Surrogate Splits with SMOTE

Contents

1 Introduction

From the previous files, we see that treating missingness on its own was not enough to get a good performance. We therefore look at how to deal with the imbalance alongisde the missingness.

In the literature, many methods have been devised to address imbalanced data. These can be grouped into 3 main categories [Chapter 5, 1]:

- **Sampling methods:** The training set is modified to produce a more balanced distribution that allows classifiers to perform in a similar manner to standard classification. These methods are sometimes called data-level modifications.
- **Algorithm-level modifications:** The classification algorithm is modified to be more attuned to class imbalance.
- **Cost-sensitive learning:** This incorporates data-level and algorithm-level modifications by considering variable misclassification costs.

We will not have time to look into all of these, so we will only look at sampling methods. We refer the reader to the book cited for more details.

2 Details behind SMOTE

We discuss one sampling method called the *Synthetic Minority Oversampling Technique* (SMOTE). It is based on the creation of synthetic data by interpolating between minority samples using their nearest neighbours (see the KNN classifier for how we define neighbours). Note that we encode each feature $\boldsymbol{x}$ so that each $x_i \in \mathbb{R}$ so that interpolation is possible. The formal process works as follows, and is adapted from [Chapter 5.4, 1]. First, an integer value N, the oversampling factor, is specified. Default behaviour in programs will usually choose this to balance the class distribution. Then, an iterative process is carried out. First, a minority class point $\boldsymbol{x}^*$ is selected at random from the training set. Next, its K nearest neighbours are obtained. Finally, N of these K instances are randomly chosen (with repetitions allowed). Then, N synthetic examples are created by random linear interpolation between $\boldsymbol{x}^*$ and each of the chosen nearest neighbours.

3 Prerequisites

We load the SMOTE data using the code below.

```
# List of required packages
packages <- c("rpart", "caret", "pROC", "ggplot2", "dplyr")

# Install missing packages
install_if_missing <- function(pkg) {
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg, dependencies = TRUE)
  }
}

# Check and install missing packages
invisible(lapply(packages, install_if_missing))
```

```
## Loading required package: rpart

## Loading required package: caret

## Loading required package: ggplot2

## Loading required package: lattice

## Loading required package: pROC

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Load the packages
lapply(packages, library, character.only = TRUE)
```

```
## [[1]]
##  [1] "dplyr"     "pROC"      "caret"     "lattice"   "ggplot2"   "rpart"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [13] "base"
##
## [[2]]
##  [1] "dplyr"     "pROC"      "caret"     "lattice"   "ggplot2"   "rpart"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [13] "base"
##
## [[3]]
##  [1] "dplyr"     "pROC"      "caret"     "lattice"   "ggplot2"   "rpart"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [13] "base"
##
## [[4]]
##  [1] "dplyr"     "pROC"      "caret"     "lattice"   "ggplot2"   "rpart"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [13] "base"
##
## [[5]]
##  [1] "dplyr"     "pROC"      "caret"     "lattice"   "ggplot2"   "rpart"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
## [13] "base"
```

```
# rpart      - Recursive partitioning for decision trees
# caret      - For confusion matrix and precision/recall calculations
# pROC       - For ROC curve calculations
# ggplot2    - For plotting
# dplyr      - For data manipulation

# Get the current working directory
current_dir <- getwd()
cat("Current directory:", current_dir, "\n")
```

```
## Current directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1/VivekP
```

```r
# Get the parent directory
parent_dir <- dirname(current_dir)
cat("Parent directory:", parent_dir, "\n")
```

```
## Parent directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1
```

```r
# Set the working directory to the parent directory
setwd(parent_dir)
cat("New working directory:", getwd(), "\n")
```

```
## New working directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1
```

```r
# Load the training and test datasets
X_train <- read.csv("data/X_train_smote.csv", row.names = 1)  # Load the SMOTE
↪  transformed training features
y_train <- read.csv("data/y_train_smote.csv", row.names = 1)  # Load the SMOTE
↪  transformed training labels
X_test <- read.csv("data/X_test.csv", row.names = 1)    # Load the test features
y_test <- read.csv("data/y_test.csv", row.names = 1)    # Load the test labels

# Combine X_train and y_train into one data frame for rpart
train_data <- cbind(X_train, y_train)

# Drop the "education" column from the training data if it exists
train_data <- train_data[, !names(train_data) %in% "education"]
```

