Over Sampling for Time Series Classification

Matthew F. Dixon, Diego Klabjan and Lan Wei

2017-08-28

Over Sampling for Time Series Classification (OSTSC) is a package for oversampling imbalanced time series classification data.

A significant number of machine learning problems require the accurate classification of rare events or outliers from time series data. For example, the detection of a flash crash in financial market data, price flips in high frequency trading data, and heart arrhythmia from an electrocardiogram. Due to the rarity of these events ('positives'), machine learning classifiers for detecting these events may be biased towards avoiding false positives. Any potential for false positives is greatly exaggerated by the number of negative samples in the data set.

OSTSC oversamples the minority classes using structure preserving oversampling. This approach has been shown to outperform other sampling approaches such as undersampling the majority class, oversampling the minority class, and SMOTE[1].

This version of the package currently only supports univariant, classification of time series. The extension to multi-features requires tensor computations which are not implemented here.

Background

The synthetic balanced samples are generated by a hybridization of the Enhanced Structure Preserving Oversampling (ESPO) and ADASYN algorithms. Unlike conventional sampling approaches which assume that the observations are drawn from an independent, identical distribution, and hence do not preserve the auto-covariance structure, ESPO preserves the covariance structure of the time series.

ESPO is used to generate a large percentage of the synthetic minority samples from univariate labeled time series under the modeling assumption that the predictors are Gaussian. EPSO estimates the covariance structure of the minority-class samples and applies a spectral filer to reduce noise. ADASYN is a nearest neighbor interpolation approach, similarly to SMOTE, which is applied to the EPSO samples.[2]

More formally, given the time series of positive labeled predictors $P = \left\{x_{11}, x_{12}, ..., x_{1|P|}\right\}$ and the negative time series $N = \left\{x_{01}, x_{02}, ..., x_{0|N|}\right\}$, 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 eigenvalues $\left\{\hat{d}_1, ..., \hat{d}_n\right\}$, 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

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

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

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

See [3] for further details of the approach.

Functionality

The package has only one callable function, OSTSC. There's ten parameters users can control, and all of them has default values except the data parameters. For example, the ratio between EPSO generated data and ADASYN generated data is defaulted to be 4:1. But users can always reset this ratio by their own needing.

The package imported R package parallel, doParallel, doSNOW and foreach for parallel control. Parallel is strongly suggested for dataset containing over 30000 observations. The package also imported mvrnorm from R package MASS to generate random vectors from the multivariate normal distribution, and imported rdist from R package fields to calculate the Euclidean distance between vector and matrix.

The vignettes displays three examples. For examining the performances, R packages keras, dummies and pROC are required in running the examples.

Examples

The synthetically generated control datasets

First of all, let's see how OSTSC performs on small datasets. The dataset synthetic_control is included in the OSTSC package, which is generated by the process in Alcock and Manolopoulos (1999) (via). The dataset has already split training and testing data, feature and label data. This dataset is small, but extremely imbalance in training dataset.

```
library(OSTSC)
data(synthetic_control)

train_label <- synthetic_control$train_y
train_sample <- synthetic_control$train_x
test_label <- synthetic_control$test_y
test_sample <- synthetic_control$test_x
```

The time series sequences recorded 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 training data is 1:50, shown below.

```
table(train_label)
```

```
## train_label
##   0   1
## 250   5
```

This is a simple 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 read in the help documents. Users only need to input at least first three variables to be able to call the function.

```
MyData <- OSTSC(train_sample, train_label, k = 4)
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.

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

```
table(over_label)
```

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

Here an Long short-term memory (LSTM) classifier is used to analysis the performance of the OSTSC approach. Using R package keras, to build a LSTM classifier to do time series data classification is effectively and fast.

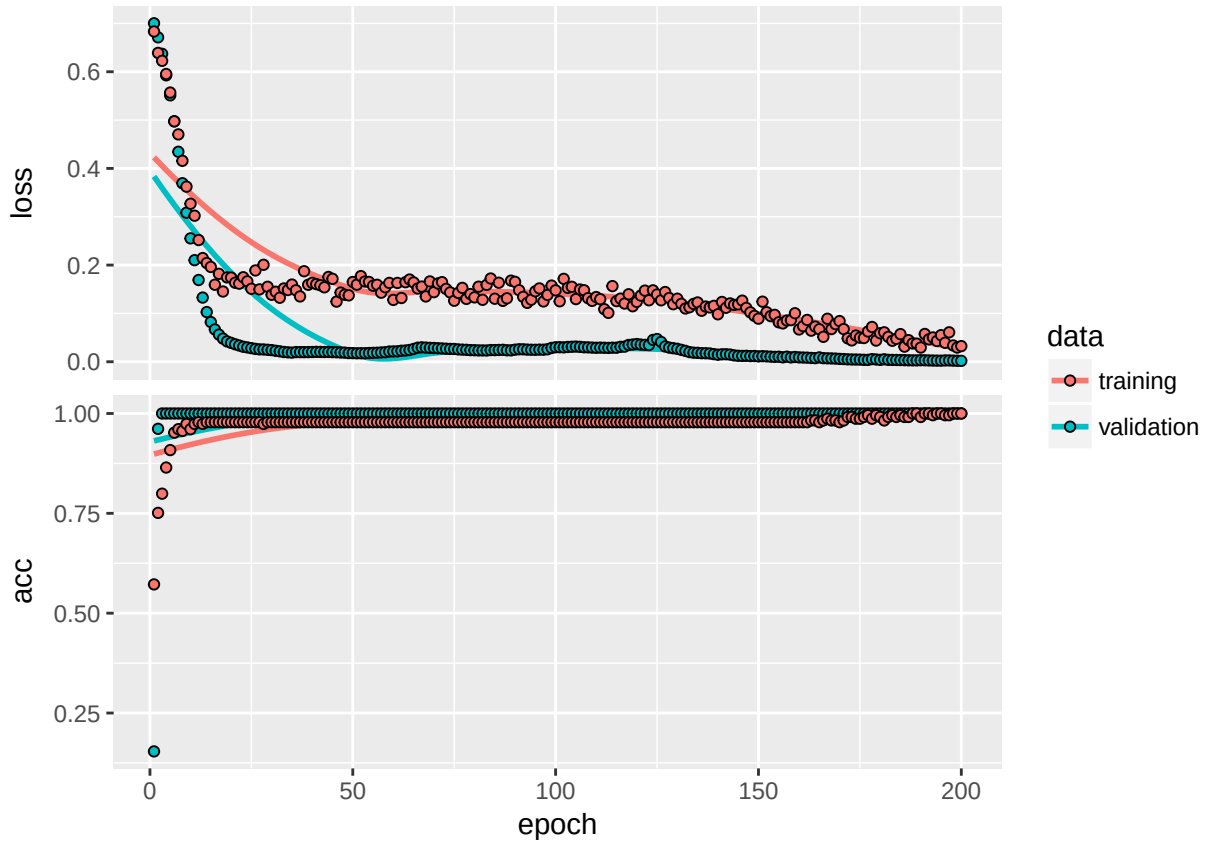
For comparison, first to determine how does the classifier perform on the original data before oversampling.

1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
library(keras)
train_y <- to_categorical(train_label)
test_y <- to_categorical(test_label)
train_x <- array(train_sample, dim = c(dim(train_sample),1))
test_x <- array(test_sample, dim = c(dim(test_sample),1))
```

