          predict(modelgbm_os, newdata = test, type = "prob"))
final_osGBM$predict <- ifelse(final_osGBM$no > 0.5, "no", "yes")
final_osGBM$predict <- as.factor(final_osGBM$predict)
cmGBM_over <- confusionMatrix(final_osGBM$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_osGBM$predict, test$diagnosis): Levels
## are not in the same order for reference and data. Refactoring data to match.
```

```
cmGBM_over
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   7   7
##        no    6 110
##
##                Accuracy : 0.9
##                  95% CI : (0.8351, 0.9457)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 0.5731
##
##                   Kappa : 0.4628
```

```
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.53846
##             Specificity : 0.94017
##          Pos Pred Value : 0.50000
##          Neg Pred Value : 0.94828
##              Prevalence : 0.10000
##          Detection Rate : 0.05385
##    Detection Prevalence : 0.10769
##       Balanced Accuracy : 0.73932
##
##        'Positive' Class : yes
##
```

```
modelgbm_os
```

```
## Stochastic Gradient Boosting
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using up-sampling
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.8840532  0.8190476  0.8395286
##   1                  100      0.8841607  0.6904762  0.8832997
##   1                  150      0.8779798  0.6904762  0.8832997
##   2                   50      0.8690620  0.6238095  0.8979125
##   2                  100      0.8783454  0.5904762  0.9161616
##   2                  150      0.8727593  0.5190476  0.9233670
##   3                   50      0.8721116  0.6571429  0.9015488
##   3                  100      0.8813612  0.5904762  0.9415488
##   3                  150      0.8901090  0.5571429  0.9488215
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

GBM using SMOTE

```
# Using cv_smote as previously defined
set.seed(777)
modelgbm_smote <- caret::train(diagnosis ~ .,
                           data = train,
                           method = "gbm",
                           verbose = FALSE,
                           trControl = cv_smote,
```

```
                          metric = "ROC")

final_smoteGBM <- data.frame(actual = test$diagnosis,
                         predict(modelgbm_smote, newdata = test, type = "prob"))
final_smoteGBM$predict <- ifelse(final_smoteGBM$no > 0.5, "no", "yes")
final_smoteGBM$predict <- as.factor(final_smoteGBM$predict)
cmGBM_smote <- confusionMatrix(final_smoteGBM$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_smoteGBM$predict, test$diagnosis):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
cmGBM_smote
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   9  16
##        no    4 101
##
##                Accuracy : 0.8462
##                  95% CI : (0.7724, 0.9034)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 0.98107
##
##                   Kappa : 0.3939
##
##  Mcnemar's Test P-Value : 0.01391
##
##             Sensitivity : 0.69231
##             Specificity : 0.86325
##          Pos Pred Value : 0.36000
##          Neg Pred Value : 0.96190
##              Prevalence : 0.10000
##          Detection Rate : 0.06923
##    Detection Prevalence : 0.19231
##       Balanced Accuracy : 0.77778
##
##        'Positive' Class : yes
##
```

```
modelgbm_smote
```

```
## Stochastic Gradient Boosting
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using SMOTE
##
```

```
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.8926359  0.6857143  0.8614815
##   1                  100      0.8972567  0.7523810  0.8469360
##   1                  150      0.8876463  0.7809524  0.8540741
##   2                   50      0.9009909  0.7190476  0.8432997
##   2                  100      0.8811191  0.6857143  0.8432997
##   2                  150      0.8622110  0.6523810  0.8432323
##   3                   50      0.8653423  0.7857143  0.8541414
##   3                  100      0.8688809  0.7523810  0.8431650
##   3                  150      0.8597451  0.7523810  0.8396633
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 10.
```

GBM using ROSE

```
# Using cv_rose as previously defined
set.seed(777)
modelgbm_rose <- caret::train(diagnosis ~ .,
                              data = train,
                              method = "gbm",
                              verbose = FALSE,
                              trControl = cv_rose,
                              metric = "ROC")

final_roseGBM <- data.frame(actual = test$diagnosis,
                            predict(modelgbm_rose, newdata = test, type = "prob"))
final_roseGBM$predict <- ifelse(final_roseGBM$no > 0.5, "no", "yes")
final_roseGBM$predict <- as.factor(final_roseGBM$predict)
cmGBM_rose <- confusionMatrix(final_roseGBM$predict, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_roseGBM$predict, test$diagnosis):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
cmGBM_rose
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   7  12
##        no    6 105
##
##                Accuracy : 0.8615
##                  95% CI : (0.79, 0.9158)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 0.9404
##
##                   Kappa : 0.3617
```

```
##
##  Mcnemar's Test P-Value : 0.2386
##
##             Sensitivity : 0.53846
##             Specificity : 0.89744
##          Pos Pred Value : 0.36842
##          Neg Pred Value : 0.94595
##              Prevalence : 0.10000
##          Detection Rate : 0.05385
##    Detection Prevalence : 0.14615
##       Balanced Accuracy : 0.71795
##
##        'Positive' Class : yes
##
```

```
modelgbm_rose
```

```
## Stochastic Gradient Boosting
##
## 305 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 244, 244, 243, 245, 244
## Addtional sampling using ROSE
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.8636588  0.6142857  0.8870707
##   1                  100      0.8641462  0.5523810  0.8943434
##   1                  150      0.8563428  0.5904762  0.9052525
##   2                   50      0.8689642  0.6809524  0.8979125
##   2                  100      0.8701651  0.5904762  0.9051852
##   2                  150      0.8684672  0.6523810  0.9051852
##   3                   50      0.8654449  0.6238095  0.8907071
##   3                  100      0.8705644  0.6285714  0.8907071
##   3                  150      0.8671990  0.5952381  0.8907071
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

GBM model without stratified train/test split, without stratified K-fold CV, and without sampling methods

(for comparison's sake)

