Over Sampling for Time Series Classification

Matthew F. Dixon, Diego Klabjan and Lan Wei

2017-10-02

Introduction

A significant number of learning problems involve the accurate classification of rare events or outliers from time series data. For example, the detection of a flash crash or rogue trading from financial markets data, or heart arrhythmia from an electrocardiogram. Due to the rarity of these events, machine learning classifiers for detecting these events may be biased towards avoiding false positives because any potential for false positives is greatly exaggerated by the number of negative samples in the data set.

Class imbalance problems are most easily addressed by either oversampling the minority class or undersampling the majority class, or both. More [7] compared a batch of resampling techniques' classification performances on imbalance datasets. Besides the conventional resampling approaches, More showed how ensemble methods keep as much original information as possible from the majority class when performing undersampling. Ensemble methods perform well and have gained popularity in the data mining literature. Dubey et al. [9] studied an ensemble system of feature selection and data sampling from imbalance Alzheimer's Disease Neuroimaging Initiative dataset.

However the imbalanced time series classification problem is more complex when the time dimension is accounted for. Not only is the assumption that the observations are independent too strong, but also the predictors may be cross-correlated too. The sample correlation structures may weaken or even broken by the conventional resampling approaches described above.

There are two existing research directions for imbalance time series classification. One is to preserve the covariance structure during oversampling proposed by Hong et al. [1]. Another one is to conduct undersampling with various learning algorithms, proposed by Liang and Zhang [8].

Both of the two approaches are limited to binary-class classification and do not consider the more general problem of multi-classification

A key observation by Hong et al. [1] is that the sampling in a time series should preserve the covariance structure. This approach has been shown to outperform other sampling approaches such as undersampling the majority class, oversampling the minority class, and SMOTE. Our R package `Over Sampling for Time Series Classification` (OSTSC) is built on this idea. OSTSC oversamples the minority classes using structure preserving oversampling and ADASYN.

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 [1].

More formally, given the time series of positive labeled predictors $P = \{x_{11}, x_{12}, ..., x_{1|P|}\}$ and the negative time series $N = \{x_{01}, x_{02}, ..., x_{0|N|}\}$, 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 $\sum_{i=1}^{|P_t|} \Gamma_i = 1$.

See [1] 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

Data loading & oversampling

The OSTSC package has three small build-in datasets.

The synthetically generated control datasets

The dataset `Dataset_Synthetic_Control` is generated by the process in Alcock and Manolopoulos (1999) (via). 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. Users load the dataset into environment by calling `data()`.

```
library(OSTSC)
data(Dataset_Synthetic_Control)

train.label <- Dataset_Synthetic_Control$train.y
train.sample <- Dataset_Synthetic_Control$train.x
```

```
test.label <- Dataset_Synthetic_Control$test.y
test.sample <- Dataset_Synthetic_Control$test.x
```

The train dataset has sequence length 60 and observations number 300. Each row is a sequence of observation.

```
dim(train.sample)
```

```
## [1] 300  60
```

The imbalance of training data is 1:5.

```
table(train.label)
```

```
## train.label
##   0   1
## 250  50
```

Here is a simple example to show how to oversample the minority data to the same amount of majority, 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 label and sample data to be able to call the function. The OSTSC function receives label data and sample data seperately.

```
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
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
```

The minority class data is oversampled to the same amount of the majority class. The minority-majority formation uses a one-vs-rest manner. For this dataset, the class 1 data has been oversampled to the same amount of class 0.

```
dim(over.sample)
```

```
## [1] 500  60
```

The automatic diatoms identification datasets

The dataset `Dataset_Adiac` is generated from a pilot study concerning automatic identification of diatoms (unicellular algae) on the basis of images (2004) (via). The dataset originally had 37 classes. But we selected only one class as positive class (class 1) and all others as negative class (class 0) to form an extremely imbalance dataset.

```
data(Dataset_Adiac)

train.label <- Dataset_Adiac$train.y
train.sample <- Dataset_Adiac$train.x
test.label <- Dataset_Adiac$test.y
test.sample <- Dataset_Adiac$test.x
```

The training dataset has sequence length 176 and observations number 390.

```
dim(train.sample)
```

```
## [1] 390 176
```

The imbalance of training data is 1:29.

```r
table(train.label)
```

```
## train.label
##   0   1
## 377  13
```

The OSTSC also performs well on this extremely imbalance dataset.

```r
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label
```

Let's check the balanced new dataset.

```r
table(over.label)
```

```
## over.label
##   0   1
## 377 377
```

The high frequency trading dataset

The OSTSC function deals with multi-class classification. The users could demand the number of the classes to be oversampled, which defaulted to be as most as possible. The oversampling would start from the class with least observations. The dataset `Dataset_HFT` is extracted from a real and giant size 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.
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.

```r
data(Dataset_HFT)

train.label <- Dataset_HFT$y
train.sample <- Dataset_HFT$x
```

The time series sequences length is set to 10. For example convenience, the data random selected 300 observations.

```r
dim(train.sample)
```

```
## [1] 300  10
```

The imbalance of dataset is 1:48:1.

```r
table(train.label)
```

```
## train.label
##  -1   0   1
##   6 288   6
```

Here we oversamples all the minority class. The oversampling is processed on a one-to-rest method, which means the minority class would be oversampled to the same number of the sum of all other classes.

```r
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label
```

Let's check the balanced new dataset.

```
table(over.label)
```

```
## over.label
##  -1   0   1
## 294 288 294
```

Above is how the OSTSC does oversampling. In the next section, we would check the oversampled data on two larger built-in datasets.

Checking OSTSC on built-in datasets

The MHEALTH dataset

The dataset `Dataset_MHEALTH` is devised to benchmark techniques dealing with human behavior analysis based on multimodal body sensing. (via) [4]. For example convenience, only subject 1 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 dataset has already split to training and testing data, feature and label data.

```
data(Dataset_MHEALTH_Check)

train.label <- Dataset_MHEALTH_Check$train.y
train.sample <- Dataset_MHEALTH_Check$train.x
test.label <- Dataset_MHEALTH_Check$test.y
test.sample <- Dataset_MHEALTH_Check$test.x
```

The time series sequences length uses 30. Each sequence occurs in one line.

```
dim(train.sample)
```

```
## [1] 2687   30
```

Class 1 stands for positive data, while class 0 stands for negative. The imbalance of the train dataset is 1:52.

```
table(train.label)
```

```
## train.label
##    0    1
## 2636   51
```

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

```
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label

table(over.label)
```

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.

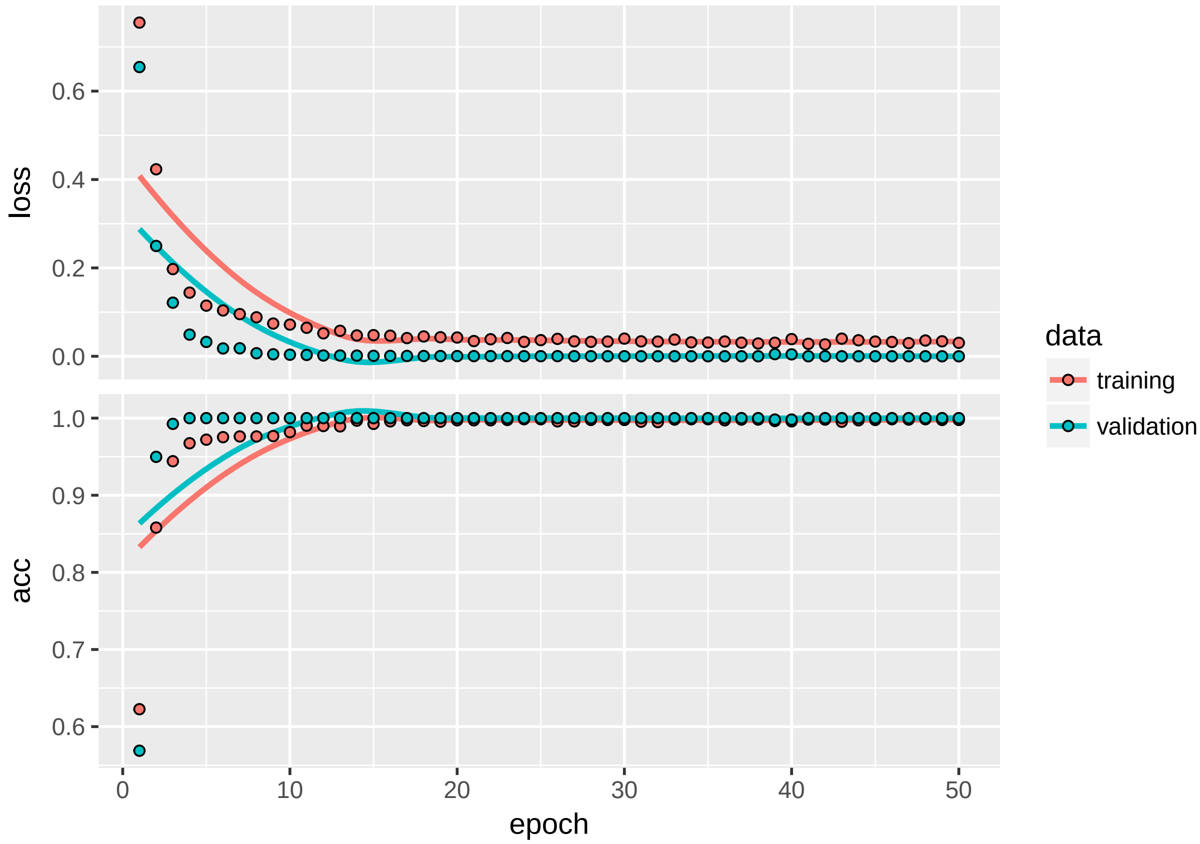
For comparison, first to determine how does the classifier perform on the original data before oversampling.