2. Initialize a sequential model. Add layers to the model. Compile the model. Store the fitting history and show the plot.

```
model = keras_model_sequential()
model %>%
  layer_lstm(10, input_shape = c(dim(train_x)[2], dim(train_x)[3])) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(dim(train_y)[2]) %>%
  layer_dropout(rate = 0.2) %>%
  layer_activation("softmax")
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
lstm_before <- model %>% fit(
  x = train_x,
  y = train_y,
  validation_split = 0.1,
  epochs = 200
)
plot(lstm_before)
```



3. Evaluate the model.

```
score <- model %>% evaluate(test_x, test_y)
```

```
## The loss value is 0.2981745 .
```

```
## The metric value (in this case 'accuracy') is 0.9266667 .
```

Then to determine how does the classifier perform on the new data after oversampling.

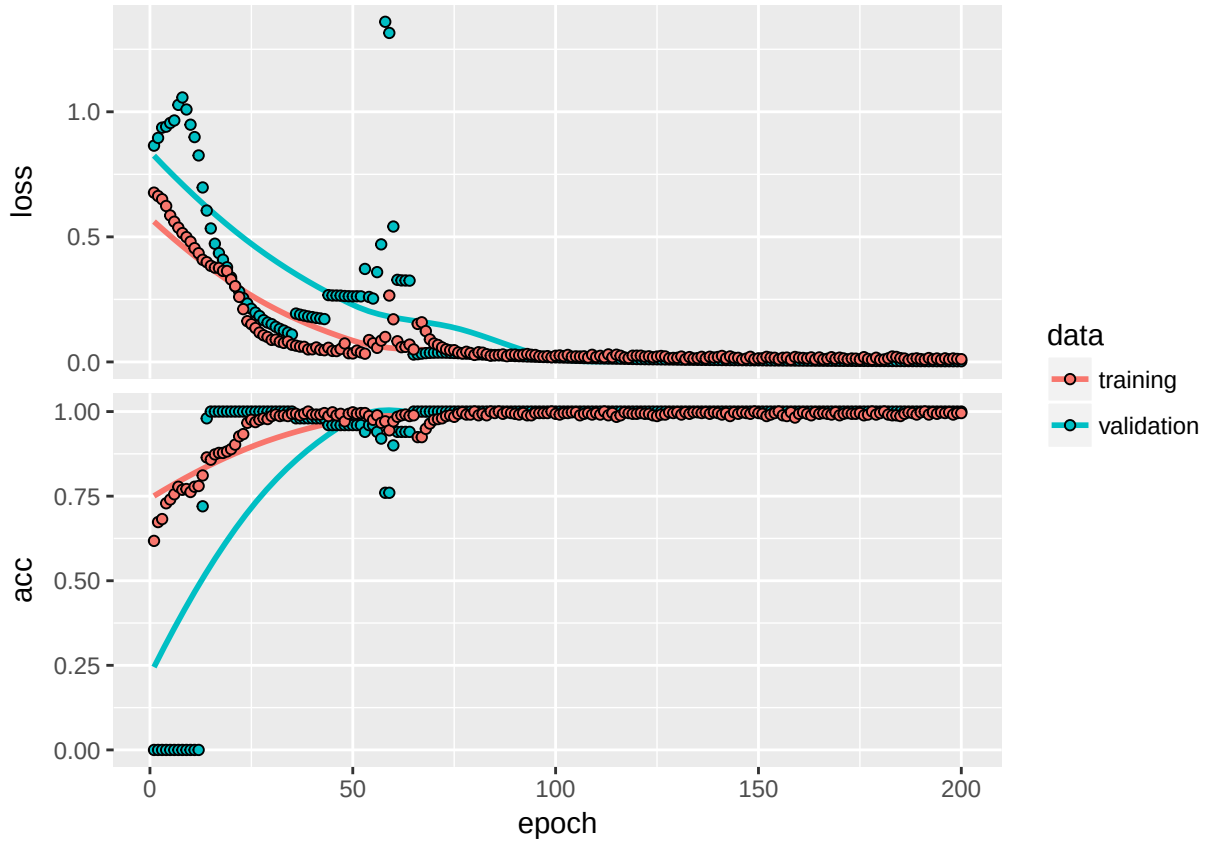
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
over_y <- to_categorical(over_label)
over_x <- array(over_sample, dim = c(dim(over_sample),1))
```

2. Initialize a sequential model. Add layers to the model. Compile the model. Store the fitting history and show the plot.

```
model_over = keras_model_sequential()
model_over %>%
  layer_lstm(10, input_shape = c(dim(over_x)[2], dim(over_x)[3])) %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(dim(over_y)[2]) %>%
  layer_dropout(rate = 0.1) %>%
  layer_activation("softmax")
model_over %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
```

```
)
lstm_after <- model_over %>% fit(
  x = over_x,
  y = over_y,
  validation_split = 0.1,
  epochs = 200
)
plot(lstm_after)
```



3. Evaluate the model.