4 Effect of SMOTE on Performance

We check the effect of SMOTE on performance using the different methods of missingness we have seen so far.

4.1 Surrogate Splits

We begin by using surrogate splits.

```r
# Fit the classification tree using rpart
fit <- rpart(income ~ ., data = train_data, method = "class", control = rpart.control(cp
↪  = 1e-6))

# Get the cost-complexity pruning table and identify the best cp based on minimum xerror
cptable <- fit$cptable
best_cp <- cptable[which.min(cptable[,"xerror"]), "CP"]
cat("Best CP:", best_cp, "\n")
```

```
## Best CP: 9.078005e-05
```

```r
# Prune the tree using the best cp
pruned_tree <- prune(fit, cp = best_cp)

# Get predicted probabilities for the positive class
pred_probs <- predict(pruned_tree, X_test, type = "prob")[, "<=50K"]  # Adjust based on
↪  your positive class

# Compute ROC curve for the model with predictions
roc_curve_model_surrogate <- roc(y_test[, 1], pred_probs, levels = c("<=50K", ">50K"))
```

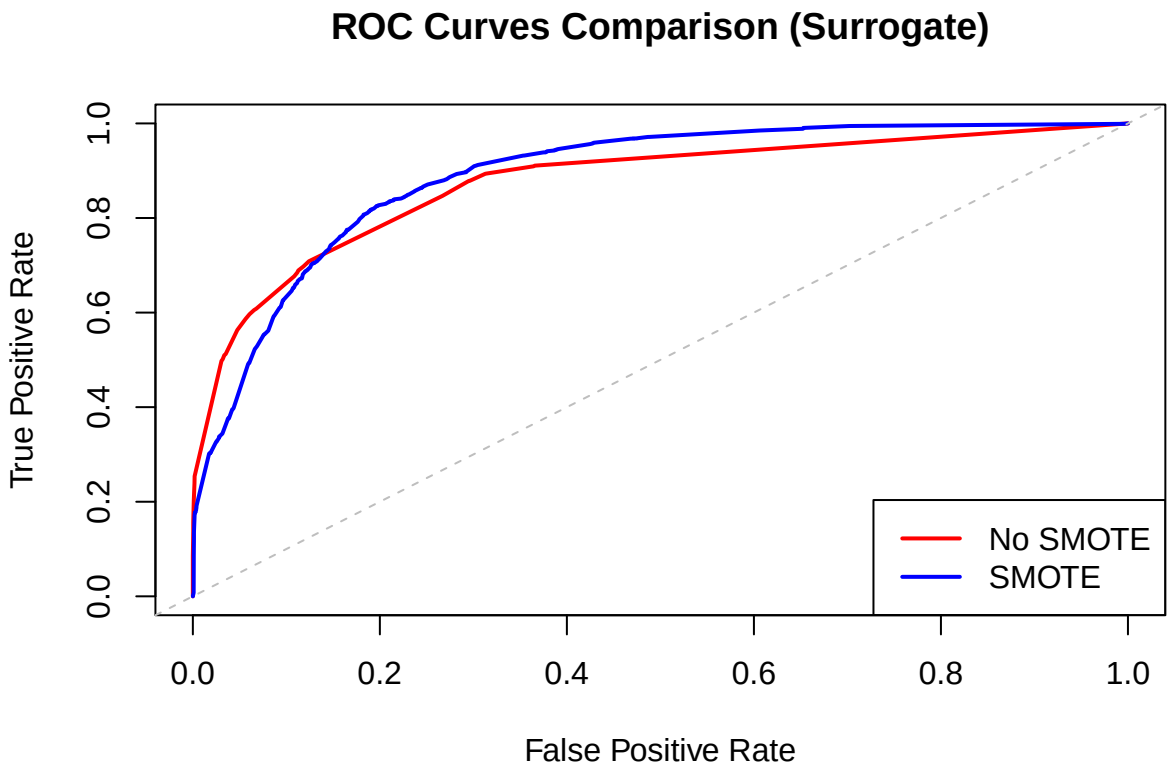
```
## Setting direction: controls > cases
```

```r
# Calculate AUC for the model
auc_value_model_surrogate <- auc(roc_curve_model_surrogate)

# Print AUC value for the model
cat("AUC for Model (Surrogate):", auc_value_model_surrogate, "\n")
```

```
## AUC for Model (Surrogate): 0.8887446
```

```r
# Create a data frame for the model's ROC data
roc_data_model_surrogate <- data.frame(
  FPR = 1 - roc_curve_model_surrogate$specificities,  # False Positive Rate
  TPR = roc_curve_model_surrogate$sensitivities         # True Positive Rate
)

# Load the saved ROC data (ensure the file name reflects "surrogate")
roc_data_saved_surrogate <- read.csv("roc_data/roc_data_model_surrogate.csv")

# Ensure the saved data has columns for FPR and TPR
if (!all(c("FPR", "TPR") %in% colnames(roc_data_saved_surrogate))) {
  stop("The saved ROC data must contain 'FPR' and 'TPR' columns.")
}

# Check and convert the columns to numeric (was getting errors)
roc_data_saved_surrogate$FPR <- as.numeric(roc_data_saved_surrogate$FPR)
roc_data_saved_surrogate$TPR <- as.numeric(roc_data_saved_surrogate$TPR)