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.

```r
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.

```r
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 = 50
)
cat("\n Plot the accuracy and loss changes during the LSTM: \n\n")
plot(lstm.before)
```



3. Evaluate the model.

```
score <- model %>% evaluate(test.x, test.y)
```

```
## The loss value is 0.1528046 .
```

```
## The metric value (in this case 'accuracy') is 0.9549851 .
```

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

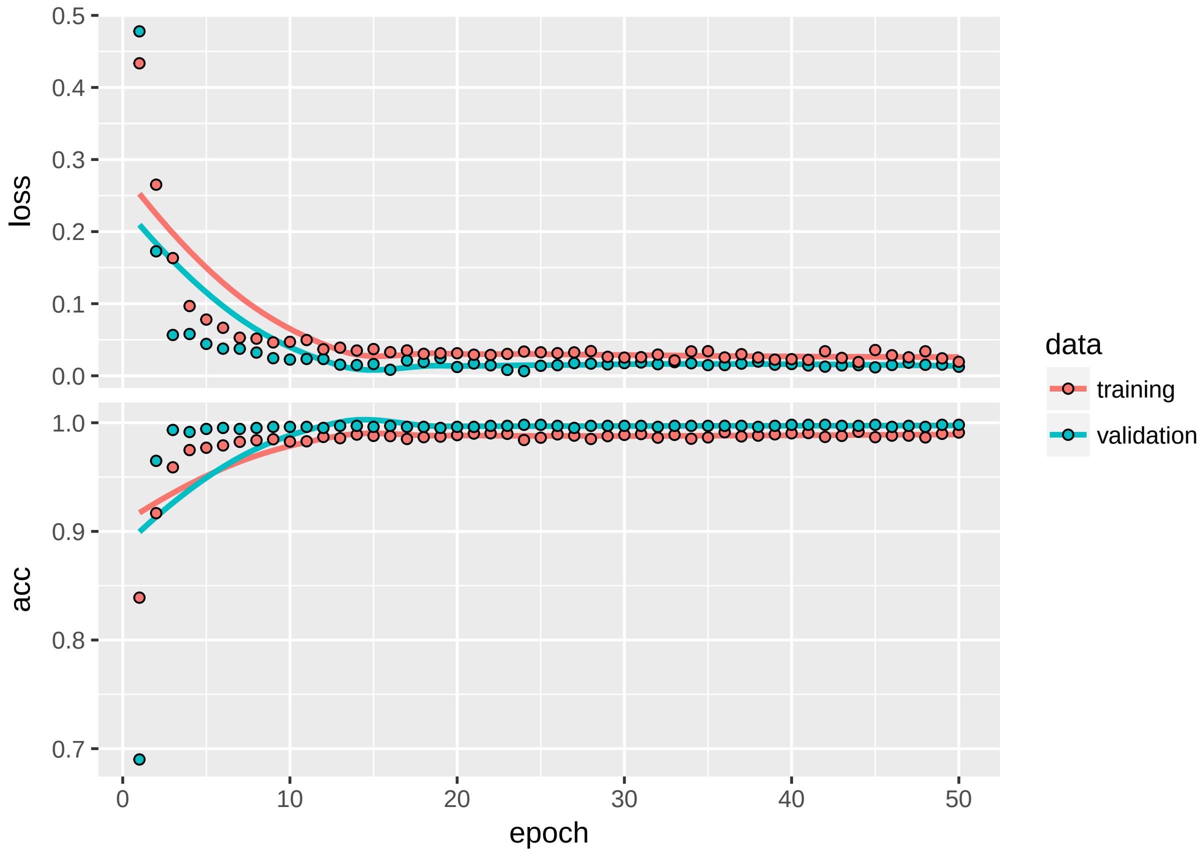
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
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 = 0.2,
  epochs = 50
)
cat("\n Plot the accuracy and loss changes during the LSTM: \n\n")
plot(lstm.after)
```



3. Evaluate the model.

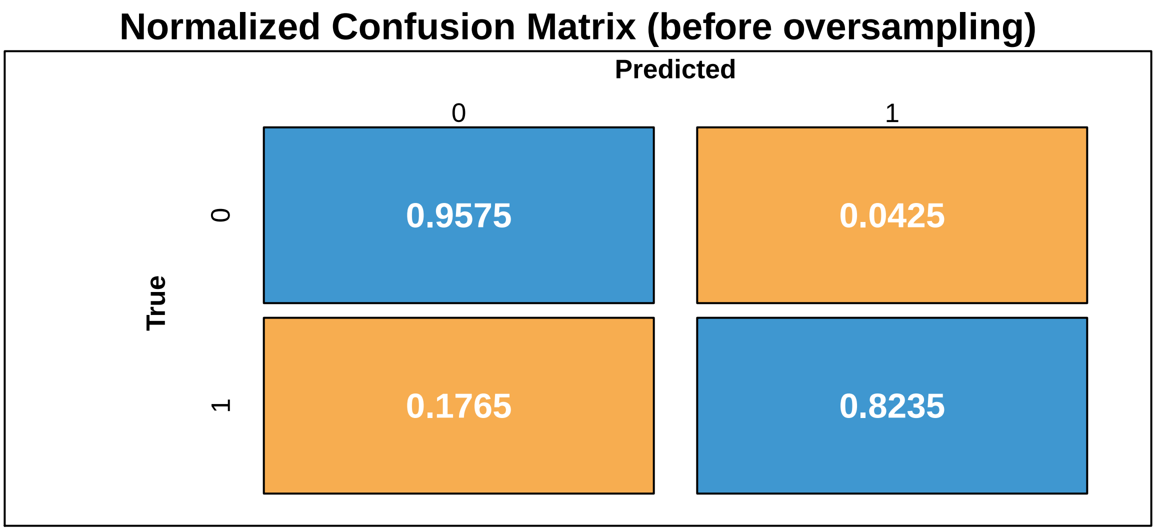
```
score.over <- model.over %>% evaluate(test.x, test.y)
```

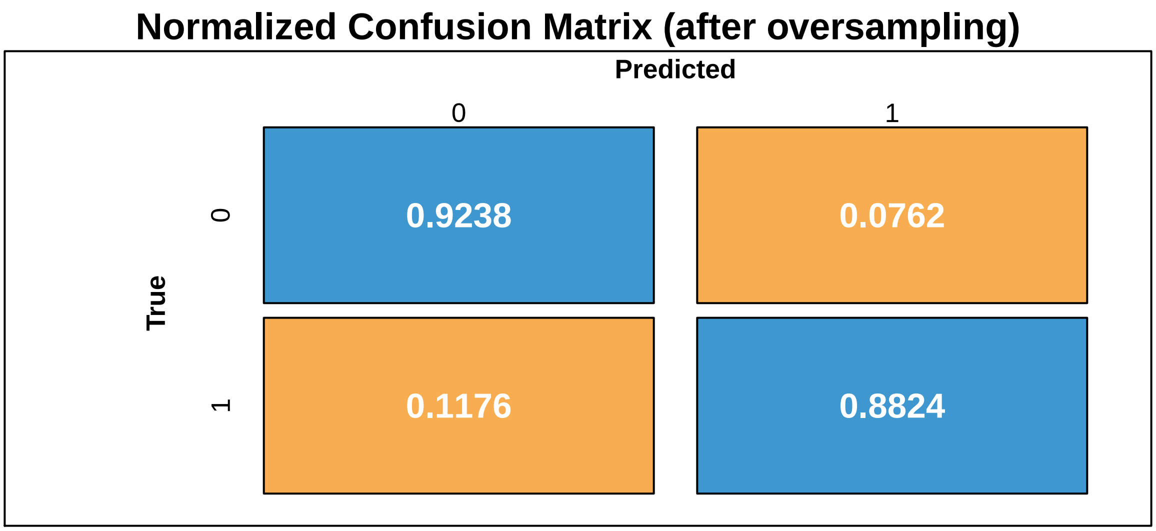
```
## The loss value is 0.4943692 .
```

```
## The metric value (in this case 'accuracy') is 0.9229911 .
```