```
score_over <- model_over %>% evaluate(test_x, test_y)
```

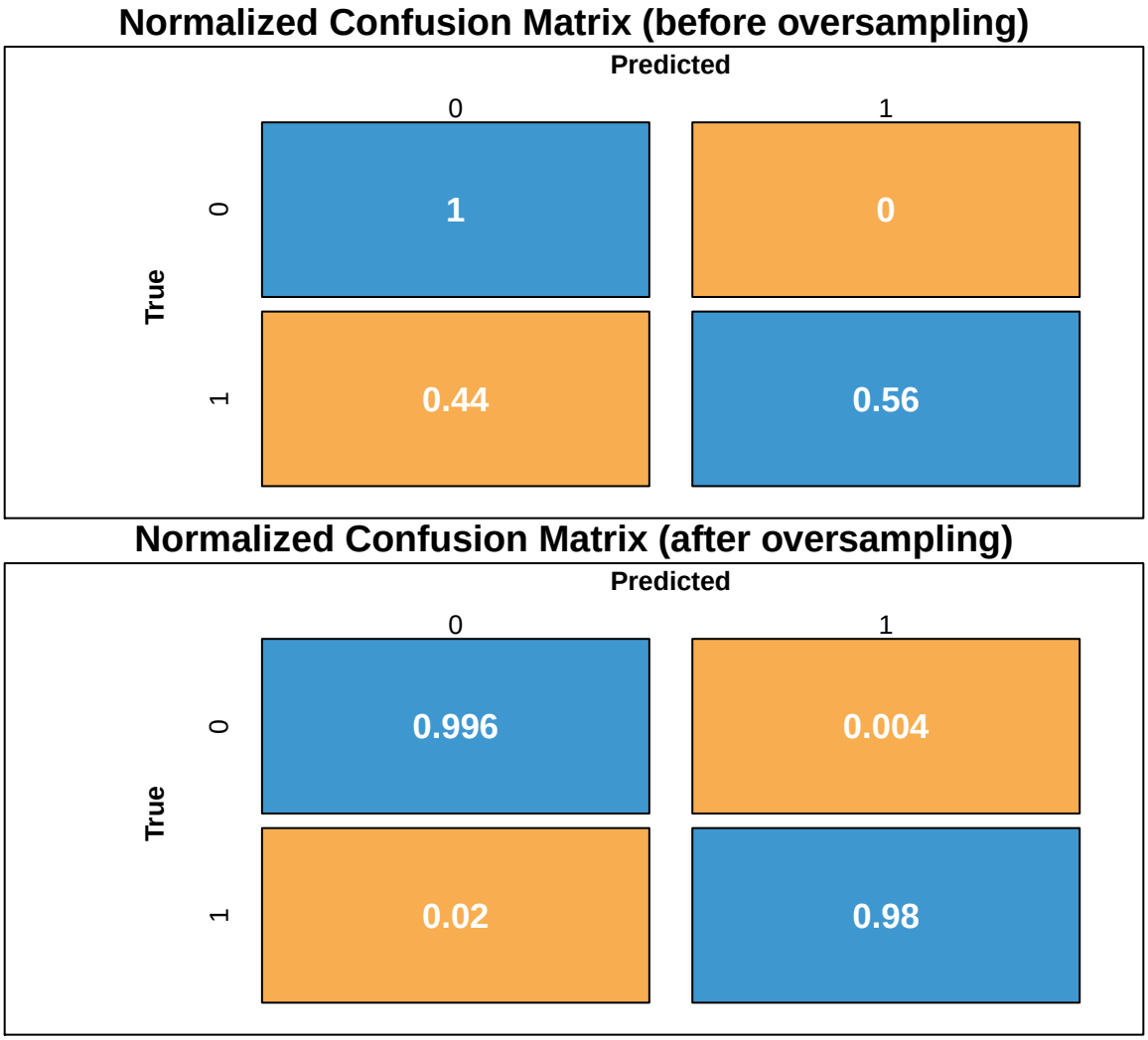
```
## The loss value is 0.0348289 .
```

```
## The metric value (in this case 'accuracy') is 0.9933333 .
```

Besides the loss and accuracy, let's compare the confusion matrices. The mis-classification gets less after oversampling.

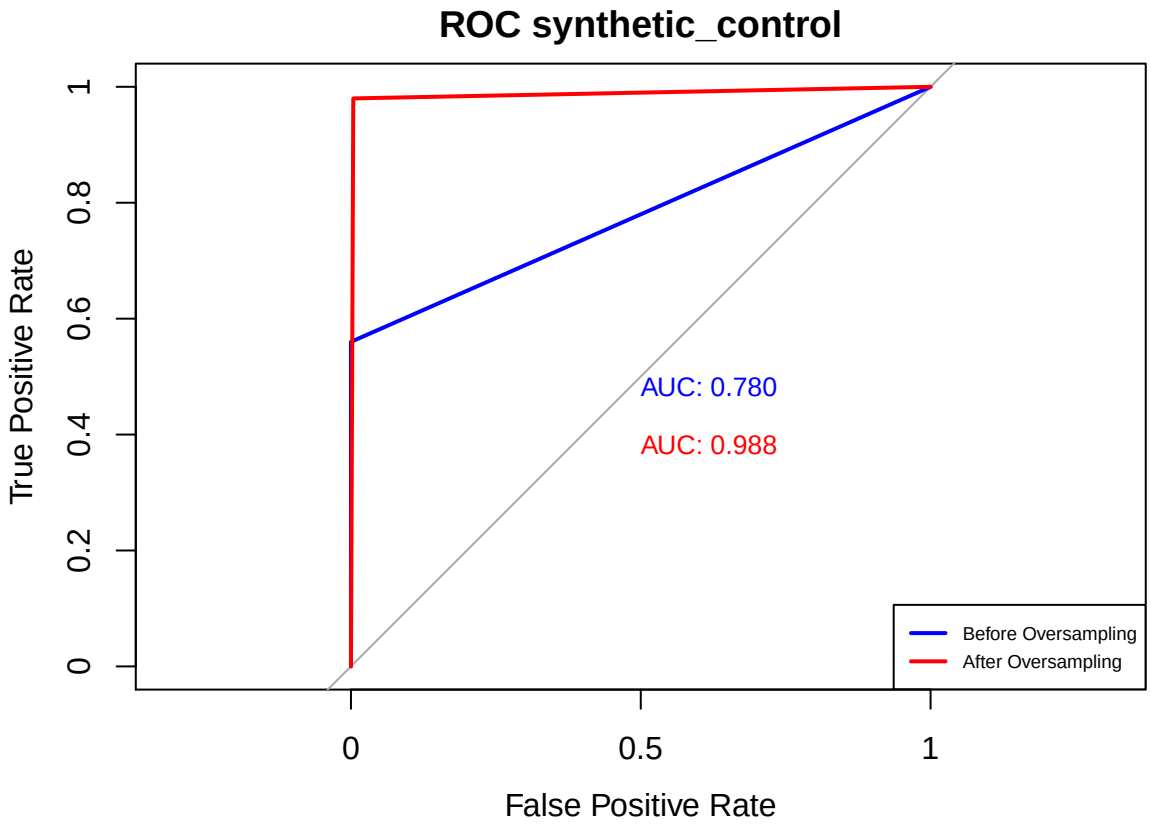
```
pred_label <- model %>% predict_classes(test_x)
pred_label_over <- model_over %>% predict_classes(test_x)
```

```
cm_before <- table(test_label, pred_label)
cm_after <- table(test_label, pred_label_over)
```



The ROC plot tells the same.

```
library(pROC)
plot.roc(test_label, pred_label, legacy.axes = TRUE, col = "blue", print.auc = TRUE,
         print.auc.cex= .8, xlab = 'False Positive Rate', ylab = 'True Positive Rate',
         main="ROC synthetic_control")
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)
```



This is a simple example on a small size dataset. Let's move to a more complex dataset MHEALTH.

The MHEALTH dataset

The MHEALTH dataset is devised to benchmark techniques dealing with human behavior analysis based on multimodal body sensing. (via).[4] For example convenience, only subject 1-5 and feature 12 (magnetometer from the left-ankle sensor (X axis)) are used, and the dataset is reformated to binary class. Class 11 (Running) is set as positive, others as negative. The time series sequences length uses 30. Each sequence occurs in one line. The dataset has already split to training and testing data, feature and label data. It stores on github, so its loading method is different from synthetic control data.

```
MHEALTH <- MHEALTH()

train_label <- MHEALTH$train_y
train_sample <- MHEALTH$train_x
test_label <- MHEALTH$test_y
test_sample <- MHEALTH$test_x
```

Class 1 stands for positive data, while class 0 stands for negative. The imbalance of the train dataset is shown below.