```
#using cv_client as defined for the very first model
set.seed(777)
modelgbm_client <- caret::train(diagnosis ~ .,
                          data = train_client,
```

```
                          method = "gbm",
                          verbose = FALSE,
                          trControl = cv_client,
                          metric = "ROC")

final_clientGBM <- data.frame(actual = test_client$diagnosis,
                        predict(modelgbm_client, newdata = test_client, type = "prob"))
final_clientGBM$predict <- ifelse(final_clientGBM$no > 0.5, "no", "yes")
final_clientGBM$predict <- as.factor(final_clientGBM$predict)
cmGBM_imbalanced <- confusionMatrix(final_clientGBM$predict, test_client$diagnosis)
```

```
## Warning in confusionMatrix.default(final_clientGBM$predict,
## test_client$diagnosis): Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
cmGBM_imbalanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes   8   8
##        no    5 110
##
##                Accuracy : 0.9008
##                  95% CI : (0.8363, 0.9461)
##     No Information Rate : 0.9008
##     P-Value [Acc > NIR] : 0.5731
##
##                   Kappa : 0.4966
##
##  Mcnemar's Test P-Value : 0.5791
##
##             Sensitivity : 0.61538
##             Specificity : 0.93220
##          Pos Pred Value : 0.50000
##          Neg Pred Value : 0.95652
##              Prevalence : 0.09924
##          Detection Rate : 0.06107
##    Detection Prevalence : 0.12214
##       Balanced Accuracy : 0.77379
##
##        'Positive' Class : yes
##
```

```
modelgbm_client
```

```
## Stochastic Gradient Boosting
##
## 304 samples
##   6 predictor
##   2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
```

```
## Summary of sample sizes: 243, 243, 243, 243, 244, 243, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  ROC        Sens       Spec
##   1                   50      0.8621302  0.3871429  0.9791380
##   1                  100      0.8624905  0.3933333  0.9758114
##   1                  150      0.8599020  0.3961905  0.9769024
##   2                   50      0.8642601  0.4057143  0.9718182
##   2                  100      0.8626428  0.4052381  0.9659125
##   2                  150      0.8609448  0.4052381  0.9669899
##   3                   50      0.8710491  0.4085714  0.9732727
##   3                  100      0.8675523  0.3833333  0.9714276
##   3                  150      0.8665001  0.3928571  0.9699461
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

Comparing confusion matrix metrics of each of the 10 models (the focus here is on sensitivity)