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

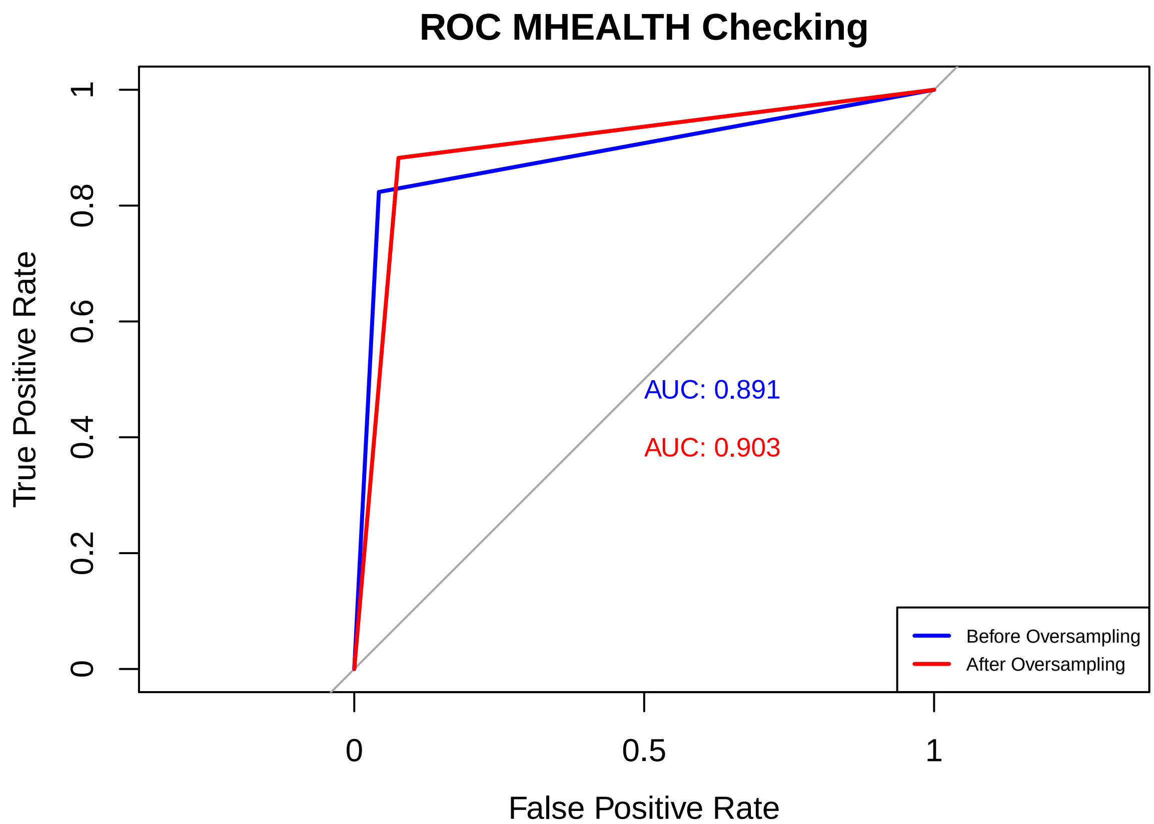
```
pred.label <- model %>% predict_classes(test.x)
pred.label.over <- model.over %>% predict_classes(test.x)

cm.before <- table(test.label, pred.label)
cm.after <- table(test.label, pred.label.over)
```


Normalized Confusion Matrix (before oversampling)
Predicted
0
1
True
0
1
0.9575
0.0425
0.1765
0.8235



The ROC plot tells the same.

```
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 Checking")
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)
```



The high frequency trading dataset

The dataset `Dataset_HFT` has been introduced in the `Data loading & oversampling` section. Here for a more detailed checking on OSTSC function, we extracted 3000 observations instead of 300 from the original high frequency trading dataset. We split the training and setting data by ratio 2:1. The first 2000 observations are training data, while the rest are testing.

```
data(Dataset_HFT_Check)

label <- Dataset_HFT_Check$y
sample <- Dataset_HFT_Check$x

train.label <- label[1:2000]
train.sample <- sample[1:2000, ]
test.label <- label[2001:3000]
test.sample <- sample[2001:3000, ]
```

The imbalance of dataset is still 1:48:1.

```
table(train.label)
```

```
## train.label
##   -1    0    1
##   40 1926   34
```

After oversampling the data is balanced.

```r
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label

table(over.label)
```

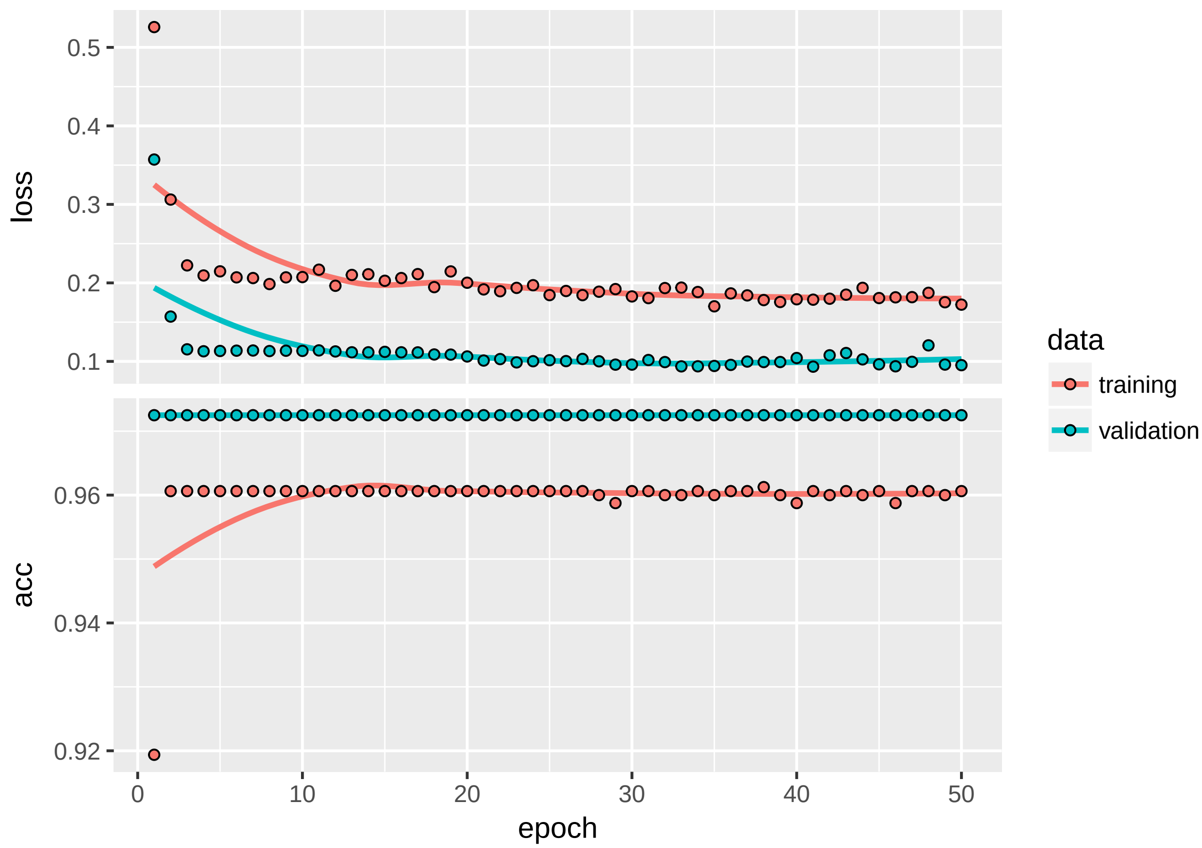
And then we test the oversampling performance on LSTM.

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.

```r
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.

```r
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 = 50
)
cat("\n Plot the accuracy and loss changes during the LSTM: \n\n")
plot(lstm.before)
```



3. Evaluate the model.

