Over_sampling_for_time_series_classification

Lan Wei

2017-08-21

Introduction

The package provides functions to deal with binary classification problems in time series imbalanced classes. Synthetic balanced samples are generated by approaches ESPO and ADASYN.

ESPO is short for enhanced structure preserving oversampling, which is synergistically combined with interpolation-based oversampling ADASYN to form OSTSC. "ESPO is used to generate a large percentage of the synthetic minority samples based on multivariate Gaussian distribution, by estimating the covariance structure of the minority-class samples and by regularizing the unreliable eigen spectrum."[1]

Given the positive data $P = \left\{x_{11}, x_{12}, ..., x_{1|P|}\right\}$ and the negative data $N = \left\{x_{01}, x_{02}, ..., x_{0|N|}\right\}$, where $|N| \gg |P|$, $x_{ij} \in \mathbb{R}^{n \times 1}$, the new samples will be generated in the following steps.

1. Removal of Common Null Space

Using $q_{ij} = L_s^T x_{ij}$ to represent x_{ij} in a lower-dimensional signal space, where L_s consists of eigenvectors in the signal space.

2. ESPO

Given $\hat{D}$ is the diagonal matrix of regularized eigen values $\left\{\hat{d_1}, ..., \hat{d_n}\right\}$, V is the eigenvector matrix from the positive-class covariance matrix, $\hat{F} = V\hat{D}^{-1/2}$, $\bar{q_1}$ is the corresponding positive-class mean vector, $z = \hat{F}(b - \bar{q_1})$, the sample in the signal space is computed by $b = \hat{D}^{1/2}V^T z + \bar{q_1}$

3. ADASYN

ADASYN is executed in the transformed signal for the efficiency. Given the transformed positive data $P_t = \{q_{1i}\}$ and negative data $N_t = \{q_{0j}\}$, each sample q_{1i} is duplicated in ratio $\Gamma_i = \left|S_{i:k-NN} \bigcap N_t\right| / Z$, where $S_{i:k-NN}$ is this sample's kNN in the entire dataset, Z is a normalization factor to make $\sum_{i=1}^{|P_t|} \Gamma_i = 1$.

More details about the theories can be read in the referenced article.

References

```
H. Cao, X.-L. Li, Y.-K. Woon and S.-K. Ng,
"Integrated Oversampling for Imbalanced Time Series Classification"
IEEE Trans. on Knowledge and Data Engineering (TKDE),
vol. 25(12), pp. 2809-2822, 2013
```

Examples

First of all, we need to load some data for oversampling. The synthetic_control_TRAIN and synthetic_control_test are two datasets included in the OSTSC package. After loading the datasets, it will be helpful to split the sample sequences and labels at once.

[1] H. Cao, X.-L. Li, Y.-K. Woon and S.-K. Ng, Integrated Oversampling for Imbalanced Time Series Classification. 2013

```
library(OSTSC)
data(synthetic_control_TRAIN)
data(synthetic_control_TEST)

train_label <- synthetic_control_TRAIN[, c(1)]
train_sample <- synthetic_control_TRAIN[, -1]
test_label <- synthetic_control_TEST[, c(1)]
test_sample <- synthetic_control_TEST[, -1]
```

The imbalance of synthetic_control data is 1:5. Its time series sequences record body moving sensor data. Class 1 aims to Normal status, while class 0 aims to Cyclic, Increasing trend, Decreasing trend, Upward shift and Downward shift. The imbalance of the train dataset is shown below.

```
table(train_label)
```

```
## train_label
##   0   1
## 250  50
```

This is a simplest example to show how to oversample the class 1 to the same amount of class 0, and export the sample and label from oversampled data. There are ten parameters in the OSTSC function, the details of them can be found in the help documents. Users only need to input the first three parameters to be able to call the function.

```
MyData <- OSTSC(train_sample, train_label, target_class = 1)
over_sample <- MyData$sample
over_label <- MyData$label
```

Now the positive data and negative data are balanced. Let's check the (im)balance of new dataset.

```
table(over_label)
```

```
## over_label
##   0   1
## 250 250
```

Here a multi-layer perceptron is used as the classifier to show the performance of the OSTSC approach. In R package mxnet, there is a function called mx.mlp for building a general multi-layer neural network to do classification. For comparison, first we get how does the classifier perform on the original data before oversampling.

```
mx.set.seed(0)
model <- mx.mlp(as.matrix(train_sample), train_label, out_activation="softmax",
                hidden_node=10, out_node=2, num.round=10, array.batch.size=15,
                learning.rate=0.07, momentum=0.9, eval.metric=mx.metric.accuracy)
preds <- predict(model, as.matrix(test_sample))
pred_label <- max.col(t(preds))-1
cm <- table(pred_label, test_label)
```