roc_data_model_surrogate$FPR <- as.numeric(roc_data_model_surrogate$FPR)
roc_data_model_surrogate$TPR <- as.numeric(roc_data_model_surrogate$TPR)

# Create the plot for the saved ROC curve
plot(roc_data_saved_surrogate$FPR, roc_data_saved_surrogate$TPR, type = "l", col = "red",
↪  lwd = 2,
     xlim = c(0, 1), ylim = c(0, 1),
     main = "ROC Curves Comparison (Surrogate)",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")

# Add the model ROC curve to the same plot
lines(roc_data_model_surrogate$FPR, roc_data_model_surrogate$TPR, col = "blue", lwd = 2)
```

```r
# Add a diagonal line for random guessing
abline(0, 1, lty = 2, col = "grey")

# Add a legend to the plot
legend("bottomright", legend = c("No SMOTE", "SMOTE"),
       col = c("red", "blue"), lwd = 2)
```



From the previous analysis, the AUC with no SMOTE was about 0.880. Here, the AUC is 0.887, so there was a minor improvement. SMOTE can therefore be beneficial, though it was not very significant here.

4.2 Encoding Missingness as ?

We repeat the code from before, this time encoding the missingness as `?`.

```r
# Load necessary libraries
library(rpart)
library(pROC)

# Replace NAs with "?" for both training and test datasets
train_data[is.na(train_data)] <- "?"
X_test[is.na(X_test)] <- "?"

# Fit the classification tree using rpart with NA handled as "?"
fit_encoded <- rpart(income ~ ., data = train_data, method = "class", control =
↪  rpart.control(cp = 1e-6))
```

```r
# Get the cost-complexity pruning table and identify the best cp based on minimum xerror
cptable <- fit_encoded$cptable
best_cp <- cptable[which.min(cptable[, "xerror"]), "CP"]
cat("Best CP for encoded model:", best_cp, "\n")
```

```
## Best CP for encoded model: 0.0001210401
```

```r
# Prune the tree using the best cp
pruned_tree_encoded <- prune(fit_encoded, cp = best_cp)

# Get predicted probabilities for the positive class
pred_probs_encoded <- predict(pruned_tree_encoded, X_test, type = "prob")[, "<=50K"]  #
↪  Adjust based on your positive class

# Compute ROC curve for the model with predictions
roc_curve_model_encoded <- roc(y_test[, 1], pred_probs_encoded, levels = c("<=50K",
↪  ">50K"))
```