```
score <- model %>% evaluate(test.x, test.y)
```

```
## The loss value is 0.1756233 .
```

```
## The metric value (in this case 'accuracy') is 0.954 .
```

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

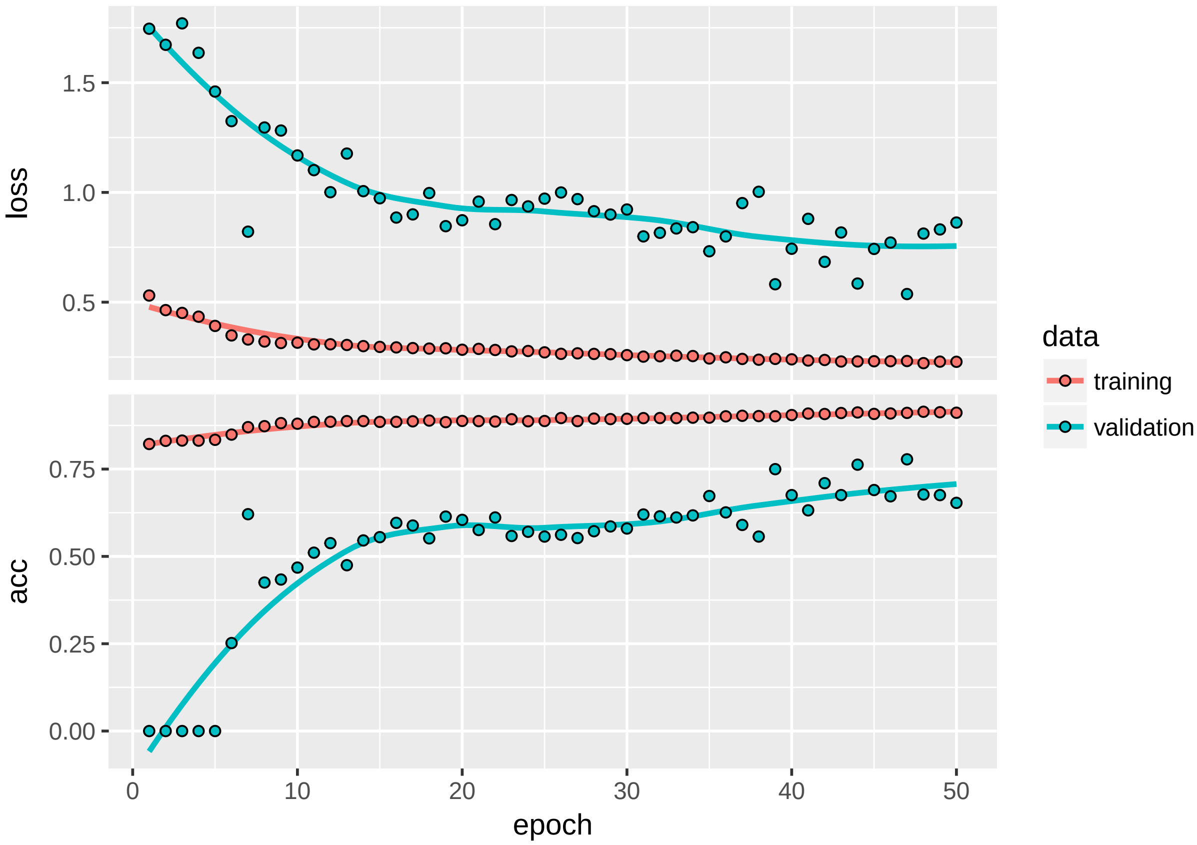
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
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 = 0.2,
  epochs = 50
)
cat("\n Plot the accuracy and loss changes during the LSTM: \n\n")
plot(lstm.after)
```



3. Evaluate the model.

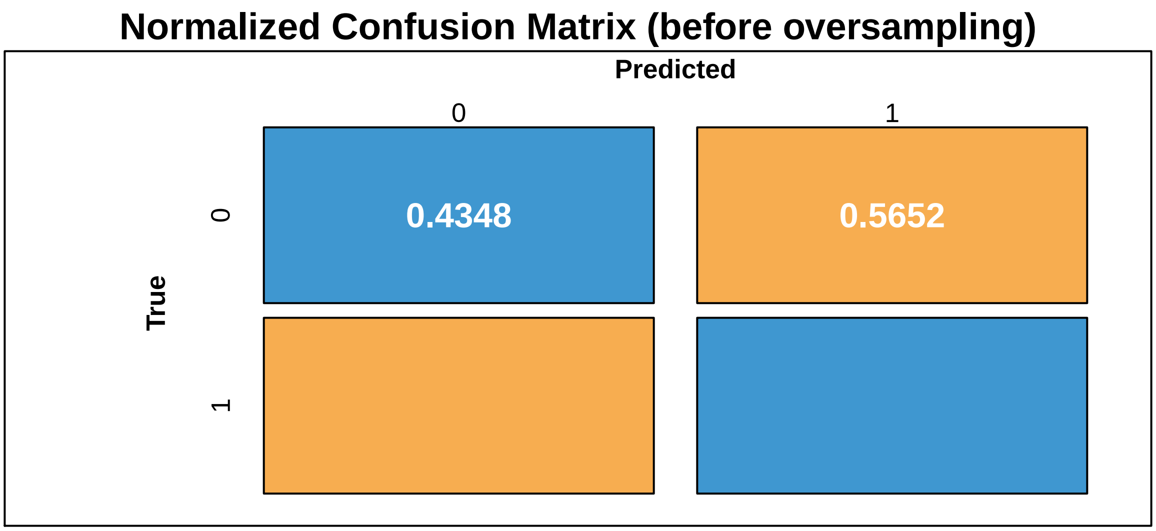
```
score.over <- model.over %>% evaluate(test.x, test.y)
```

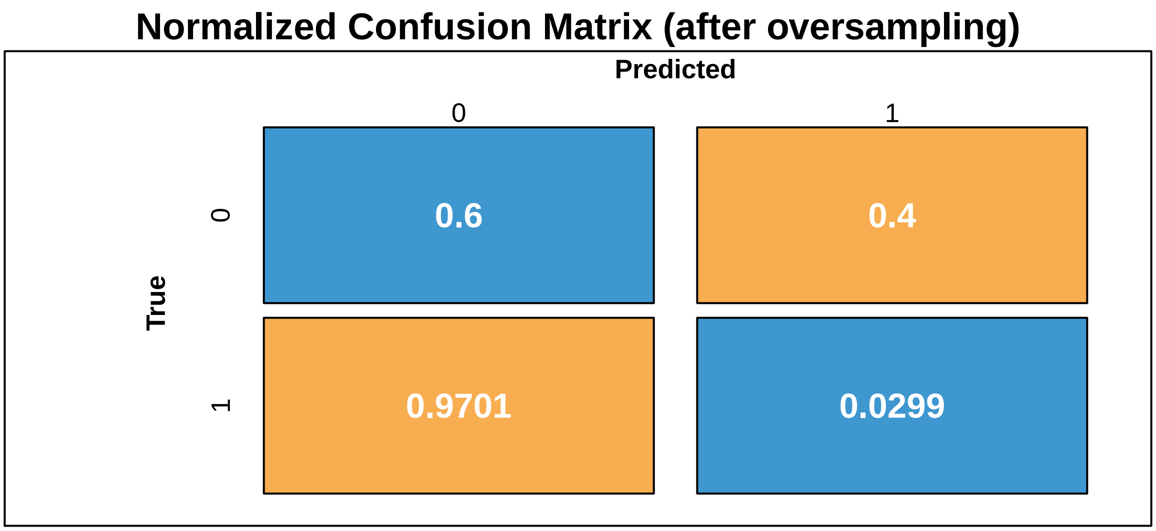
```
## The loss value is 1.065939 .
```

```
## The metric value (in this case 'accuracy') is 0.593 .
```