```
table(train_label)
```

```
## train_label
##    0    1
```

[4]Banos, O., Garcia, R., Holgado, J. A., Damas, M., Pomares, H., Rojas, I., Saez, A., Villalonga, C. mHealthDroid: a novel framework for agile development of mobile health applications. Proceedings of the 6th International Work-conference on Ambient Assisted Living an Active Ageing (IWAAL 2014), Belfast, Northern Ireland, December 2-5, (2014)

```
## 10584   255
```

After Oversampling by OSTSC, the positive data and negative data are balanced.

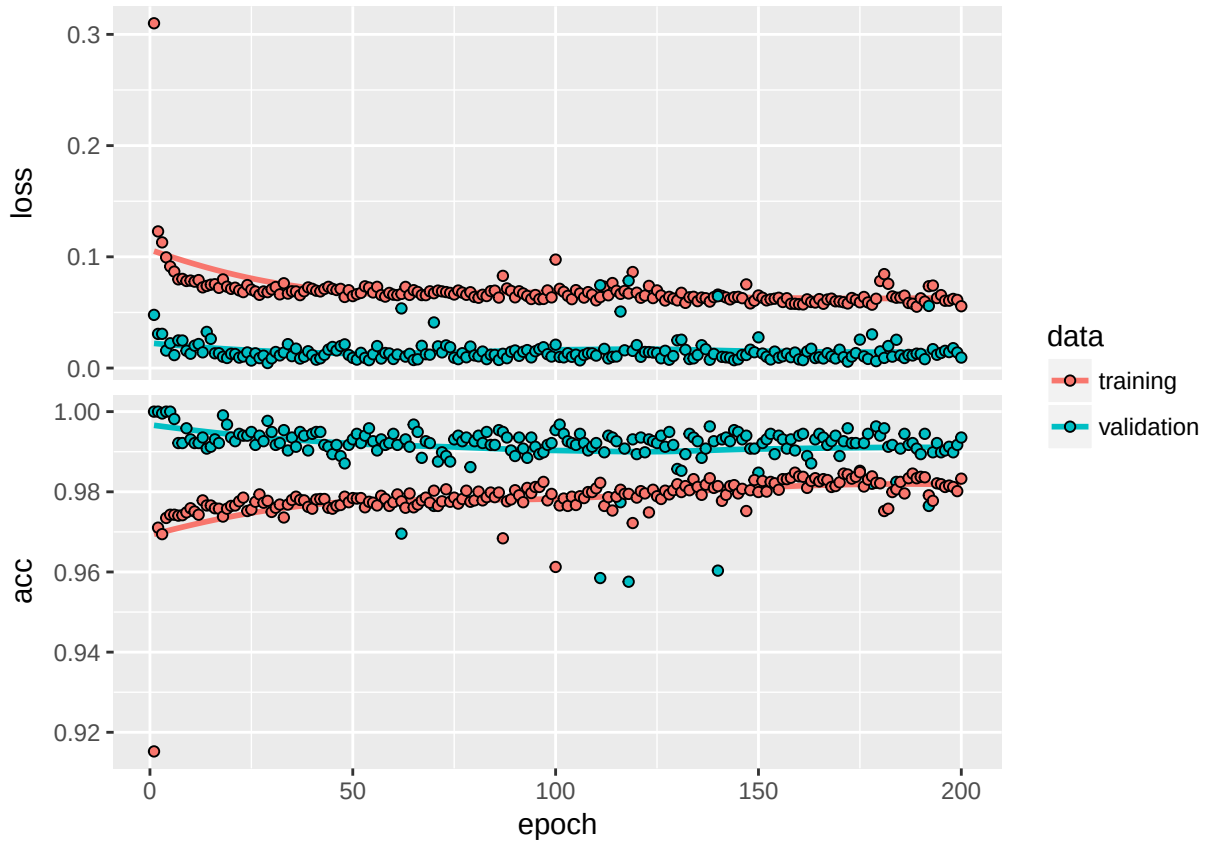
```
MyData <- OSTSC(train_sample, train_label)
over_sample <- MyData$sample
over_label <- MyData$label

table(over_label)
```

For comparison, we use the same Long short-term memory (LSTM) classifier.

1. To train the classifier on the original data before oversampling.

```
library(keras)
train_y <- to_categorical(train_label)
test_y <- to_categorical(test_label)
train_x <- array(train_sample, dim = c(dim(train_sample),1))
test_x <- array(test_sample, dim = c(dim(test_sample),1))
```

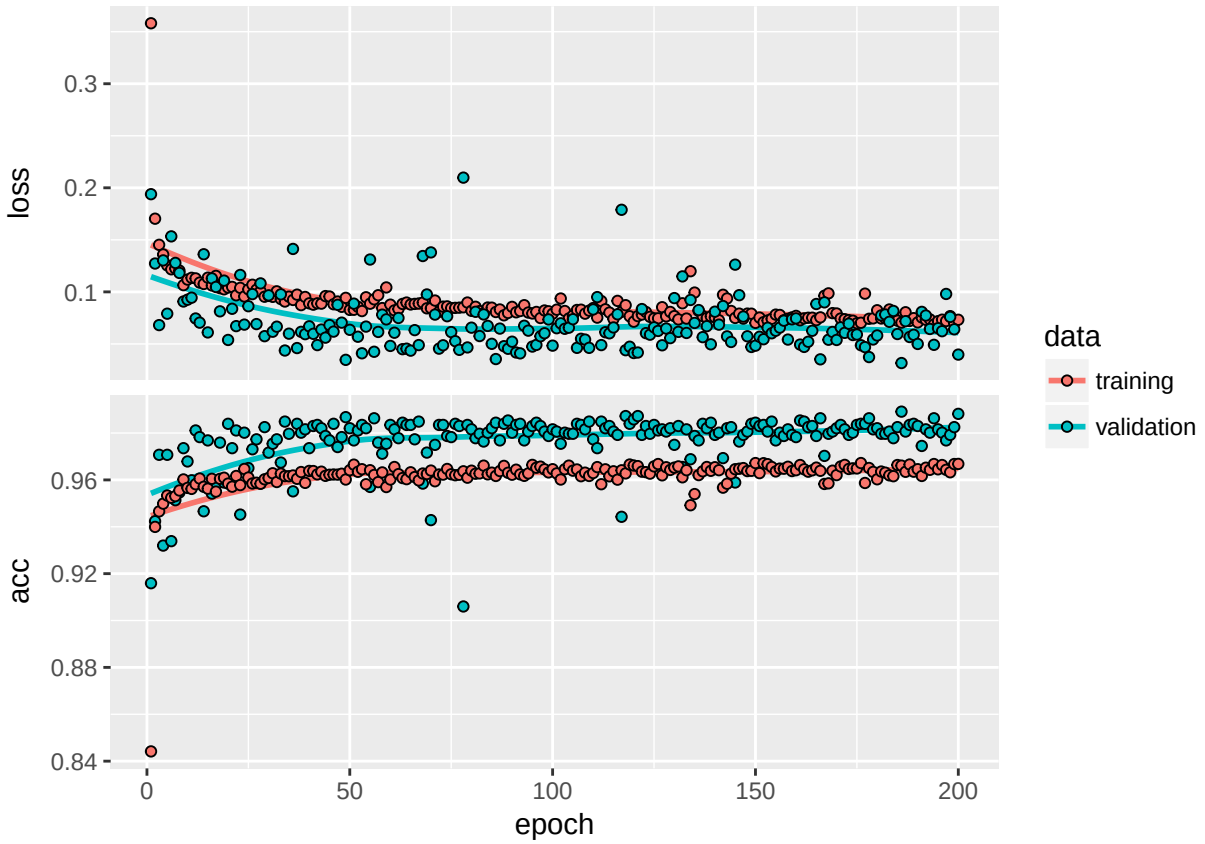
```
model = keras_model_sequential()
model %>%
  layer_lstm(10, input_shape = c(dim(train_x)[2], dim(train_x)[3])) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(dim(train_y)[2]) %>%
  layer_dropout(rate = 0.2) %>%
  layer_activation("softmax")
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
lstm_before <- model %>% fit(
  x = train_x,
  y = train_y,
  validation_split = 0.2,
  epochs = 200
)
plot(lstm_before)
```



2. To train the classifier on the new data after oversampling.