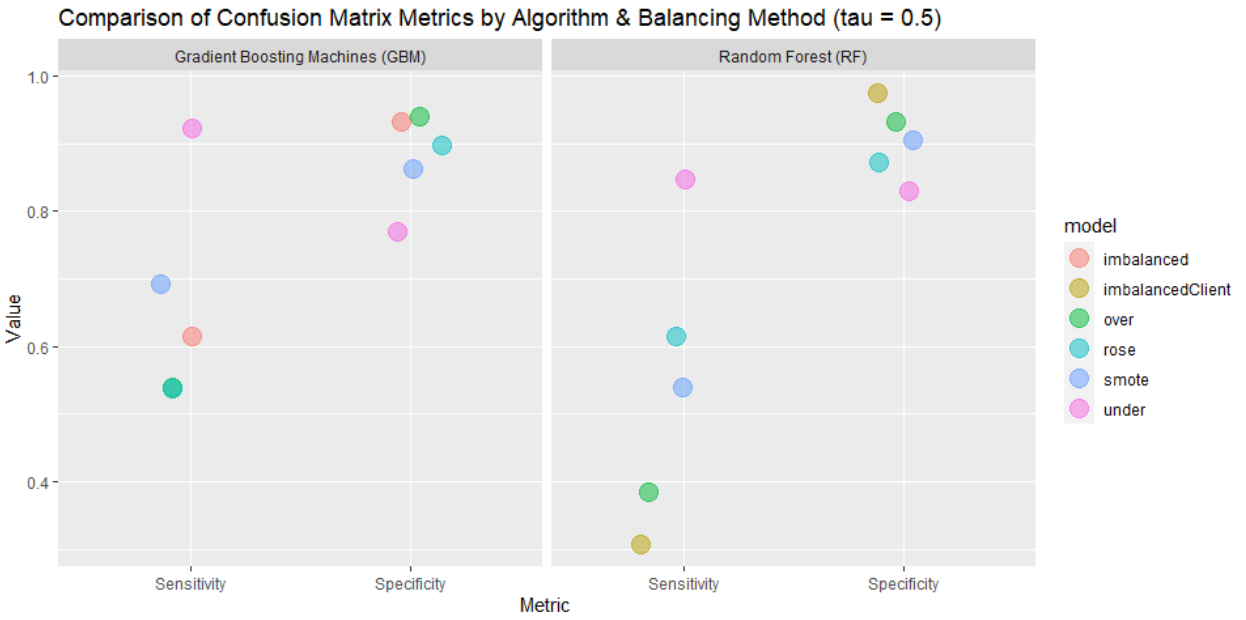
```
#RF
models <- list(imbalancedClient = modelrf_client,
               under = modelrf_us,
               over = modelrf_os,
               smote = modelrf_smote,
               rose = modelrf_rose)

comparison1 <- data.frame(model = names(models))

for (name in names(models)) {
  model <- get(paste0("cmRF_", name))

  comparison1[comparison1$model == name, "Sensitivity"] <- model$byClass[["Sensitivity"]]
  comparison1[comparison1$model == name, "Specificity"] <- model$byClass[["Specificity"]]
}

comparison1 %>%
  gather(x, y, Sensitivity:Specificity)
```

```
##              model           x         y
## 1  imbalancedClient Sensitivity 0.3076923
## 2             under Sensitivity 0.8461538
## 3              over Sensitivity 0.3846154
## 4             smote Sensitivity 0.5384615
## 5              rose Sensitivity 0.6153846
## 6  imbalancedClient Specificity 0.9745763
## 7             under Specificity 0.8290598
## 8              over Specificity 0.9316239
## 9             smote Specificity 0.9059829
## 10             rose Specificity 0.8717949
```

```
#GBM
models <- list(imbalanced = modelgbm_client,
                under = modelgbm_us,
                over = modelgbm_os,
                smote = modelgbm_smote,
                rose = modelgbm_rose)

comparison2 <- data.frame(model = names(models))

for (name in names(models)) {
  model <- get(paste0("cmGBM_", name))

  comparison2[comparison2$model == name, "Sensitivity"] <- model$byClass[["Sensitivity"]]
  comparison2[comparison2$model == name, "Specificity"] <- model$byClass[["Specificity"]]
}

comparison2 %>%
  gather(x, y, Sensitivity:Specificity)
```

```
##         model           x         y
## 1  imbalanced Sensitivity 0.6153846
## 2       under Sensitivity 0.9230769
## 3        over Sensitivity 0.5384615
## 4       smote Sensitivity 0.6923077
## 5        rose Sensitivity 0.5384615
## 6  imbalanced Specificity 0.9322034
## 7       under Specificity 0.7692308
## 8        over Specificity 0.9401709
## 9       smote Specificity 0.8632479
## 10       rose Specificity 0.8974359
```

```
#the plot
comparison1 <- cbind(comparison1,"Random Forest (RF)")
colnames(comparison1)[4] <- 'algo'
comparison2 <- cbind(comparison2,"Gradient Boosting Machines (GBM)")
colnames(comparison2)[4] <- 'algo'
both <- rbind(comparison1,comparison2)

both %>%
  gather(x, y, Sensitivity:Specificity) %>%
ggplot(aes(x=x,y=y, color = model)) +
  geom_jitter(width = 0.2, alpha = 0.5, size = 5) +
  facet_wrap(~algo) +
  labs(title="Comparison of Confusion Matrix Metrics by Algorithm & Balancing Method (tau = 0.5)") +
  labs(x="Metric",y="Value")
```