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)
```


Normalized Confusion Matrix (before oversampling)
Predicted
0
1
True
0
1
0.4348
0.5652



The ROC plot tells the same.

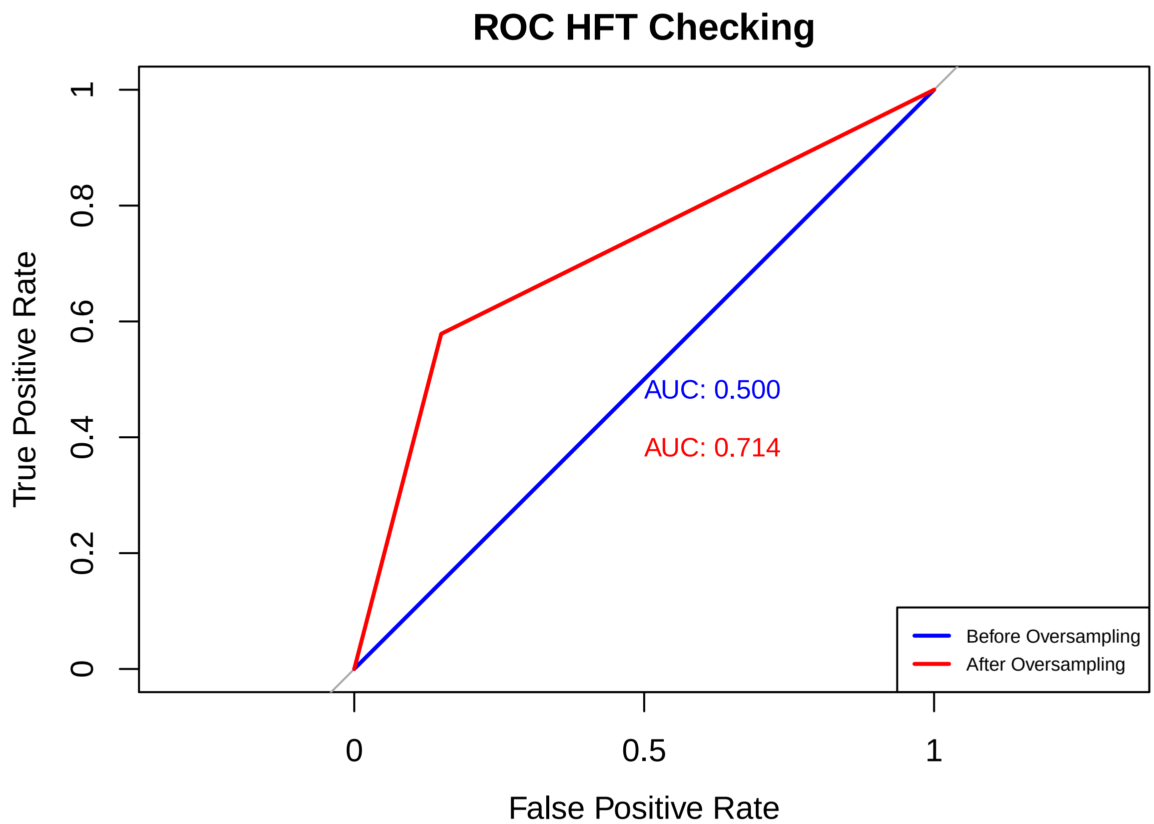
```
library(pROC)
plot.roc(as.vector(test.label), pred.label, legacy.axes = TRUE, col = "blue", print.auc = TRUE,
         print.auc.cex= .8, xlab = 'False Positive Rate', ylab = 'True Positive Rate',
         main="ROC HFT Checking")
```

```
## Warning in roc.default(x, predictor, plot = TRUE, ...): 'response' has
## more than two levels. Consider setting 'levels' explicitly or using
## 'multiclass.roc' instead
```

```
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)
```

```
## Warning in roc.default(x, predictor, plot = TRUE, ...): 'response' has
## more than two levels. Consider setting 'levels' explicitly or using
## 'multiclass.roc' instead
```

```
legend("bottomright", legend=c("Before Oversampling", "After Oversampling"),
       col=c("blue", "red"), lwd=2, cex= .6)
```



Evaluating OSTSC on large datasets

In the evaluation section, we uses the MHEALTH dataset and HFT dataset again. But for evaluation, we increases the data sizes by 5 or 10 times. The evaluation process takes about two hours on one dataset on a four-cores laptop because of the LSTM process.

The MHEALTH dataset

Instead of using only subject 1, here we uses subject 1-5. All other features kept same.

```
data(Dataset_MHEALTH_Eval)

train.label <- Dataset_MHEALTH_Eval$train.y
train.sample <- Dataset_MHEALTH_Eval$train.x
test.label <- Dataset_MHEALTH_Eval$test.y
test.sample <- Dataset_MHEALTH_Eval$test.x
```

Class 1 stands for positive data, while class 0 stands for negative. The imbalance of the train dataset is 1:42.

```
table(train.label)
```

```
## train.label
##     0     1
## 10584   255
```

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

```r
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label

table(over.label)
```

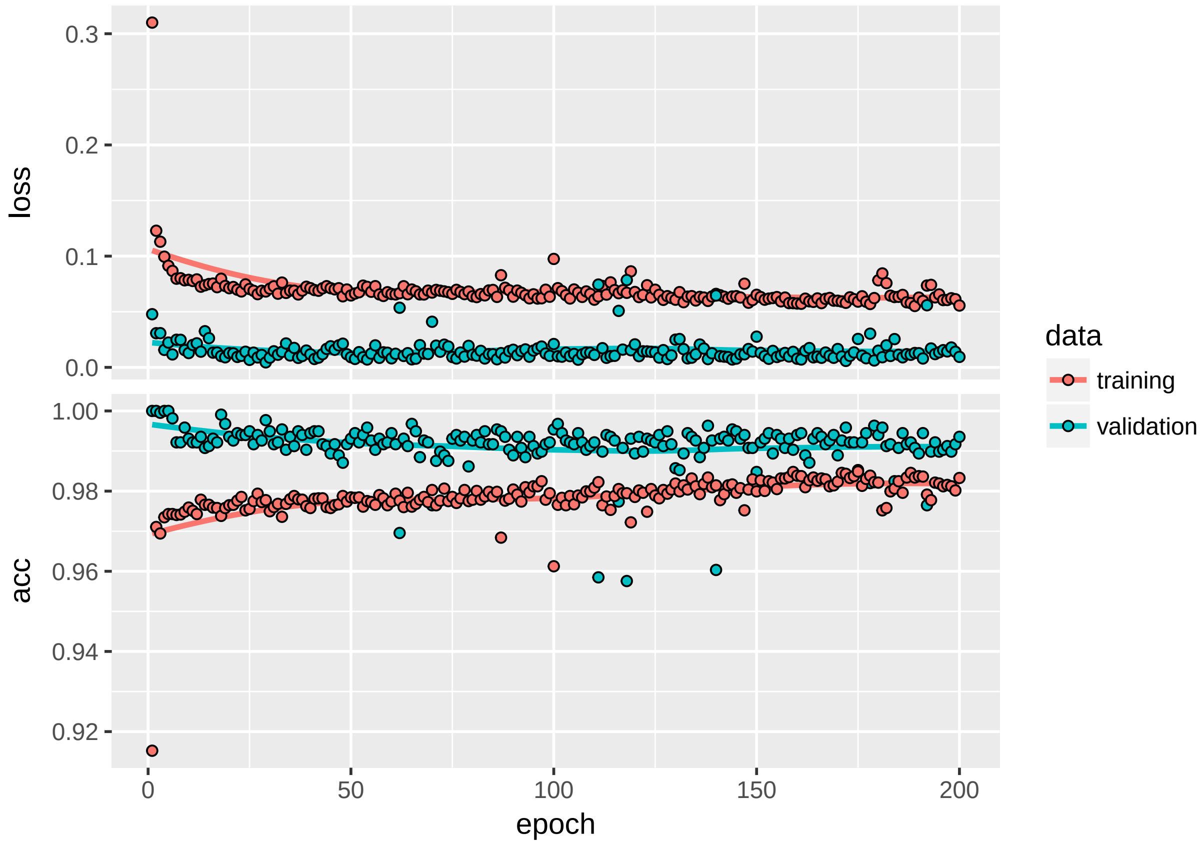
For comparison, we use the same Long short-term memory (LSTM) classifier.

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

```r
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))
```

```r
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)
```

```
##
##  Plot the accuracy and loss changes during the LSTM:
```

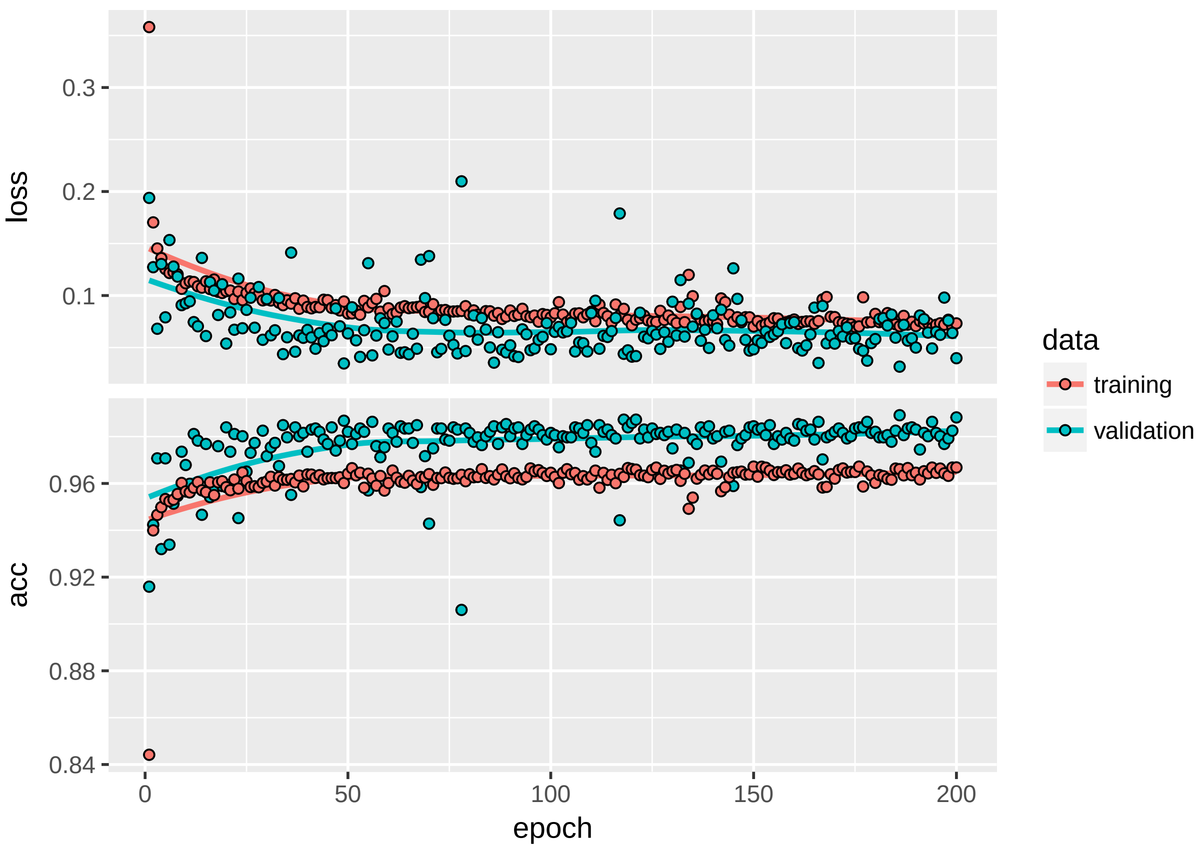


2. To train the classifier on the new data after oversampling.

