Surrogate Splits with SMOTE

Contents

1 Introduction

We got a very weak specificity in the previous model. Treating missingness on its own was not enough to get a good performance. We therefore look at how to deal with the imbalance.

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/report
```

```
# 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
```

```
# 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
```

```
# 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 Model Fitting

We fit the model and use cost-complexity pruning.

```
# 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: 8.252732e-05
```

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

5 Model Evaluation

We now look at the ROC curve.

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

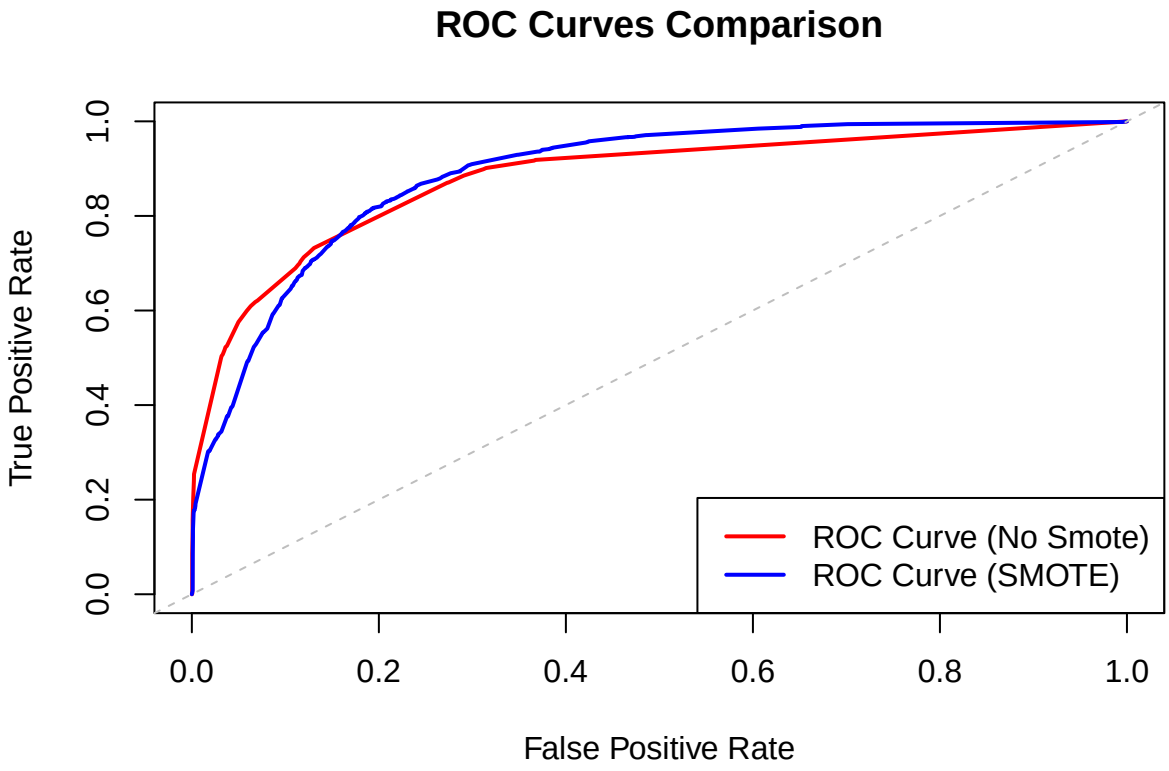
```r
# Compute ROC curve for the model with predictions
roc_curve_model <- roc(y_test[, 1], pred_probs, levels = c("<=50K", ">50K"))
```

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

```r
# Calculate AUC for the model
auc_value_model <- auc(roc_curve_model)

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

```
## AUC for Model: 0.8884168
```

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

# Load the saved ROC data
roc_data_saved <- read.csv("roc_data_no_smote.csv")

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

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

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

# Add the model ROC curve to the same plot
lines(roc_data_model$FPR, roc_data_model$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 (No Smote)", "ROC Curve (SMOTE)"),
       col = c("red", "blue"), lwd = 2)
```



The AUC with no SMOTE was 0.8789168. Here, the AUC is 0.8867056, so there was a minor improvement. SMOTE can therefore be beneficial, though it was not very significant in our case.

6 Upcoming Content

We saw SMOTE can be beneficial. Therefore, we change focus and look at other methods of dealing with missing values, in combination with SMOTE. We leave this to the next section.

7 References

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