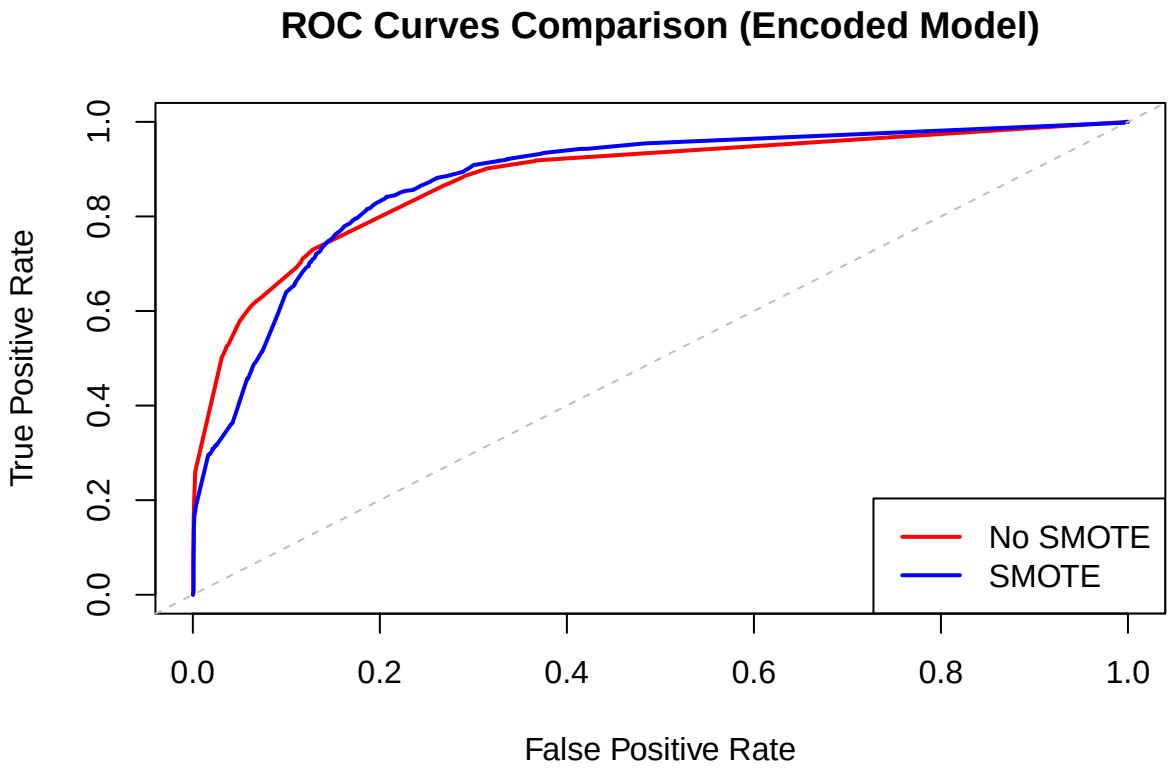
```
## Setting direction: controls > cases
```

```r
# Calculate AUC for the model
auc_value_model_encoded <- auc(roc_curve_model_encoded)

# Print AUC value for the model
cat("AUC for Model (Encoded):", auc_value_model_encoded, "\n")
```

```
## AUC for Model (Encoded): 0.8789603
```

```r
# Create a data frame for the model's ROC data
roc_data_model_encoded <- data.frame(
  FPR = 1 - roc_curve_model_encoded$specificities,  # False Positive Rate
  TPR = roc_curve_model_encoded$sensitivities         # True Positive Rate
)

# Load the saved ROC data from the encoded model
roc_data_saved_encoded <- read.csv("roc_data/roc_data_model_encoded.csv")

# Ensure the saved data has columns for FPR and TPR
if (!all(c("FPR", "TPR") %in% colnames(roc_data_saved_encoded))) {
  stop("The saved ROC data must contain 'FPR' and 'TPR' columns.")
}

# Check and convert the columns to numeric (was getting errors)
roc_data_saved_encoded$FPR <- as.numeric(roc_data_saved_encoded$FPR)
roc_data_saved_encoded$TPR <- as.numeric(roc_data_saved_encoded$TPR)
roc_data_model_encoded$FPR <- as.numeric(roc_data_model_encoded$FPR)
roc_data_model_encoded$TPR <- as.numeric(roc_data_model_encoded$TPR)

# Create the plot for the saved ROC curve
```

```r
plot(roc_data_saved_encoded$FPR, roc_data_saved_encoded$TPR, type = "l", col = "red", lwd
↪  = 2,
     xlim = c(0, 1), ylim = c(0, 1),
     main = "ROC Curves Comparison (Encoded Model)",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")

# Add the model ROC curve to the same plot
lines(roc_data_model_encoded$FPR, roc_data_model_encoded$TPR, col = "blue", lwd = 2)

# Add a diagonal line for random guessing
abline(0, 1, lty = 2, col = "grey")

# Add a legend to the plot
legend("bottomright", legend = c("No SMOTE", "SMOTE"),
       col = c("red", "blue"), lwd = 2)
```