```
over_y <- to_categorical(over_label)
over_x <- array(over_sample, dim = c(dim(over_sample),1))
```

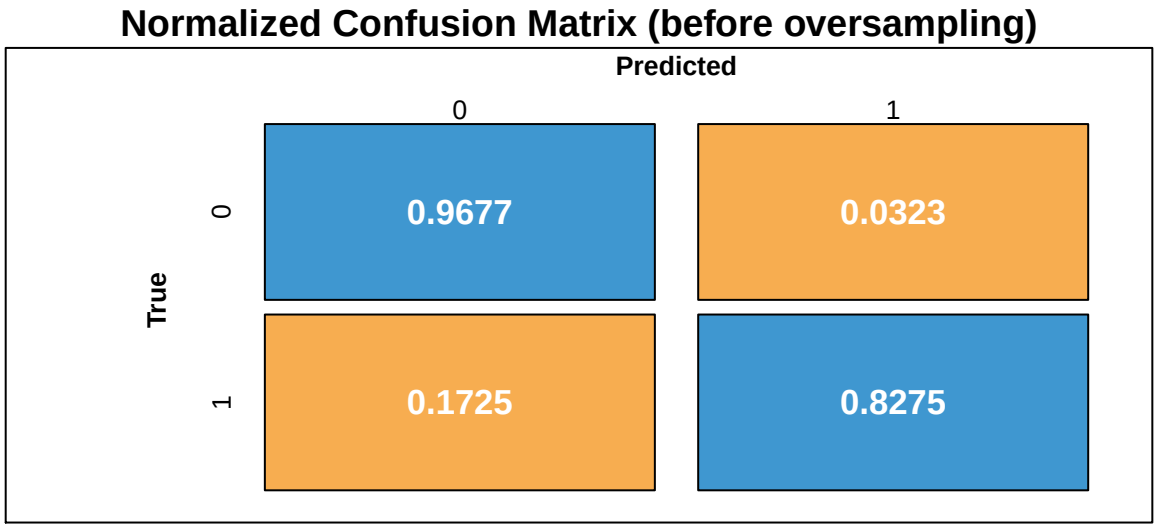
```
model_over = keras_model_sequential()
model_over %>%
  layer_lstm(10, input_shape = c(dim(over_x)[2], dim(over_x)[3])) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(dim(over_y)[2]) %>%
  layer_dropout(rate = 0.2) %>%
  layer_activation("softmax")
model_over %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
lstm_after <- model_over %>% fit(
  x = over_x,
  y = over_y,
  validation_split = 0.1,
  epochs = 200
)
plot(lstm_after)
```

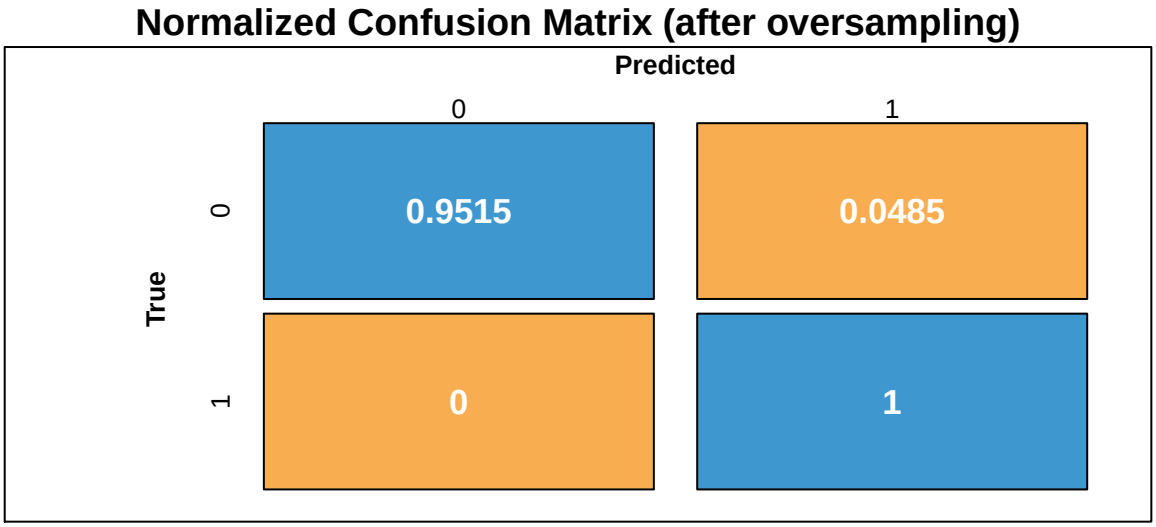


3. To compare the confusion matrices and ROC plot. The power of OSTSC will come out more on more training epochs.

