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:

```
# 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)

Example: SMOTEBagging

```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
```

```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
##          0 399  45
##          1 101 223
##
##                Accuracy : 0.8099
##                  95% CI : (0.7803, 0.8371)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.601
##
##  Mcnemar's Test P-Value : 5.318e-06
##
##             Sensitivity : 0.7980
##             Specificity : 0.8321
##          Pos Pred Value : 0.8986
##          Neg Pred Value : 0.6883
##              Prevalence : 0.6510
##          Detection Rate : 0.5195
##    Detection Prevalence : 0.5781
##       Balanced Accuracy : 0.8150
```

```
##
##        'Positive' Class : 0
##
##
## $Accuracy
## [1] 0.8098958
##
## $WeightedAccuracy
## [1] 0.7934601
##
## $Precision
## [1] 0.8320896
##
## $Recall
## [1] 0.6882716
##
## $F1
## [1] 0.7533784
##
## $Specificity
## [1] 0.8986486
##
## $GMean
## [1] 0.7864568
##
## $ROCAUC
## Area under the curve: 0.8973
```

Boosting-Based Methods (bboost)

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

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

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

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

```
##          0 487  18
##          1  13 250
##
##                Accuracy : 0.9596
##                  95% CI : (0.9432, 0.9724)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9108
##
##  Mcnemar's Test P-Value : 0.4725
##
##             Sensitivity : 0.9740
##             Specificity : 0.9328
##          Pos Pred Value : 0.9644
##          Neg Pred Value : 0.9506
##              Prevalence : 0.6510
##          Detection Rate : 0.6341
##    Detection Prevalence : 0.6576
##       Balanced Accuracy : 0.9534
##
##        'Positive' Class : 0
##
##
## $Accuracy
## [1] 0.9596354
##
## $WeightedAccuracy
## [1] 0.9574634
##
## $Precision
## [1] 0.9328358
##
## $Recall
## [1] 0.9505703
##
## $F1
## [1] 0.9416196
##
## $Specificity
## [1] 0.9643564
##
## $GMean
## [1] 0.9574386
##
## $ROCAUC
## Area under the curve: 0.993
```

EasyEnsemble

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

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: 0.9953
```

Balance Cascade

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

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
##          0 487   0
##          1  13 268
##
##                Accuracy : 0.9831
##                  95% CI : (0.9712, 0.991)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9632
##
##  Mcnemar's Test P-Value : 0.0008741
##
##             Sensitivity : 0.9740
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9537
##              Prevalence : 0.6510
##          Detection Rate : 0.6341
##    Detection Prevalence : 0.6341
##       Balanced Accuracy : 0.9870
##
##        'Positive' Class : 0
##
```

```
##
## $Accuracy
## [1] 0.9830729
##
## $WeightedAccuracy
## [1] 0.9768683
##
## $Precision
## [1] 1
##
## $Recall
## [1] 0.9537367
##
## $F1
## [1] 0.9763206
##
## $Specificity
## [1] 1
##
## $GMean
## [1] 0.9765944
##
## $ROCAUC
## Area under the curve: 1
```

Hybrid Methods: SMOTETomek

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

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
## 470 506
```