The AUC with no SMOTE was about 0.874 and here it is 0.888. So using SMOTE gives a notable improvement in this case.

4.3 Complete Case Analysis

```r
# Load necessary libraries
library(rpart)
library(pROC)

# Remove all rows with "?" in both training and test datasets
train_data <- train_data[!apply(train_data == "?", 1, any), ]

# Fit the classification tree using rpart with NA handled as "?"
fit_deleted <- rpart(income ~ ., data = train_data, method = "class", control =
↪  rpart.control(cp = 1e-6))

# Get the cost-complexity pruning table and identify the best cp based on minimum xerror
cptable <- fit_deleted$cptable
best_cp <- cptable[which.min(cptable[, "xerror"]), "CP"]
cat("Best CP for deleted model:", best_cp, "\n")
```

```
## Best CP for deleted model: 9.903278e-05
```

```r
# Prune the tree using the best cp
pruned_tree_deleted <- prune(fit_deleted, cp = best_cp)

# Get predicted probabilities for the positive class
pred_probs_deleted <- predict(pruned_tree_deleted, X_test, type = "prob")[, "<=50K"]  #
↪  Adjust based on your positive class

# Compute ROC curve for the model with predictions
roc_curve_model_deleted <- roc(y_test[, 1], pred_probs_deleted, levels = c("<=50K",
↪  ">50K"))
```

