Handling Class Imbalance in Random Forest Using Resampling and Cost-Sensitive Learning

Haozhe (Howard) Zeng

Handling Class Imbalance in Random Forest Using Resampling and Cost-Sensitive Learning

Class imbalance is a common challenge in machine learning, particularly in classification tasks. This document demonstrates the use of **resampling techniques** (e.g., oversampling and undersampling) and **cost-sensitive learning methods** to address imbalanced data in random forest models.

Key Topics Covered:

1. **Bagging-Based Methods**:
   - SMOTEBagging, RUSBagging, ROSBagging, Random Balance Bagging (RBBagging)
2. **Boosting-Based Methods**:
   - SMOTEBoost, RUSBoost, AdaBoost, Cost-Sensitive AdaBoost (AdaC2)
3. **Specialized Ensemble Methods**:
   - EasyEnsemble, BalanceCascade
4. **Hybrid Methods**:
   - SMOTETomek (SMOTE combined with Tomek link removal)

Required Libraries

Before proceeding, ensure the necessary packages are installed and loaded:

```r
# Load dataset
data("PimaIndiansDiabetes")
pima <- PimaIndiansDiabetes

# Prepare features and labels
x <- pima %>% select(-diabetes)
y <- as.factor(ifelse(pima$diabetes == "pos", 1, 0))
table(y) # Check class distribution
```

```
## y
##   0   1
## 500 268
```

Bagging-Based Methods (bbaging)

The `bbaging` function implements bagging-based resampling methods, including:

- Random Under-Sampling (RUSBagging)

- Random Over-Sampling (ROSBagging)
- SMOTE (Synthetic Minority Oversampling Technique) Bagging
- Random Balance Bagging (RBBagging)

Parameters for `bbaging`:

- `x`: A data frame containing the predictor variables.
- `y`: A factor representing the response variable.
- `numBag`: The number of bagging iterations to perform. Default is `10`.
- `type`: The type of bagging method to use. Options include:
  - `"SMOTEBagging"`: Uses SMOTE for oversampling.
  - `"RUSBagging"`: Applies random undersampling.
  - `"ROSBagging"`: Performs random oversampling.
  - `"RBBagging"`: Uses random balance bagging.

Example: SMOTEBagging

```r
# Train SMOTEBagging model
model <- bbaging(x, y, numBag = 10, type = "SMOTEBagging")

# Predictions
predictions_prob <- predict(model, x, type = "probability")[, 2]
predictions_label <- ifelse(predictions_prob > 0.5, 1, 0)

# Calculate metrics
metrics <- calculate_metrics(y, predictions_label, predictions_prob)
print(metrics)
```

```
## $ConfusionMatrix
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 405  43
##          1  95 225
##
##                Accuracy : 0.8203
##                  95% CI : (0.7913, 0.8468)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6216
##
##  Mcnemar's Test P-Value : 1.416e-05
##
##             Sensitivity : 0.8100
##             Specificity : 0.8396
##          Pos Pred Value : 0.9040
##          Neg Pred Value : 0.7031
##              Prevalence : 0.6510
##          Detection Rate : 0.5273
##    Detection Prevalence : 0.5833
##       Balanced Accuracy : 0.8248
##
##        'Positive' Class : 0
##
```

```
##
## $Accuracy
## [1] 0.8203125
##
## $WeightedAccuracy
## [1] 0.8035714
##
## $Precision
## [1] 0.8395522
##
## $Recall
## [1] 0.703125
##
## $F1
## [1] 0.7653061
##
## $Specificity
## [1] 0.9040179
##
## $GMean
## [1] 0.7972688
##
## $ROCAUC
## Area under the curve: 0.8874
```

Boosting-Based Methods (bboost)

The `bboost` function applies boosting with resampling or cost-sensitive approaches, such as:

- AdaBoost
- SMOTEBoost
- RUSBoost
- Cost-Sensitive AdaBoost (AdaC2)

Parameters for `bboost`:

- **`x`**: A data frame containing the predictor variables.
- **`y`**: A factor representing the response variable.
- **`iter`**: The number of boosting iterations. Default is `20`.
- **`type`**: The type of boosting method to use. Options include:
  - `"AdaBoost"`: Standard AdaBoost.
  - `"SMOTEBoost"`: Combines boosting with SMOTE.
  - `"RUSBoost"`: Combines boosting with random undersampling.
  - `"AdaC2"`: Cost-sensitive AdaBoost.