```
pred_label <- model %>% predict_classes(test_x)
pred_label_over <- model_over %>% predict_classes(test_x)
```

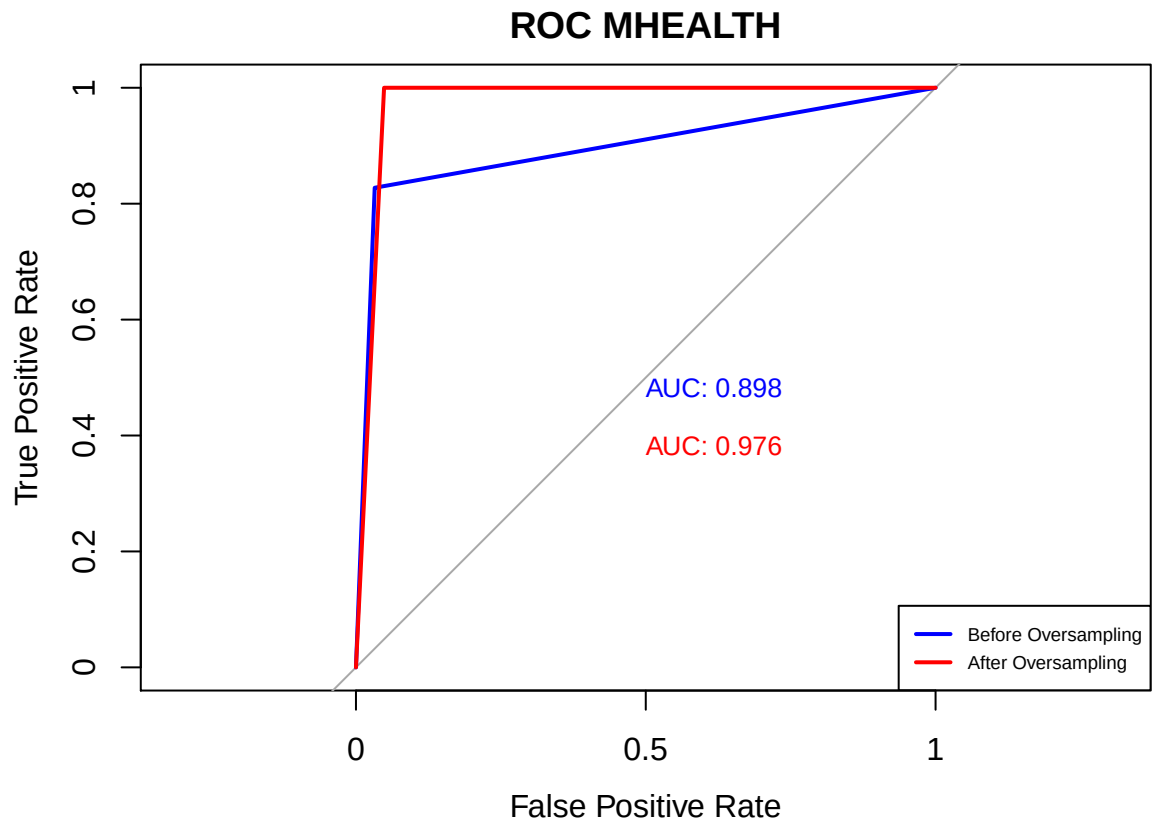
```
cm_before <- table(test_label, pred_label)
cm_after <- table(test_label, pred_label_over)
```





The ROC plot shows the classification performances more intuitively.

```
library(pROC)
plot.roc(as.vector(test_label), pred_label, legacy.axes = TRUE, col = "blue",
         print.auc = TRUE, print.auc.cex= .8, xlab = 'False Positive Rate',
         ylab = 'True Positive Rate', main="ROC MHEALTH")
plot.roc(as.vector(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)
```



Let's see how the OSTSC performs on a high frequency trading dataset.

The high frequency trading dataset

The feature is from instantaneous liquidity imbalance using the best bid to ask ratio, up-tick as class 1, down-tick as class -1, and normal status as class 0. The time series sequences length is set to 10. For example convenience, the data using here is random selected from a real and giant size high frequency trading dataset. While the whole observations are ordered in the time order, the dataset haven't split training and setting data. The users can split it by any ratio they like. Here we split it by half and half. The first half observations are training data, while the rest are testing. Its loading method is same with MHEALTH data for their similar sizes.

```
HFT <- HFT ()

label <- HFT$y
sample <- HFT$x
train_label <- label[1:15000]
train_sample <- sample[1:15000, ]
test_label <- label[15001:30000]
test_sample <- sample[15001:30000, ]
```

The imbalance of the train dataset is shown below.

```
table(train_label)
```

```
## train_label
##    -1     0     1
##   297 14424   279
```

The OSTSC function deals with multi-class classification, so the oversampling could apply on one or two classes. If running on only one class, the class with least observations would be chosen. The oversampling is formed using one-vs-rest manner.

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

The imbalance of the oversampled dataset is shown below.

```
table(over_label)
```

```
## over_label
##    -1     0     1
## 14703 14424 14721
```

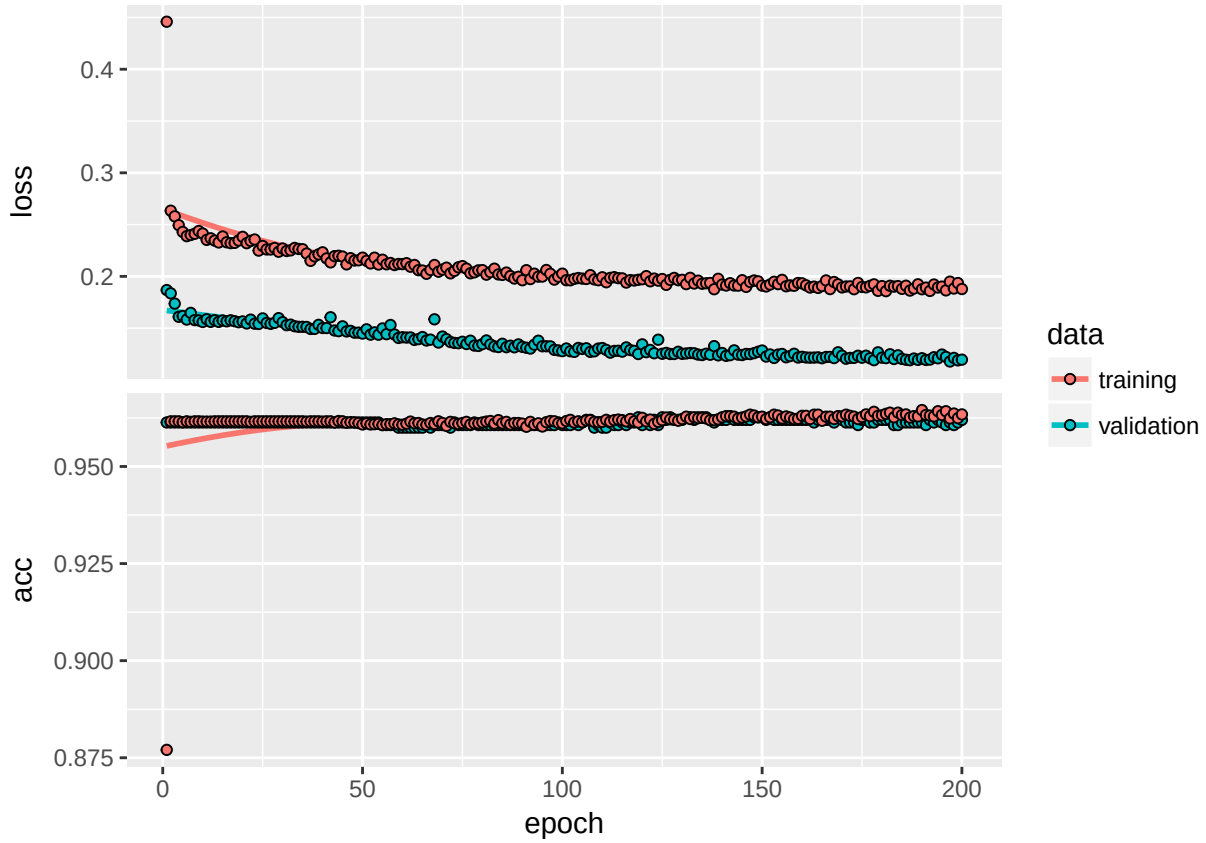
As same as elder examples, we use the same Long short-term memory (LSTM) classifier.

1. To train the classifier on the original data before oversampling.