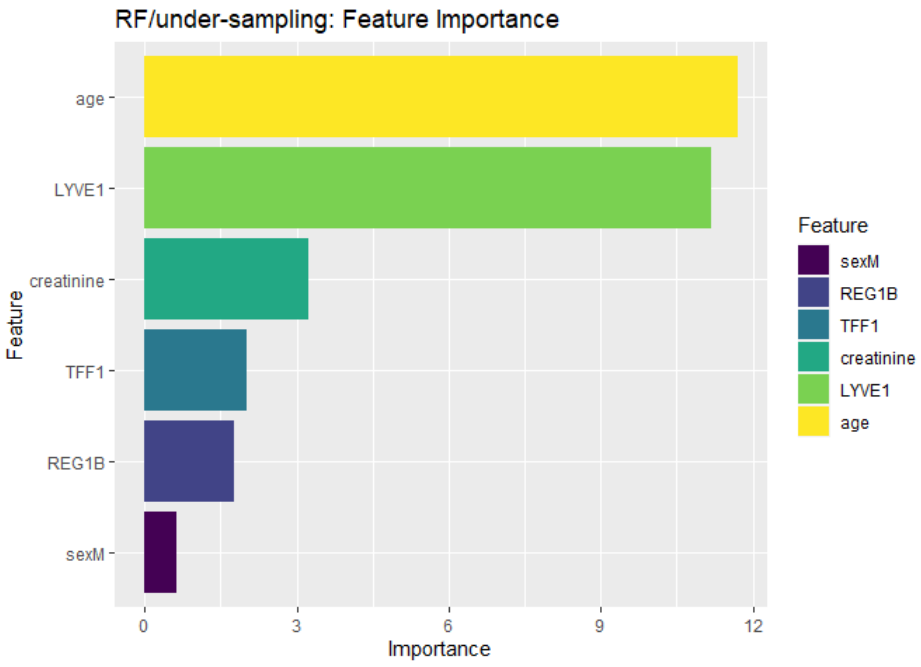


We see that under-sampling yields the highest sensitivity for the RF model, and under-sampling yields the highest sensitivity for the GBM model, but GBM/under achieves a higher sensitivity. In fact, the GBM/under model reaches the goals, since it shows a sensitivity of 0.923 and a 1-specificity of 0.23 (a specificity of 0.77), however the RF/under model does not reach a sensitivity of 0.90 or higher. We will tune the probability cutoff for both models in order to see if RF/under can meet the goals, and if it can, which of the tuned models provides the most optimal sensitivity and 1-specificity, given the client goals.

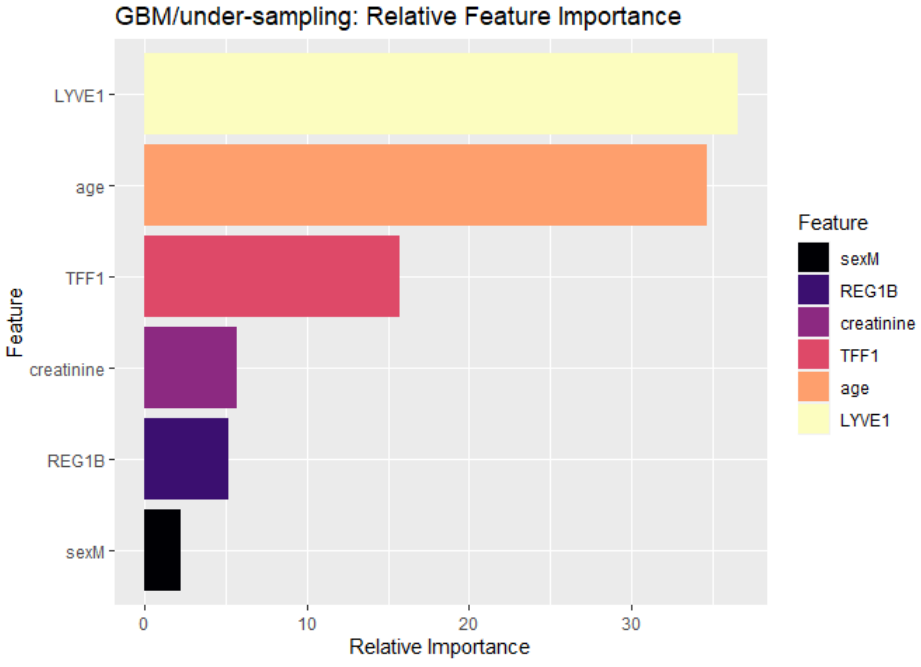
Before we try tuning these models, we can get feature importance plots for each, as these are invariable to tuning).