```
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)
```

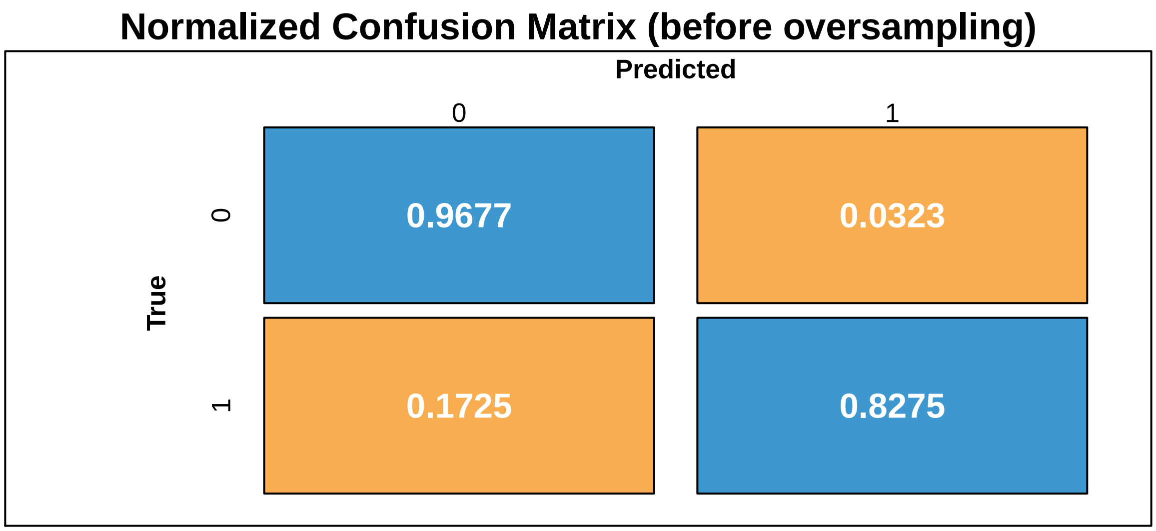
```
##
##  Plot the accuracy and loss changes during the LSTM:
```

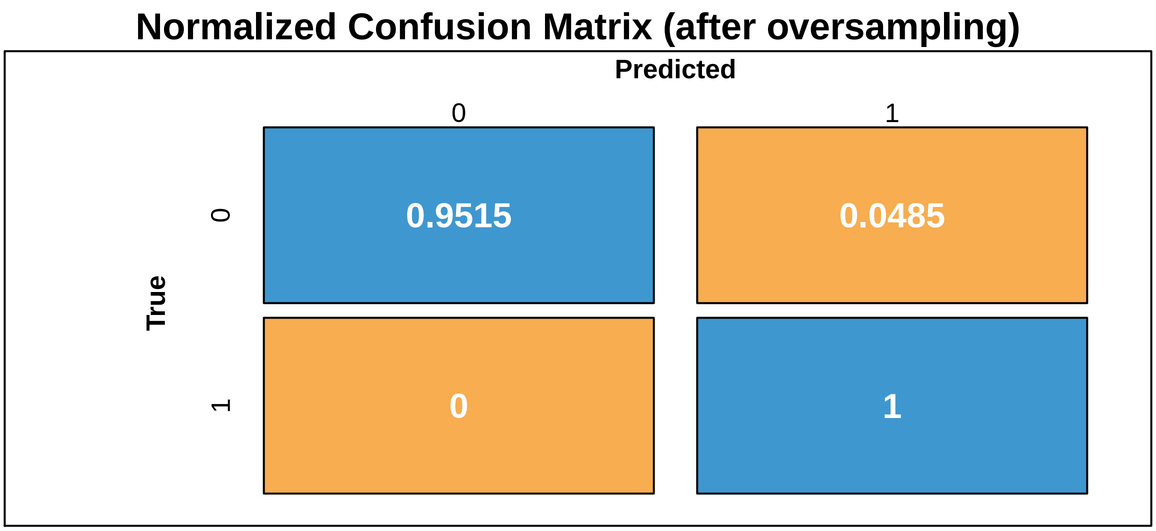


3. To compare the confusion matrices and ROC plot. The power of OSTSC will come out more on more training epochs.

```
pred.label <- model %>% predict_classes(test.x)
pred.label.over <- model.over %>% predict_classes(test.x)
```

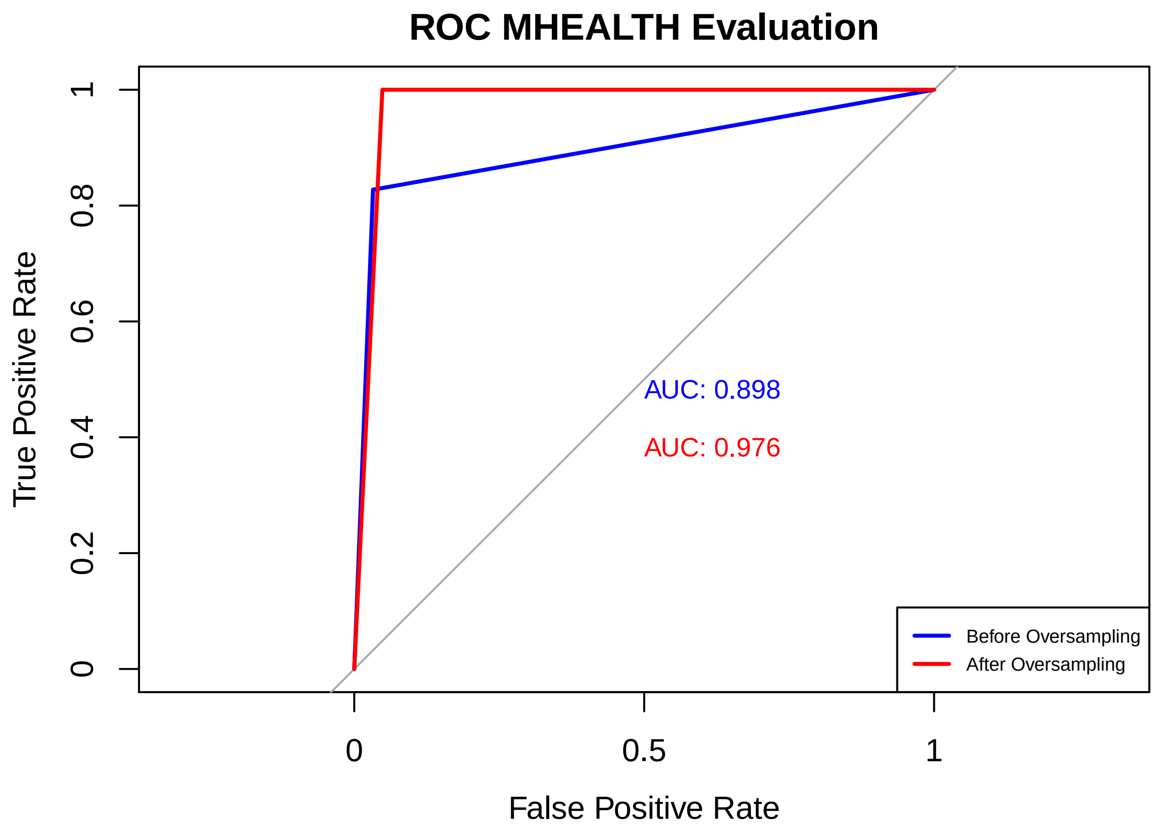
```
cm.before <- table(test.label, pred.label)
cm.after <- table(test.label, pred.label.over)
```


Normalized Confusion Matrix (before oversampling)
Predicted
0
1
True
0
1
0.9677
0.0323
0.1725
0.8275



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 Evaluation")
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)
```



The high frequency trading dataset

We extracted 30000 observations from the original high frequency trading dataset for evaluation. We split the training and setting data by ratio 1:1. The first half observations are training data, while the rest are testing.

```
data(Dataset_HFT_Eval)

label <- Dataset_HFT_Eval$y
sample <- Dataset_HFT_Eval$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 still 1:48:1.