```
library(keras)
library(dummies)
train_y <- dummy(train_label)
test_y <- dummy(test_label)
train_x <- array(train_sample, dim = c(dim(train_sample),1))
test_x <- array(test_sample, dim = c(dim(test_sample),1))
```

```
model = keras_model_sequential()
model %>%
```

```
  layer_lstm(10, input_shape = c(dim(train_x)[2], dim(train_x)[3])) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(dim(train_y)[2]) %>%
  layer_dropout(rate = 0.2) %>%
  layer_activation("softmax")
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
lstm_before <- model %>% fit(
  x = train_x,
  y = train_y,
  validation_split = 0.1,
  epochs = 200
)
plot(lstm_before)
```

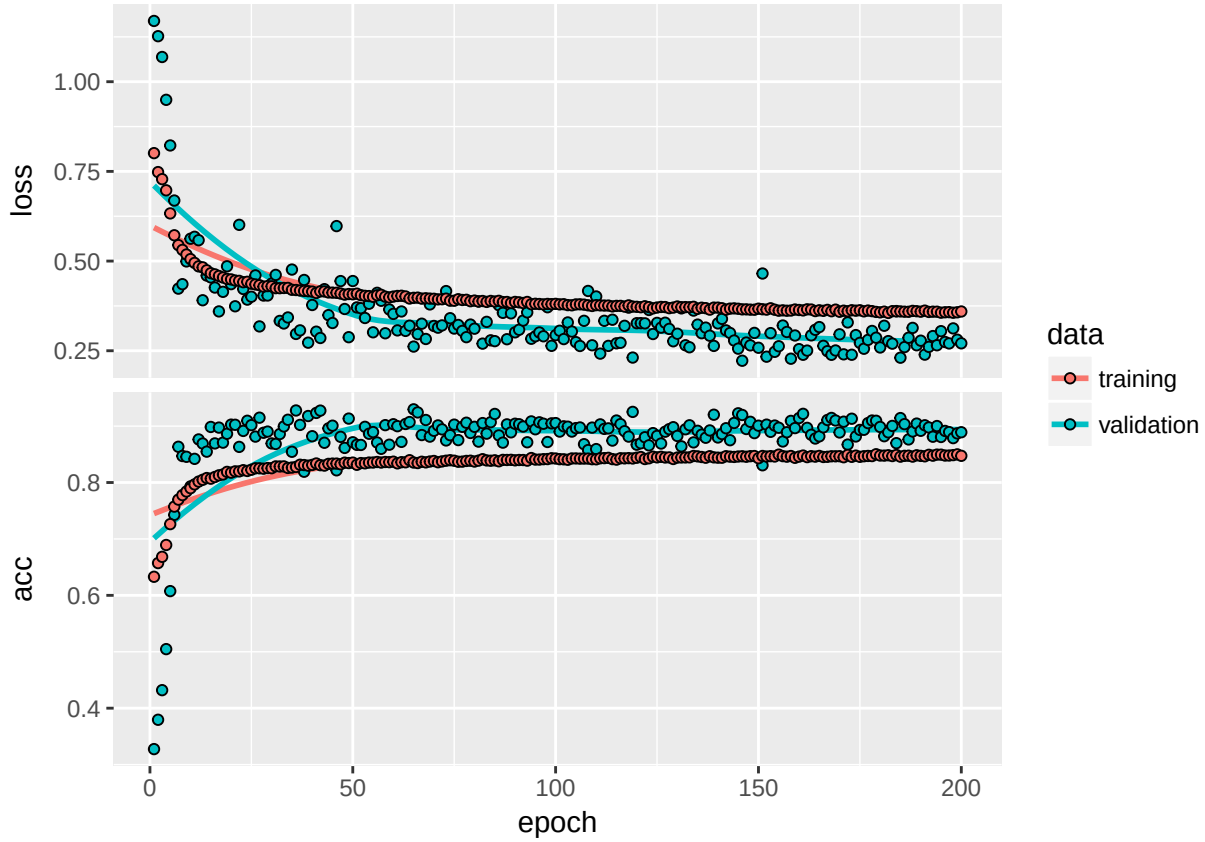


2. To train the classifier on the new data after oversampling.

```
over_y <- dummy(over_label)
over_x <- array(over_sample, dim = c(dim(over_sample),1))
```