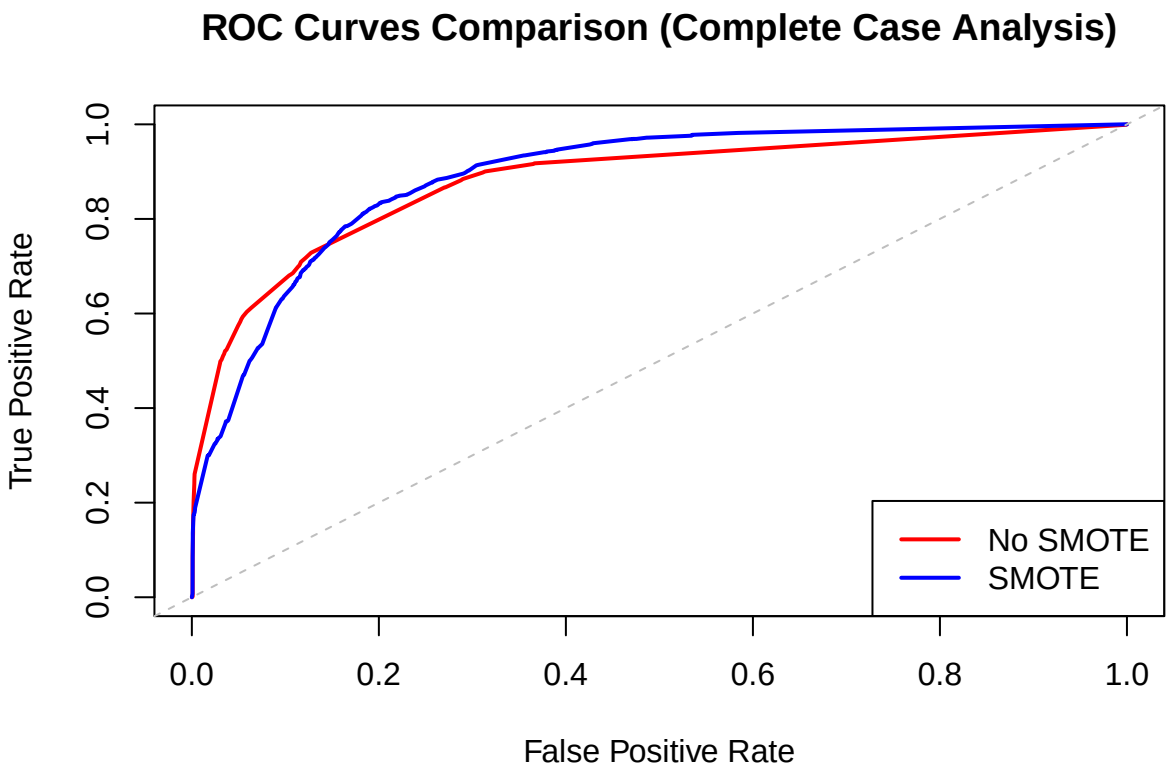
```
## Setting direction: controls > cases
```

```r
# Calculate AUC for the model
auc_value_model_deleted <- auc(roc_curve_model_deleted)

# Print AUC value for the model
cat("AUC for Model (Deleted):", auc_value_model_deleted, "\n")
```

```
## AUC for Model (Deleted): 0.8885526
```

```r
# Create a data frame for the model's ROC data
roc_data_model_deleted <- data.frame(
  FPR = 1 - roc_curve_model_deleted$specificities,  # False Positive Rate
  TPR = roc_curve_model_deleted$sensitivities         # True Positive Rate
)

# Load the saved ROC data from the deleted model
roc_data_saved_deleted <- read.csv("roc_data/roc_data_model_deleted.csv")

# Ensure the saved data has columns for FPR and TPR
if (!all(c("FPR", "TPR") %in% colnames(roc_data_saved_deleted))) {
```

```
  stop("The saved ROC data must contain 'FPR' and 'TPR' columns.")
}

# Check and convert the columns to numeric (was getting errors)
roc_data_saved_deleted$FPR <- as.numeric(roc_data_saved_deleted$FPR)
roc_data_saved_deleted$TPR <- as.numeric(roc_data_saved_deleted$TPR)
roc_data_model_deleted$FPR <- as.numeric(roc_data_model_deleted$FPR)
roc_data_model_deleted$TPR <- as.numeric(roc_data_model_deleted$TPR)

# Create the plot for the saved ROC curve
plot(roc_data_saved_deleted$FPR, roc_data_saved_deleted$TPR, type = "l", col = "red", lwd
↪  = 2,
     xlim = c(0, 1), ylim = c(0, 1),
     main = "ROC Curves Comparison (Complete Case Analysis)",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")

# Add the model ROC curve to the same plot
lines(roc_data_model_deleted$FPR, roc_data_model_deleted$TPR, col = "blue", lwd = 2)

# Add a diagonal line for random guessing
abline(0, 1, lty = 2, col = "grey")

# Add a legend to the plot
legend("bottomright", legend = c("No SMOTE", "SMOTE"),
       col = c("red", "blue"), lwd = 2)
```



The AUC from before was about 0.880 and here it is about 0.879. Here SMOTE did not make much of a difference (with a slightly worse AUC).