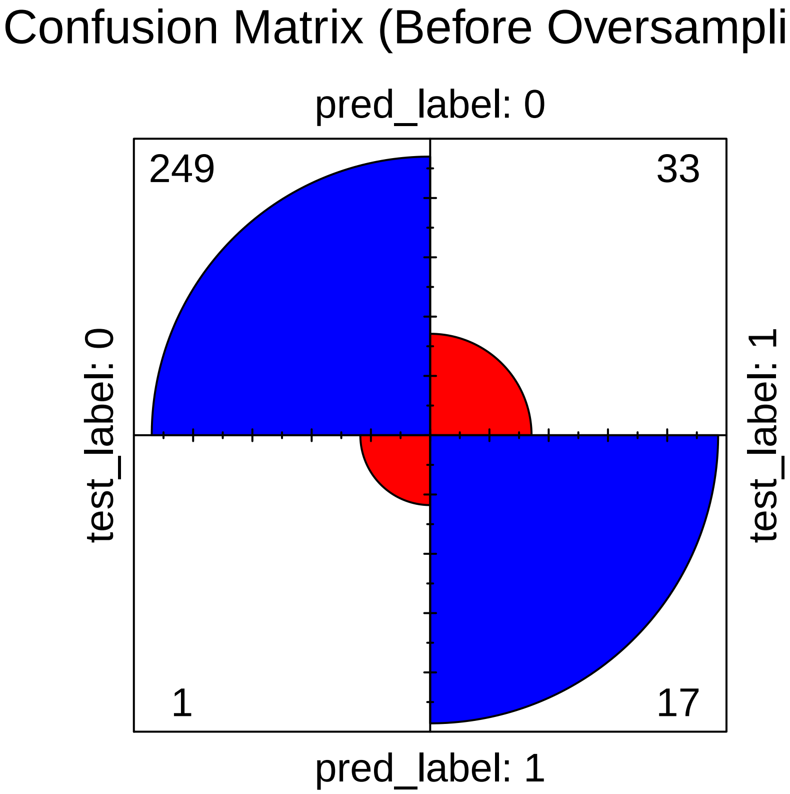
Then we get how does the classifier perform on the new data after oversampling.

```
mx.set.seed(0)
model_over <- mx.mlp(over_sample, over_label, out_activation="softmax", hidden_node=10,
                     out_node=2, num.round=10, array.batch.size=15, learning.rate=0.07,
                     momentum=0.9, eval.metric=mx.metric.accuracy)
preds_over <- predict(model_over, as.matrix(test_sample))
pred_label_over <- max.col(t(preds_over))-1
cm_over <- table(pred_label_over, test_label)
```

The performances are compared in three mothods, confusion matrix, auc and roc plot.

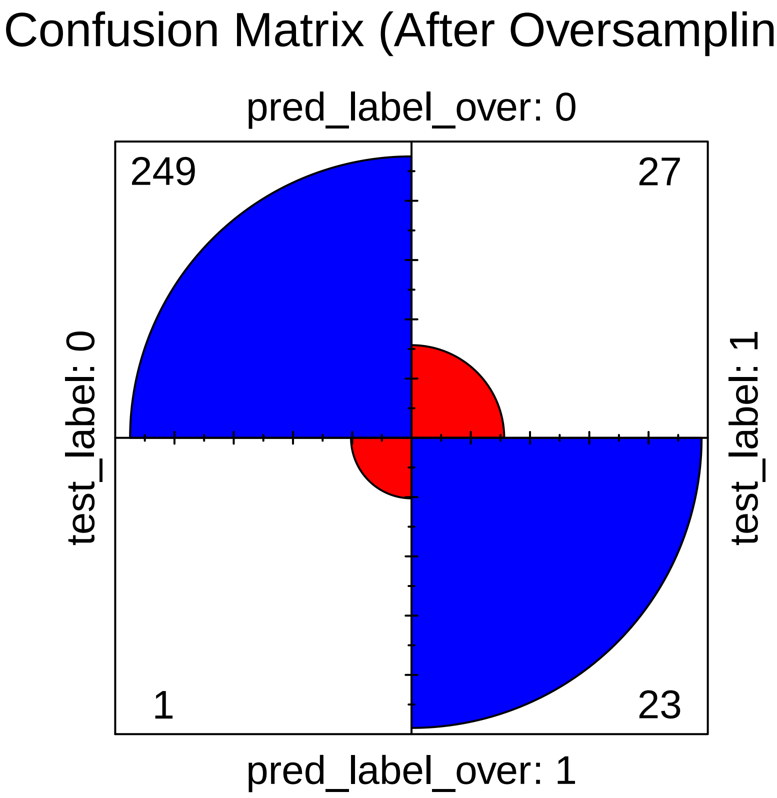
1. From the confusion matrices, the mis-classification rate obviously decreases after oversampling.

```
fourfoldplot(cm, color = c("#FF0000", "#0000FF"), conf.level = 0, margin = 1,
             main = "Confusion Matrix (Before Oversampling)")
```



```
##            test_label
## pred_label   0   1
##          0 249  33
##          1   1  17
```

```
fourfoldplot(cm_over, color = c("#FF0000", "#0000FF"), conf.level = 0, margin = 1,
             main = "Confusion Matrix (After Oversampling)")
```



```
##                 test_label
## pred_label_over   0   1
##               0 249  27
##               1   1  23
```