Check variable importance for the best models

```
# RF/Under
VI <- varImp(modelrf_us, scale=FALSE)
VI <- VI$importance
VI$feat <- row.names(VI)
VI <- transform(VI, feat = reorder(feat, Overall))
ggplot2::ggplot(VI, aes(Overall, feat, fill = feat)) +
  geom_col(aes()) +
  labs(title="RF/under-sampling: Feature Importance") +
  labs(x="Importance",y="Feature") +
    scale_fill_viridis_d(option="viridis", name = "Feature")
```




```
# GBM/Under
RI <- summary(modelgbm_us, plot = FALSE)
RI <- transform(RI, var = reorder(var, rel.inf))
ggplot2::ggplot(RI, aes(rel.inf, var, fill = var)) +
  geom_col(aes()) +
  labs(title="GBM/under-sampling: Relative Feature Importance") +
  labs(x="Relative Importance",y="Feature") +
    scale_fill_viridis_d(option="magma", name = "Feature")
```



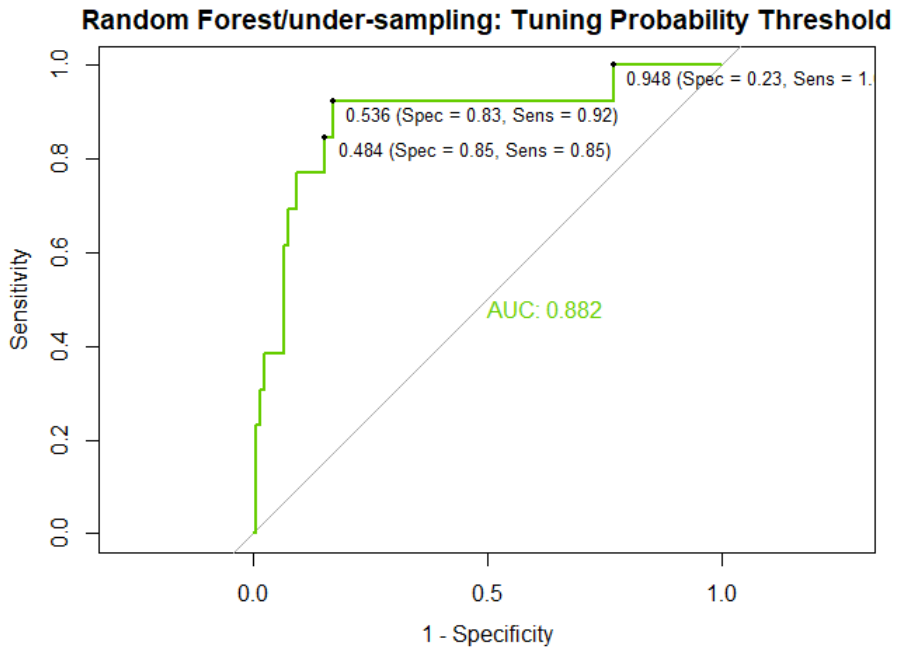
Find optimal probability thresholds for the 2 competing models

```
## Get the ROC curve for RF/under-sampling model
roc_rf <- roc(test$diagnosis,
```

```
            predict(modelrf_us, test, type = "prob")[,2],
            levels = rev(levels(test$diagnosis)))
```

```
## Setting direction: controls > cases
```

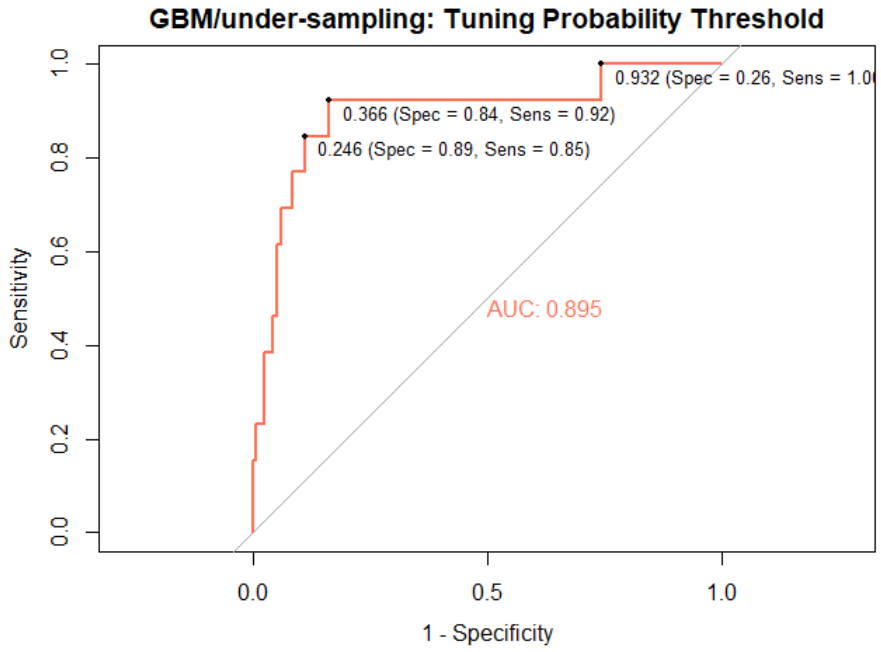
```
#roc_rf


plot(roc_rf, print.thres = c(0.484, 0.536, 0.948), type = "S", col = "chartreuse3",
     print.thres.pattern = "%.3f (Spec = %.2f, Sens = %.2f)",
     print.thres.cex = .8,
     print.auc = TRUE,
     legacy.axes = TRUE, main = "Random Forest/under-sampling: Tuning Probability Threshold")
```