4.4 Imputation

We now perform mode imputation as before.

```
## Current directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1/VivekP

## Parent directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1

## New working directory: C:/Users/megar/Documents/Code/Data Science Toolbox/dst-group-project-1
```

```r
# Define function to calculate mode
get_mode <- function(x) {
  unique_x <- unique(x)
  unique_x[which.max(tabulate(match(x, unique_x)))]
}

# Function to impute missing values with mode
impute_with_mode <- function(data) {
  for (i in 1:ncol(data)) {
    if (is.factor(data[[i]]) || is.character(data[[i]])) {
      mode_value <- get_mode(data[[i]])
```

```
      data[[i]][is.na(data[[i]])] <- mode_value
    }
  }
  return(data)
}

# Impute the training and test datasets
train_data <- impute_with_mode(train_data)
X_test <- impute_with_mode(X_test)

# Fit the classification tree using rpart with mode imputation
fit_imputed <- rpart(income ~ ., data = train_data, method = "class", control =
↪  rpart.control(cp = 1e-6))

# Get the cost-complexity pruning table and identify the best cp based on minimum xerror
cptable <- fit_imputed$cptable
best_cp <- cptable[which.min(cptable[, "xerror"]), "CP"]
cat("Best CP for imputed model:", best_cp, "\n")
```

```
## Best CP for imputed model: 9.078005e-05
```

```
# Prune the tree using the best cp
pruned_tree_imputed <- prune(fit_imputed, cp = best_cp)

# Get predicted probabilities for the positive class
pred_probs_imputed <- predict(pruned_tree_imputed, X_test, type = "prob")[, "<=50K"]  #
↪  Adjust based on your positive class

# Compute ROC curve for the model with predictions
roc_curve_model_imputed <- roc(y_test[, 1], pred_probs_imputed, levels = c("<=50K",
↪  ">50K"))
```