```
table(train.label)
```

```
## train.label
##    -1     0     1
##   297 14424   279
```

After oversampling the data is fully balanced.

```
MyData <- OSTSC(train.sample, train.label, parallel = FALSE)
over.sample <- MyData$sample
over.label <- MyData$label
```

```
table(over.label)
```

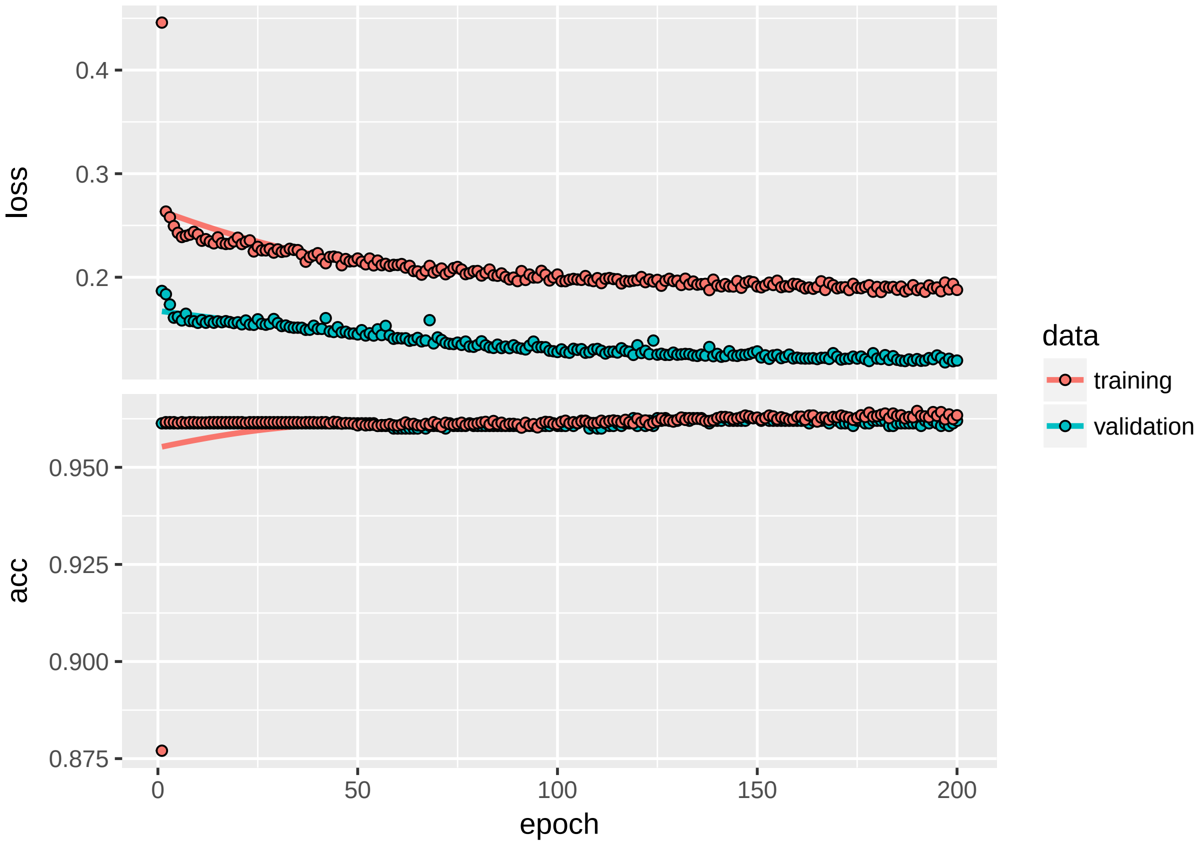
As same as elder examples, we use the same Long short-term memory (LSTM) classifier.

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)
```

```
##
##  Plot the accuracy and loss changes during the LSTM:
```

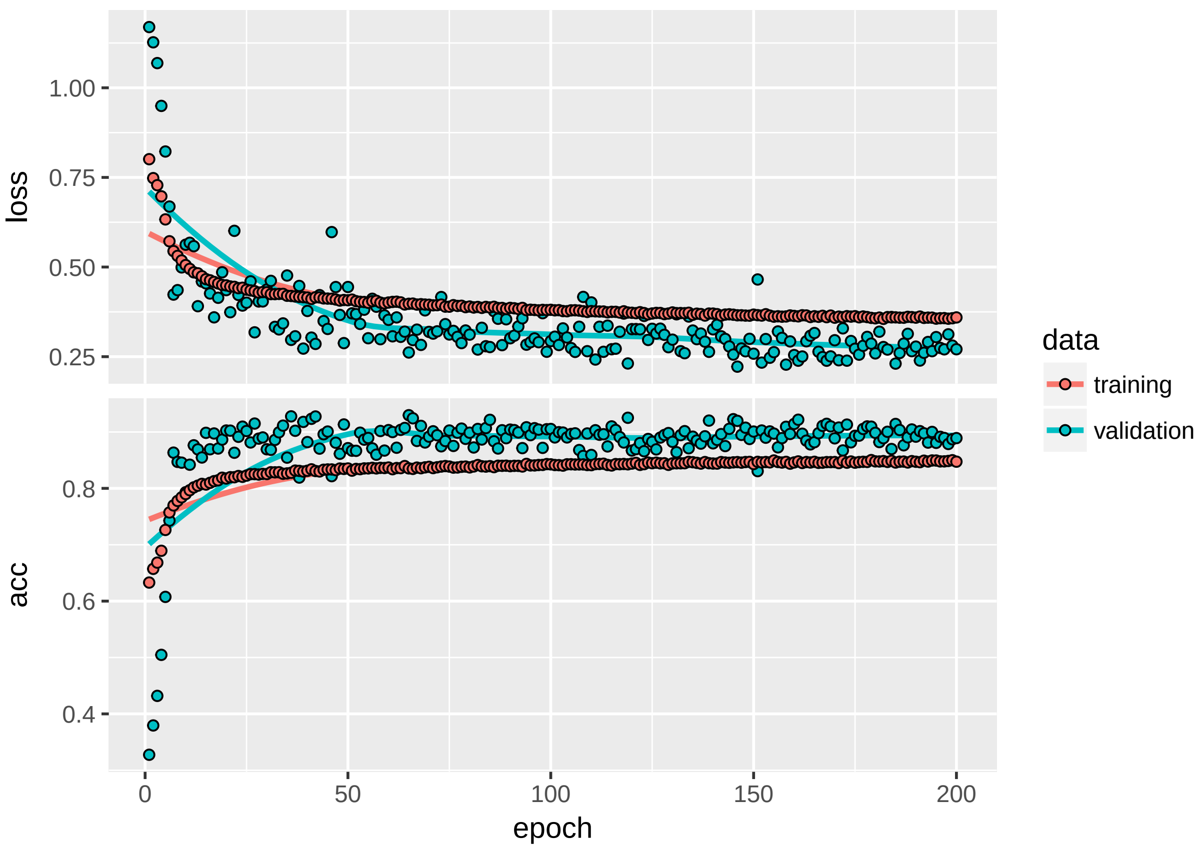


2. To train the classifier on the new data after oversampling.

```
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)
```