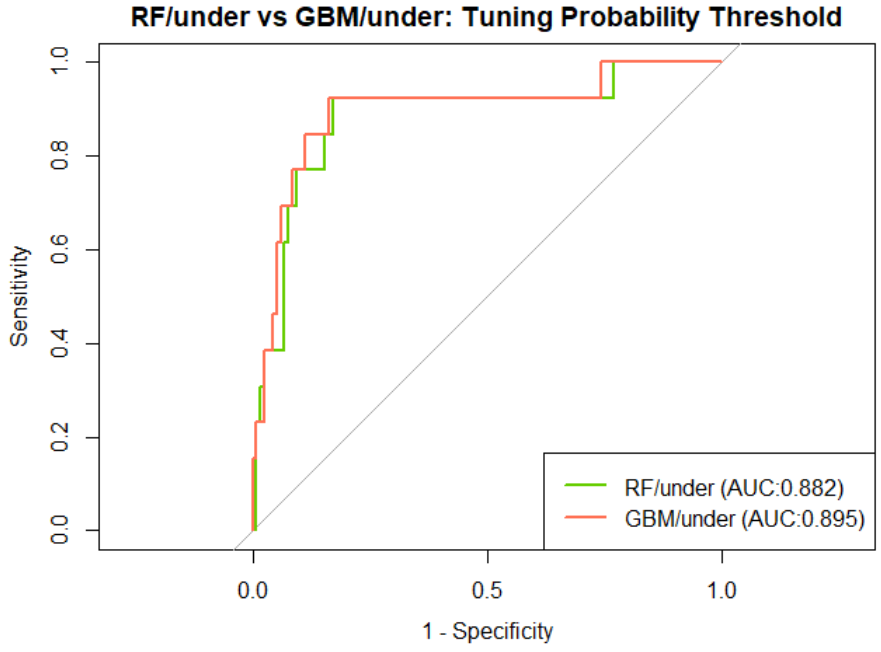

```
## Get the ROC curve for GBM/under-sampling model
roc_gbm <- roc(test$diagnosis,
            predict(modelgbm_us, test, type = "prob")[,2],
            levels = rev(levels(test$diagnosis)))
```

```
## Setting direction: controls > cases
```

```
#roc_gbm


plot(roc_gbm, print.thres = c(0.246, 0.366, 0.9322), type = "S", col = "coral1",
     print.thres.pattern = "%.3f (Spec = %.2f, Sens = %.2f)",
     print.thres.cex = .8,
     print.auc = TRUE,
     legacy.axes = TRUE, main = "GBM/under-sampling: Tuning Probability Threshold")
```




```
#plot both together
plot(roc_rf, type = "S", col = "chartreuse3",
     print.thres.pattern = "%.3f (Spec = %.2f, Sens = %.2f)",
     print.thres.cex = .8,
     legacy.axes = TRUE, main = "RF/under vs GBM/under: Tuning Probability Threshold")
lines(roc_gbm, type = "S", col = "coral1",
     print.thres.pattern = "%.3f (Spec = %.2f, Sens = %.2f)",
     print.thres.cex = .8,
     legacy.axes = TRUE, main="")
legend("bottomright", c("RF/under (AUC:0.882)","GBM/under (AUC:0.895)"),
       col=c("chartreuse3","coral1"), lty=c(1,1), lwd=c(2,2))
```



Make a table of RF/under thresholds and corresponding sensitivities & 1-specificities. Do the same for GBM/under.

```
#RF/under
TH <- roc_rf$thresholds[15:90]
SN <- roc_rf$sensitivities[15:90]
SP <- roc_rf$specificities[15:90]

df <- round(cbind(TH,SN,1-SP),3)

kable(df, col.names = c("Threshold (tau)","Sensitivity (Prop. of true +)",
                        "1 - Specificity (Prop. of false +)"),
      caption = "RF/under:  Comparing Probability Thresholds")
```

RF/under: Comparing Probability Thresholds

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.960	1.000	0.778
0.948	1.000	0.769
0.937	0.923	0.769
0.932	0.923	0.761
0.927	0.923	0.752
0.925	0.923	0.744
0.922	0.923	0.735
0.919	0.923	0.718
0.915	0.923	0.701
0.911	0.923	0.675
0.907	0.923	0.658
0.903	0.923	0.650
0.899	0.923	0.632
0.895	0.923	0.615
0.892	0.923	0.607
0.889	0.923	0.598
0.886	0.923	0.590
0.883	0.923	0.581
0.879	0.923	0.573
0.875	0.923	0.564
0.870	0.923	0.556
0.865	0.923	0.547
0.863	0.923	0.538
0.859	0.923	0.530