Example: SMOTEBoost

```r
# Train SMOTEBoost model
model <- bboost(x, y, iter = 20, type = "SMOTEBoost")

# Predictions
predictions_prob <- predict(model, x, type = "probability")[, 2]
predictions_label <- ifelse(predictions_prob > 0.5, 1, 0)

# Calculate metrics
```

```r
metrics <- calculate_metrics(y, predictions_label, predictions_prob)
print(metrics)
```

```
## $ConfusionMatrix
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 484  13
##          1  16 255
##
##                Accuracy : 0.9622
##                  95% CI : (0.9462, 0.9746)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9171
##
##  Mcnemar's Test P-Value : 0.7103
##
##             Sensitivity : 0.9680
##             Specificity : 0.9515
##          Pos Pred Value : 0.9738
##          Neg Pred Value : 0.9410
##              Prevalence : 0.6510
##          Detection Rate : 0.6302
##    Detection Prevalence : 0.6471
##       Balanced Accuracy : 0.9597
##
##        'Positive' Class : 0
##
##
## $Accuracy
## [1] 0.9622396
##
## $WeightedAccuracy
## [1] 0.9574012
##
## $Precision
## [1] 0.9514925
##
## $Recall
## [1] 0.9409594
##
## $F1
## [1] 0.9461967
##
## $Specificity
## [1] 0.9738431
##
## $GMean
## [1] 0.95726
##
## $ROCAUC
```

```
## Area under the curve: 0.9959
```

EasyEnsemble

EasyEnsemble creates multiple balanced datasets by undersampling the majority class and training individual classifiers.

Parameters for `EasyEnsemble`:

- **`x`**: A data frame containing the predictor variables.
- **`y`**: A factor representing the response variable.
- **`iter`**: The number of ensemble iterations. Default is `4`.
- **`allowParallel`**: A logical indicating whether to enable parallel computation. Default is `FALSE`.

Example: EasyEnsemble

```r
# Train EasyEnsemble model
model <- EasyEnsemble(x, y, iter = 4)

# Predictions
predictions_prob <- predict(model, x, type = "probability")[, 2]
predictions_label <- ifelse(predictions_prob > 0.5, 1, 0)

# Calculate metrics
metrics <- calculate_metrics(y, predictions_label, predictions_prob)
print(metrics)
```

```
## $ConfusionMatrix
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 456   0
##          1  44 268
##
##                Accuracy : 0.9427
##                  95% CI : (0.9238, 0.9581)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8785
##
##  Mcnemar's Test P-Value : 9.022e-11
##
##             Sensitivity : 0.9120
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8590
##              Prevalence : 0.6510
##          Detection Rate : 0.5938
##    Detection Prevalence : 0.5938
##       Balanced Accuracy : 0.9560
##
##        'Positive' Class : 0
##
```

```
##
## $Accuracy
## [1] 0.9427083
##
## $WeightedAccuracy
## [1] 0.9294872
##
## $Precision
## [1] 1
##
## $Recall
## [1] 0.8589744
##
## $F1
## [1] 0.9241379
##
## $Specificity
## [1] 1
##
## $GMean
## [1] 0.9268087
##
## $ROCAUC
## Area under the curve: 0.9958
```

Balance Cascade

Balance Cascade iteratively trains classifiers while removing easy-to-classify majority instances.

Parameters for `BalanceCascade`:

- **`x`**: A data frame containing the predictor variables.
- **`y`**: A factor representing the response variable.
- **`iter`**: The number of cascade iterations. Default is 4.

Example: Balance Cascade

```r
# Train BalanceCascade model
model <- BalanceCascade(x, y, iter = 4)

# Predictions
predictions_prob <- predict(model, x, type = "probability")[, 2]
predictions_label <- ifelse(predictions_prob > 0.5, 1, 0)

# Calculate metrics
metrics <- calculate_metrics(y, predictions_label, predictions_prob)
print(metrics)
```

```
## $ConfusionMatrix
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 483   0
##          1  17 268
```

```
##
##                Accuracy : 0.9779
##                  95% CI : (0.9648, 0.9871)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.952
##
##  Mcnemar's Test P-Value : 0.0001042
##
##             Sensitivity : 0.9660
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9404
##              Prevalence : 0.6510
##          Detection Rate : 0.6289
##    Detection Prevalence : 0.6289
##       Balanced Accuracy : 0.9830
##
##        'Positive' Class : 0
##
##
## $Accuracy
## [1] 0.9778646
##
## $WeightedAccuracy
## [1] 0.9701754
##
## $Precision
## [1] 1
##
## $Recall
## [1] 0.9403509
##
## $F1
## [1] 0.9692586
##
## $Specificity
## [1] 1
##
## $GMean
## [1] 0.9697169
##
## $ROCAUC
## Area under the curve: 1
```

Hybrid Methods: SMOTETomek

SMOTETomek combines SMOTE oversampling with Tomek link removal for better balancing of the dataset.

Parameters for `SMOTETomek`:

- `x`: A data frame containing the predictor variables.
- `y`: A factor representing the response variable.

- **percOver**: The percentage of oversampling to apply. Default is `100`.
- **k**: The number of nearest neighbors to use in SMOTE. Default is `5`.

Example: SMOTETomek

```r
# Plot original class distribution
print("Before")
```

```
## [1] "Before"
```

```r
table(y)
```

```
## y
##   0   1
## 500 268
```

```r
# Apply SMOTETomek
balanced_data <- SMOTETomek(x, y, percOver = 100)

# Plot new class distribution
print("After")
```

```
## [1] "After"
```

```r
table(balanced_data$y)
```

```
##
##   0   1
## 472 508
```