2. The auc value becomes larger after oversampling.

```r
auc(roc(test_label, pred_label))
```

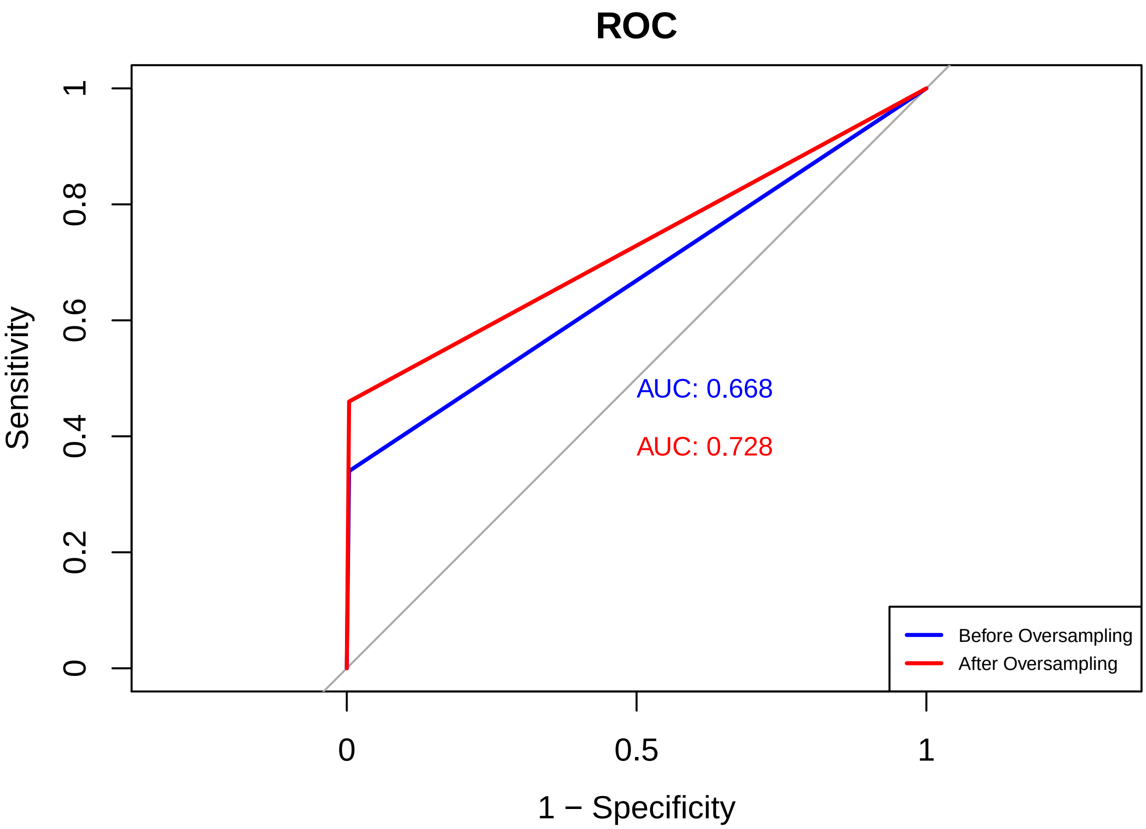
```
## Area under the curve: 0.668
```

```r
auc(roc(test_label, pred_label_over))
```

```
## Area under the curve: 0.728
```

3. The ROC plot tells the same.

```r
plot.roc(test_label, pred_label, legacy.axes = TRUE, col = "blue", print.auc = TRUE,
         main="ROC", print.auc.cex= .8)
plot.roc(test_label, pred_label_over, legacy.axes = TRUE, col = "red", print.auc = TRUE,
         print.auc.y = .4, print.auc.cex= .8, add = TRUE)
legend("bottomright", legend=c("Before Oversampling", "After Oversampling"),
       col=c("blue", "red"), lwd=2, cex= .6)
```



The power of OSTSC will come out more with larger size or extremely imbalanced datasets.