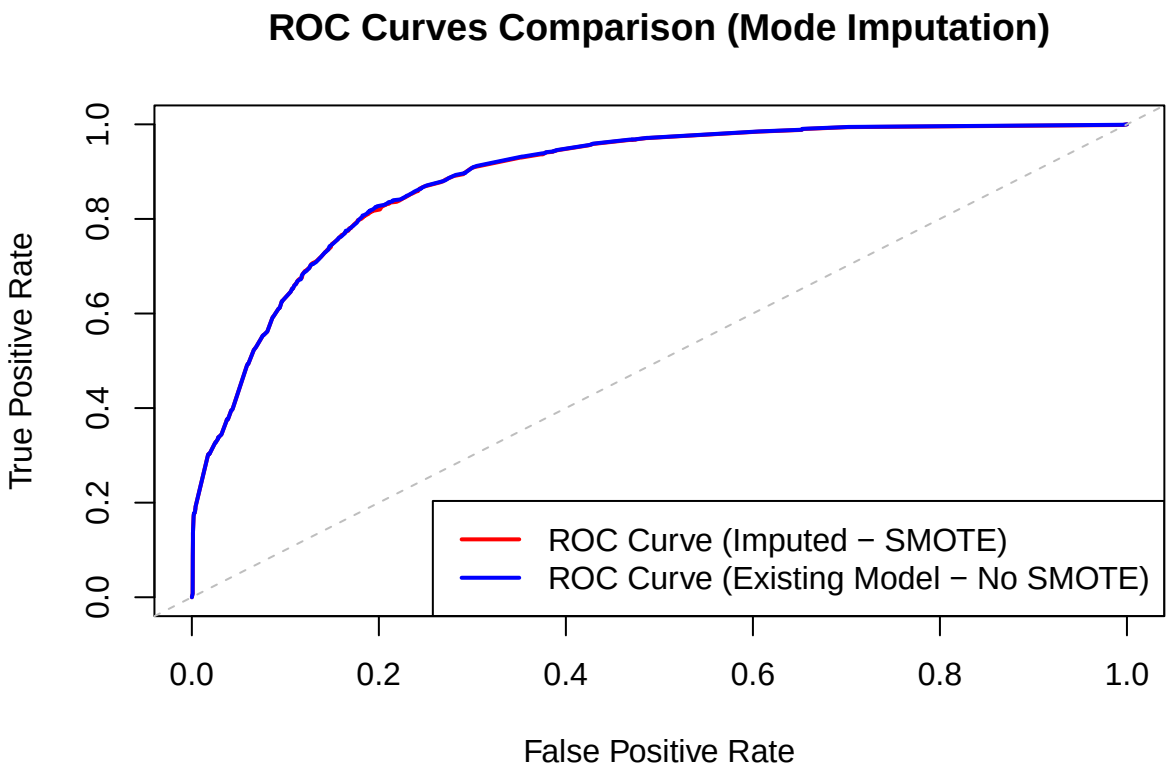
```
## Setting direction: controls > cases
```

```
# Calculate AUC for the model
auc_value_model_imputed <- auc(roc_curve_model_imputed)

# Print AUC value for the model
cat("AUC for Model (Imputed):", auc_value_model_imputed, "\n")
```

```
## AUC for Model (Imputed): 0.8887446
```

```
# Create a data frame for the model's ROC data
roc_data_model_imputed <- data.frame(
  FPR = 1 - roc_curve_model_imputed$specificities,  # False Positive Rate
  TPR = roc_curve_model_imputed$sensitivities         # True Positive Rate
)

# Load the saved ROC data from the imputed model (for SMOTE)
roc_data_saved_imputed <- read.csv("roc_data/roc_data_model_imputed.csv")
```

```r
# Ensure the saved data has columns for FPR and TPR
if (!all(c("FPR", "TPR") %in% colnames(roc_data_saved_imputed))) {
  stop("The saved ROC data must contain 'FPR' and 'TPR' columns.")
}

# Convert the columns to numeric
roc_data_saved_imputed$FPR <- as.numeric(roc_data_saved_imputed$FPR)
roc_data_saved_imputed$TPR <- as.numeric(roc_data_saved_imputed$TPR)
roc_data_model_imputed$FPR <- as.numeric(roc_data_model_imputed$FPR)
roc_data_model_imputed$TPR <- as.numeric(roc_data_model_imputed$TPR)

# Create the plot for the saved ROC curve
plot(roc_data_saved_imputed$FPR, roc_data_saved_imputed$TPR, type = "l", col = "red", lwd
↪  = 2,
     xlim = c(0, 1), ylim = c(0, 1),
     main = "ROC Curves Comparison (Mode Imputation)",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")

# Add the model ROC curve to the same plot
lines(roc_data_model_imputed$FPR, roc_data_model_imputed$TPR, col = "blue", lwd = 2)

# Add a diagonal line for random guessing
abline(0, 1, lty = 2, col = "grey")

# Add a legend to the plot
legend("bottomright", legend = c("ROC Curve (Imputed - SMOTE)", "ROC Curve (Existing
↪  Model - No SMOTE)"),
       col = c("red", "blue"), lwd = 2)
```



We obtain an AUC of about 0.887, compared to around 0.874 before. This is a notable improvement.

5 Conclusion

We saw SMOTE can be beneficial, but the performance increase was not very big. We will pursue this idea further when considering ensembles, and check whether SMOTE makes a bigger difference there.

6 References

[1] Fernández, Alberto, et al. Learning from imbalanced data sets. Vol. 10. No. 2018. Cham: Springer, 2018.