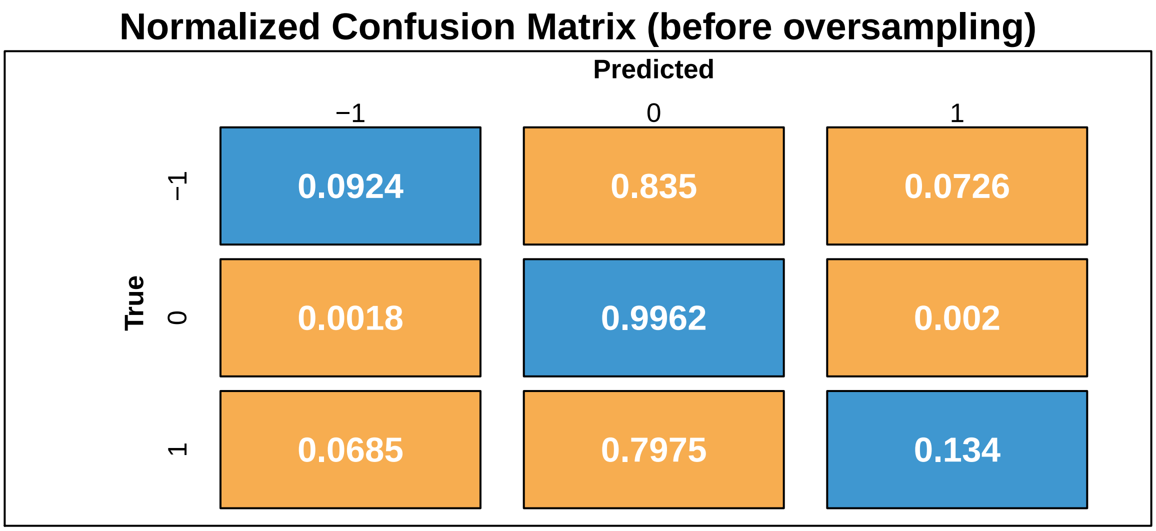
```
##
##  Plot the accuracy and loss changes during the LSTM:
```

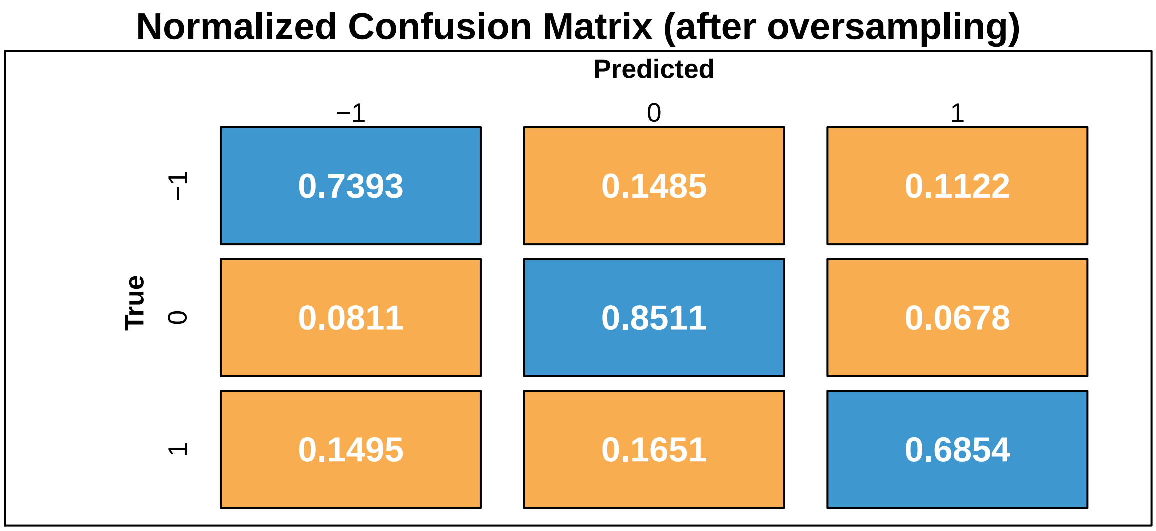


3. To compare the confusion matrices and ROC plot. When the dataset size gets larger and the imbalance degree gets more severe, the OSTSC performs better than unoversampled data.

```
pred.label <- model %>% predict_classes(test.x)
pred.label.over <- model.over %>% predict_classes(test.x)
```

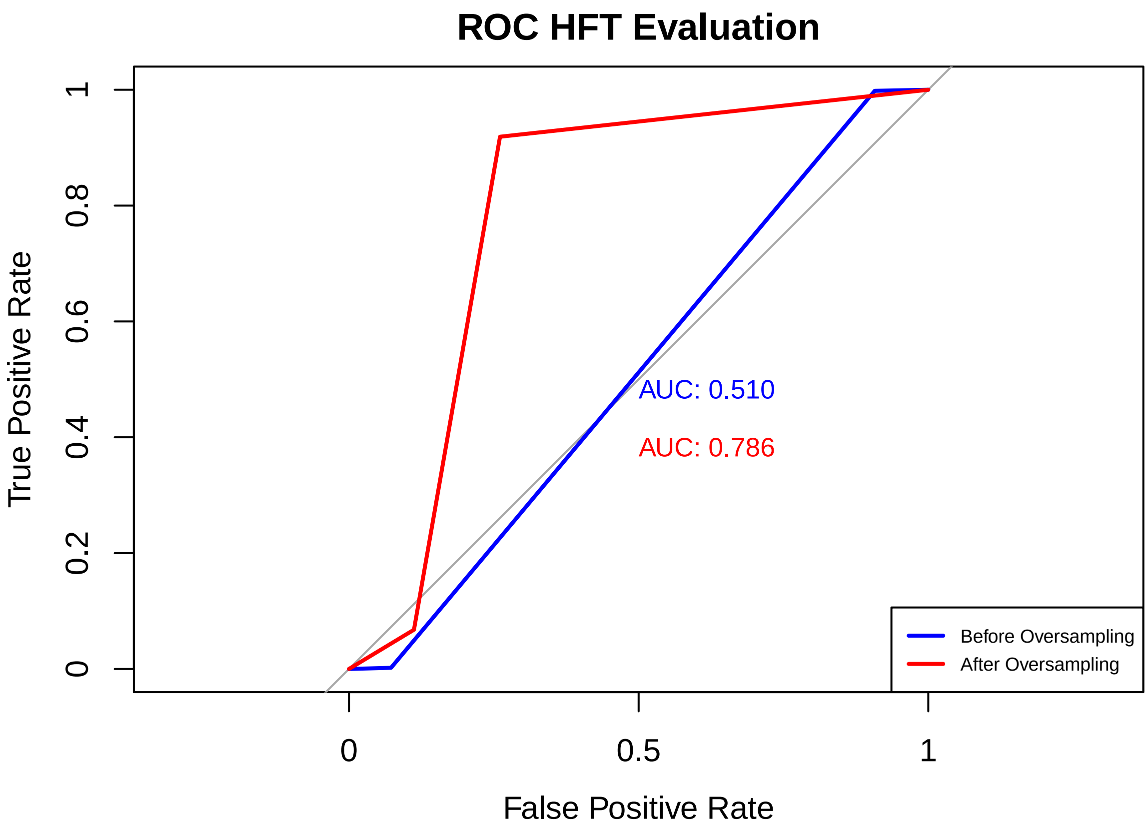
```
cm.before <- table(test.label, pred.label)
cm.after <- table(test.label, pred.label.over)
```


Normalized Confusion Matrix (before oversampling)
Predicted
−1
0
1
True
−1
0
1
0.0924
0.835
0.0726
0.0018
0.9962
0.002
0.0685
0.7975
0.134



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

```r
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 HFT Evaluation")
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)
```



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

References


```
## [1] O. Banos, C. Villalonga, R. Garcia, et al. "Design,
## implementation and validation of a novel open framework for agile
## development of mobile health applications". In: _BioMedical
## Engineering OnLine_ 14.2 (Aug. 2015), p. S6. ISSN: 1475-925X.
##
## [2] H. Cao, X. Li, D. Y. Woon, et al. "Integrated Oversampling for
## Imbalanced Time Series Classification". In: _IEEE Transactions on
## Knowledge and Data Engineering_ 25 (2013), pp. 2809-2822.
##
## [3] H. Cao, X. Li, D. Y. Woon, et al. "SPO: Structure Preserving
## Oversampling for Imbalanced Time Series Classification". In: _2011
## IEEE 11th International Conference on Data Mining_ (2011), pp.
## 1008-1013.
##
## [4] H. Cao, V. Y. F. Tan and J. Z. F. Pang. "A Parsimonious
## Mixture of Gaussian Trees Model for Oversampling in Imbalanced and
## Multimodal Time-Series Classification". In: _IEEE Transactions on
## Neural Networks and Learning Systems_ 25 (2014), pp. 2226-2239.
##
## [5] R. Dubey, J. Zhou, Y. Wang, et al. "Analysis of sampling
## techniques for imbalanced data: An n = 648 ADNI study". In:
```

```
## _NeuroImage_ 87 (2014), pp. 220-41.
##
## [6] G. Liang and C. Zhang. "A Comparative Study of Sampling
## Methods and Algorithms for Imbalanced Time Series Classification".
## In: _AI 2012: Advances in Artificial Intelligence: 25th
## Australasian Joint Conference, Sydney, Australia, December 4-7,
## 2012._. Ed. by M. Thielscher and D. Zhang. Berlin, Heidelberg:
## Springer Berlin Heidelberg, 2012, pp. 637-648. ISBN:
## 978-3-642-35101-3.
##
## [7] A. More. "Survey of resampling techniques for improving
## classification performance in unbalanced datasets". In: _ArXiv
## e-prints_ (Aug. 2016). arXiv: 1608.06048 [stat.AP].
```