```
model_over = keras_model_sequential()
model_over %>%
  layer_lstm(10, input_shape = c(dim(over_x)[2], dim(over_x)[3])) %>%
  layer_dropout(rate = 0.2) %>%
```

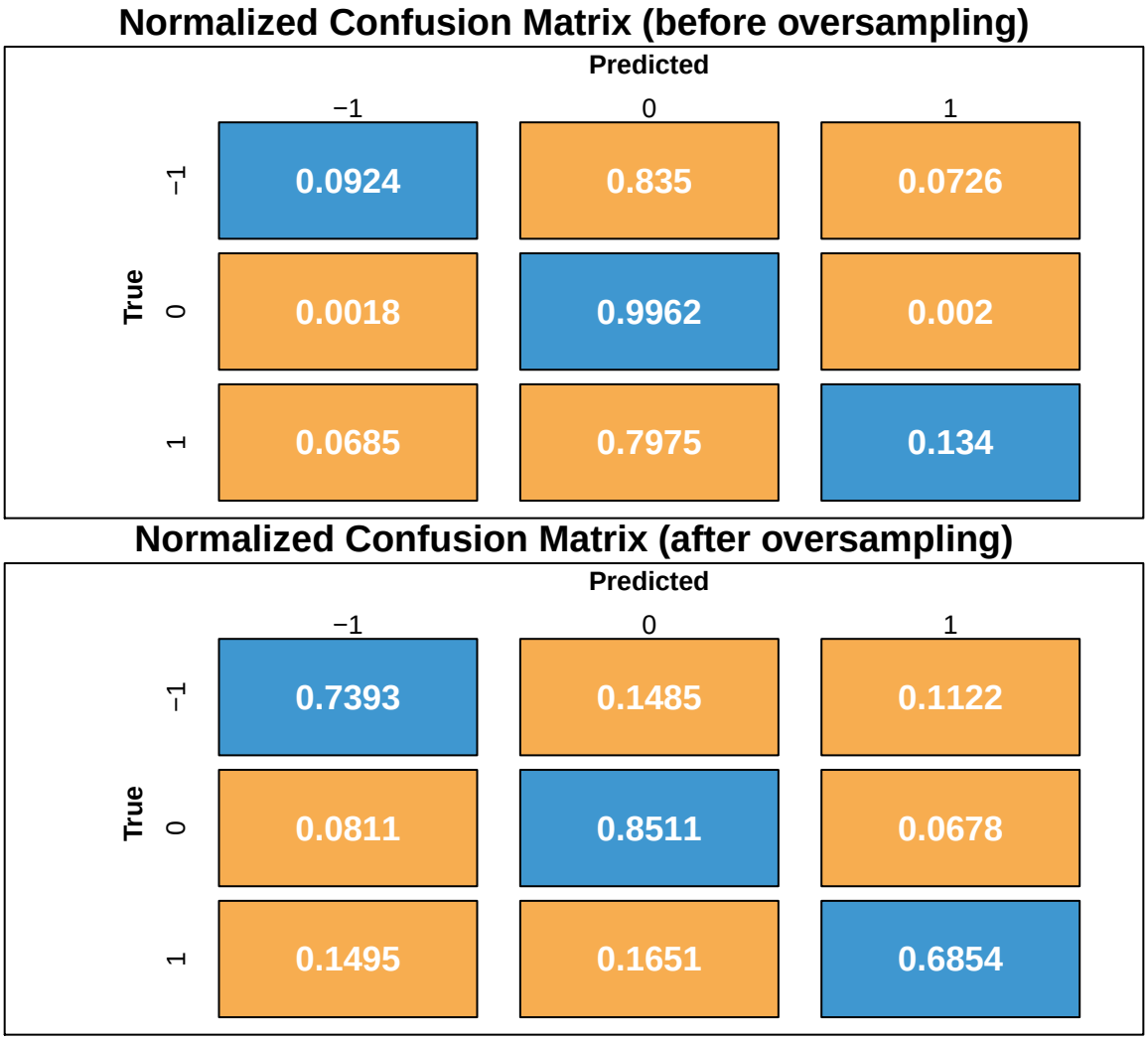
```
  layer_dense(dim(over_y)[2]) %>%
  layer_dropout(rate = 0.2) %>%
  layer_activation("softmax")
model_over %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
lstm_after <- model_over %>% fit(
  x = over_x,
  y = over_y,
  validation_split = 0.1,
  epochs = 200
)
plot(lstm_after)
```



3. To compare the confusion matrices and ROC plot. When the dataset size gets larger and the imbalance degree gets more severe, the OSTSC performs better than unoversampled data.

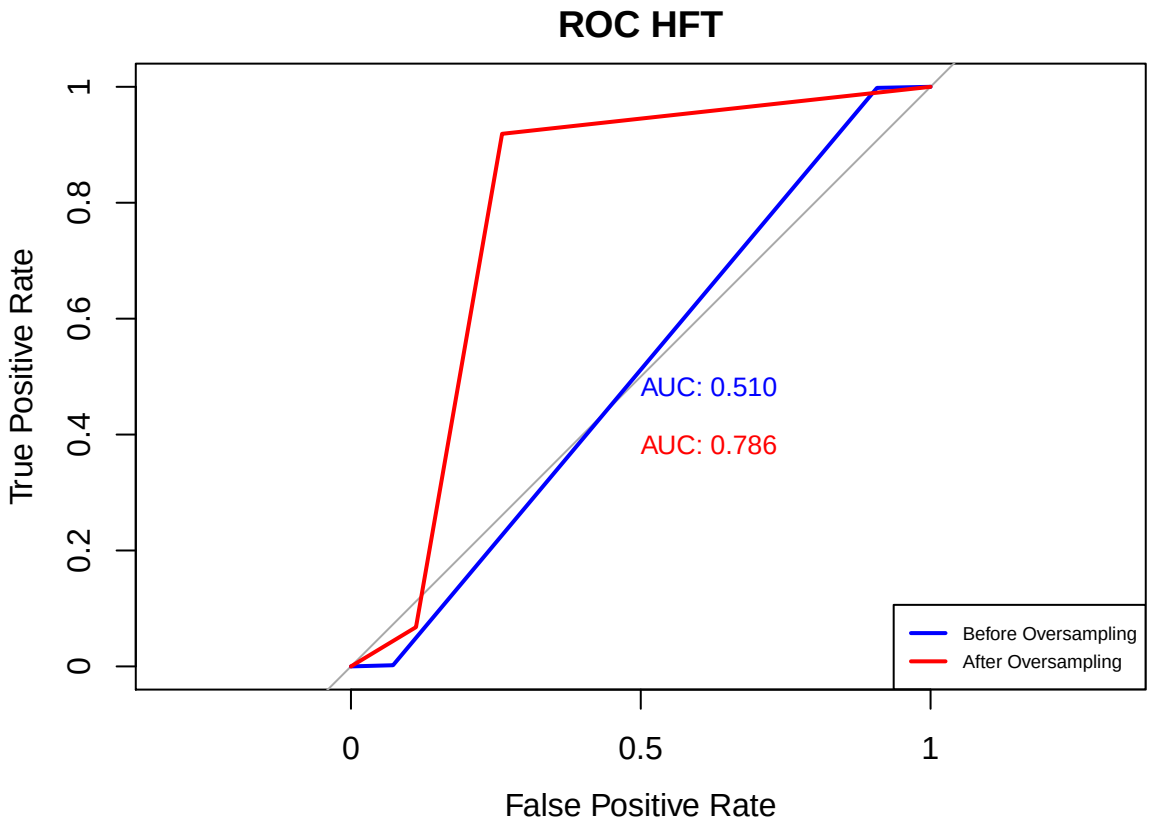
```
pred_label <- model %>% predict_classes(test_x)
pred_label_over <- model_over %>% predict_classes(test_x)
```

```
cm_before <- table(test_label, pred_label)
cm_after <- table(test_label, pred_label_over)
```



The ROC plot shows the classification performances more intuitively. Because the dataset has three classes, the AUC value calculates by average.

```
library(pROC)
plot.roc(test_label, pred_label, legacy.axes = TRUE, col = "blue", print.auc = TRUE,
         print.auc.cex= .8, xlab = 'False Positive Rate', ylab = 'True Positive Rate',
         main="ROC HFT")
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)
```



Above are three examples on three different datasets. OSTSC package could have a wide usage over multi-regions.

References

[1]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.

[2]H. Cao, V. Y. F. Tan and J. Z. F. Pang. A Parsimonious Mixture of Gaussian Trees Model for Oversampling in Imbalanced and Multi-Modal Time-Series Classification. IEEE Trans. on Neural Network and Learning System (TNNLS), vol. 25(12), pp. 2226-2239, 2014.

[3]H. Cao, X. L. Li, Y. K. Woon and S. K. Ng. SPO: Structure Preserving Oversampling for Imbalanced Time Series Classification. Proc. IEEE Int. Conf. on Data Mining ICDM, pp. 1008-1013, 2011.

[4]O. Banos, R. Garcia, J. A. Holgado, M. Damas, H. Pomares, I. Rojas, A. Saez, and C. Villalonga. mHealthDroid: A Novel Framework for Agile Development of Mobile Health Applications. Proceedings of the 6th International Work-conference on Ambient Assisted Living an Active Ageing (IWAAL 2014), Belfast, Northern Ireland, December 2-5, 2014.

[5]O. Banos, C. Villalonga, R. Garcia, A. Saez, DM. Damas, J. A. Holgado, S. Lee, H. Pomares, and I. Rojas. Design, Implementation and Validation of A Novel Open Framework for Agile Development of Mobile Health Applications. BioMedical Engineering OnLine, vol. 14, no. S2:S6, pp. 1-20, 2015.

[6]M. F. Dixon. Sequence Classification of the Limit Order Book using Recurrent Neural Networks, to appear in J. Computational Science, arXiv:1707.05642, 2017.