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.854	0.923	0.521
0.849	0.923	0.513
0.840	0.923	0.504
0.833	0.923	0.496
0.831	0.923	0.487
0.826	0.923	0.479
0.821	0.923	0.470
0.816	0.923	0.462
0.810	0.923	0.444
0.805	0.923	0.436
0.800	0.923	0.427
0.794	0.923	0.419
0.786	0.923	0.410
0.780	0.923	0.402
0.776	0.923	0.393
0.770	0.923	0.385
0.765	0.923	0.376
0.759	0.923	0.368
0.752	0.923	0.359
0.743	0.923	0.350
0.730	0.923	0.342
0.719	0.923	0.333
0.703	0.923	0.325
0.689	0.923	0.316
0.679	0.923	0.308
0.668	0.923	0.299
0.662	0.923	0.291
0.652	0.923	0.274
0.640	0.923	0.265
0.634	0.923	0.256
0.630	0.923	0.248

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.622	0.923	0.239
0.612	0.923	0.231
0.593	0.923	0.222
0.575	0.923	0.214
0.569	0.923	0.197
0.562	0.923	0.188
0.548	0.923	0.179
0.536	0.923	0.171
0.515	0.846	0.171
0.493	0.846	0.162
0.484	0.846	0.154
0.476	0.769	0.154
0.463	0.769	0.145
0.451	0.769	0.128
0.440	0.769	0.120
0.428	0.769	0.111
0.424	0.769	0.103
0.413	0.769	0.094
0.390	0.692	0.094
0.363	0.692	0.085
0.347	0.692	0.077

```
#GBM/under:
TH <- roc_gbm$thresholds[20:100]
SN <- roc_gbm$sensitivities[20:100]
SP <- roc_gbm$specificities[20:100]

df <- round(cbind(TH,SN,1-SP),3)

kable(df, col.names = c("Threshold (tau)","Sensitivity (Prop. of true +)",
                        "1 - Specificity (Prop. of false +)"),
      caption = "GBM/under:  Comparing Probability Thresholds")
```

GBM/under: Comparing Probability Thresholds

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.946	1.000	0.769
0.939	1.000	0.761

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.933	1.000	0.752
0.932	1.000	0.744
0.930	0.923	0.744
0.926	0.923	0.735
0.926	0.923	0.726
0.923	0.923	0.718
0.920	0.923	0.709
0.915	0.923	0.692
0.911	0.923	0.684
0.909	0.923	0.675
0.908	0.923	0.667
0.906	0.923	0.658
0.900	0.923	0.650
0.894	0.923	0.641
0.891	0.923	0.632
0.889	0.923	0.624
0.886	0.923	0.615
0.878	0.923	0.607
0.869	0.923	0.598
0.863	0.923	0.581
0.856	0.923	0.573
0.851	0.923	0.564
0.848	0.923	0.556
0.836	0.923	0.547
0.824	0.923	0.538
0.820	0.923	0.530
0.812	0.923	0.521
0.805	0.923	0.513
0.801	0.923	0.504
0.796	0.923	0.496
0.793	0.923	0.487

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.783	0.923	0.479
0.765	0.923	0.462
0.750	0.923	0.453
0.746	0.923	0.444
0.742	0.923	0.436
0.734	0.923	0.427
0.720	0.923	0.419
0.709	0.923	0.410
0.709	0.923	0.402
0.705	0.923	0.393
0.700	0.923	0.376
0.698	0.923	0.368
0.684	0.923	0.359
0.670	0.923	0.350
0.667	0.923	0.342
0.662	0.923	0.333
0.658	0.923	0.325
0.656	0.923	0.316
0.640	0.923	0.308
0.619	0.923	0.299
0.612	0.923	0.291
0.608	0.923	0.282
0.590	0.923	0.274
0.572	0.923	0.265
0.565	0.923	0.256
0.558	0.923	0.248
0.537	0.923	0.239
0.491	0.923	0.231
0.439	0.923	0.222
0.416	0.923	0.214
0.407	0.923	0.205

Threshold (tau)	Sensitivity (Prop. of true +)	1 - Specificity (Prop. of false +)
0.396	0.923	0.197
0.387	0.923	0.188
0.376	0.923	0.179
0.368	0.923	0.171
0.366	0.923	0.162
0.360	0.846	0.162
0.348	0.846	0.154
0.333	0.846	0.145
0.320	0.846	0.137
0.301	0.846	0.128
0.271	0.846	0.120
0.246	0.846	0.111
0.239	0.769	0.111
0.235	0.769	0.103
0.231	0.769	0.094
0.221	0.769	0.085
0.197	0.692	0.085

Comparing GBM/under sensitivities: tau = 0.5 vs. tau = 0.366

```
#using CV_smote, modelrf_smote, and final_smote as previously defined

final_usGBM$predictTuned <- ifelse(final_usGBM$no > 0.366, "no", "yes")
final_usGBM$predictTuned <- as.factor(final_usGBM$predictTuned)
cmGBM_underTuned <- confusionMatrix(final_usGBM$predictTuned, test$diagnosis)
```

```
## Warning in confusionMatrix.default(final_usGBM$predictTuned, test$diagnosis):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

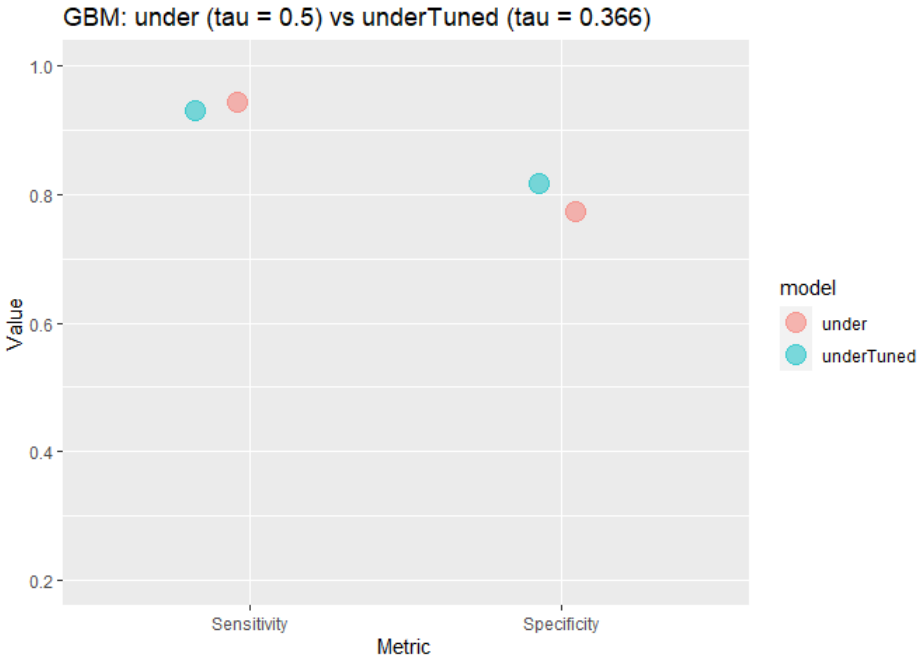
```
cmGBM_underTuned
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes no
##        yes  12 19
##        no    1 98
##
##               Accuracy : 0.8462
```

```
##                 95% CI : (0.7724, 0.9034)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 0.9810694
##
##                   Kappa : 0.4709
##
##  Mcnemar's Test P-Value : 0.0001439
##
##             Sensitivity : 0.92308
##             Specificity : 0.83761
##          Pos Pred Value : 0.38710
##          Neg Pred Value : 0.98990
##              Prevalence : 0.10000
##          Detection Rate : 0.09231
##    Detection Prevalence : 0.23846
##       Balanced Accuracy : 0.88034
##
##        'Positive' Class : yes
##
```

Plot confusion matrix metrics comparing GBM/under using tau = 0.50 vs tau = 0.366

```
# label models
models <- list(under = modelgbm_us,
               underTuned = modelgbm_us)

# data manipulation to prep for plot
comparison <- data.frame(model = names(models))

for (name in names(models)) {
  model <- get(paste0("cmGBM_", name))

  comparison[comparison$model == name, "Sensitivity"] <- model$byClass[["Sensitivity"]]
  comparison[comparison$model == name, "Specificity"] <- model$byClass[["Specificity"]]
}

# make the plot
comparison %>%
  gather(x, y, Sensitivity:Specificity) %>%
  ggplot(aes(x = x, y = y, color = model)) +
  geom_jitter(width = 0.2, alpha = 0.5, size = 5) +
  labs(title="GBM: under (tau = 0.5) vs underTuned (tau = 0.366)") +
  labs(x="Metric",y="Value") +
  ylim(0.2, 1.0)
```



Choosing the GBM/under classifier procedure, even before the probability threshold is tuned, accomplishes the stated goal of reaching true positive rate (sensitivity) of at least .90 while keeping false positive rate (1-specificity) from surpassing 0.25 (sensitivity >= 0.75). In tuning its threshold to 0.366, false positive rate was reduced to 0.162 from 0.23), while maintaining a sensitivity of 0.923. The RF/under classifier procedure can accomplish the same goal using tau = 0.536, however the RF/SMOTE classifier is recommended as it not only has a slightly higher overall predicitive performance (AUC), but it also achieves a slightly lower rate of false positives (0.162 vs. 0.171) at a sensitivity of 0.923.