Over Sampling for Time Series Classification

Matthew F. Dixon, Diego Klabjan and Lan Wei

2017-09-26

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, binary, 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 $P = \left\{x_{11}, x_{12}, ..., x_{1|P|}\right\}$ 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 $\Gamma_i = \left|S_{i:k-NN} \bigcap N_t\right| / Z$ 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.

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

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

The OSTSC also performs well on this extremely imbalance dataset.

```
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
##   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.

```
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.

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

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

The imbalance of dataset is 1:48:1.

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

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

Here we oversamples all the minority class.

```
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.

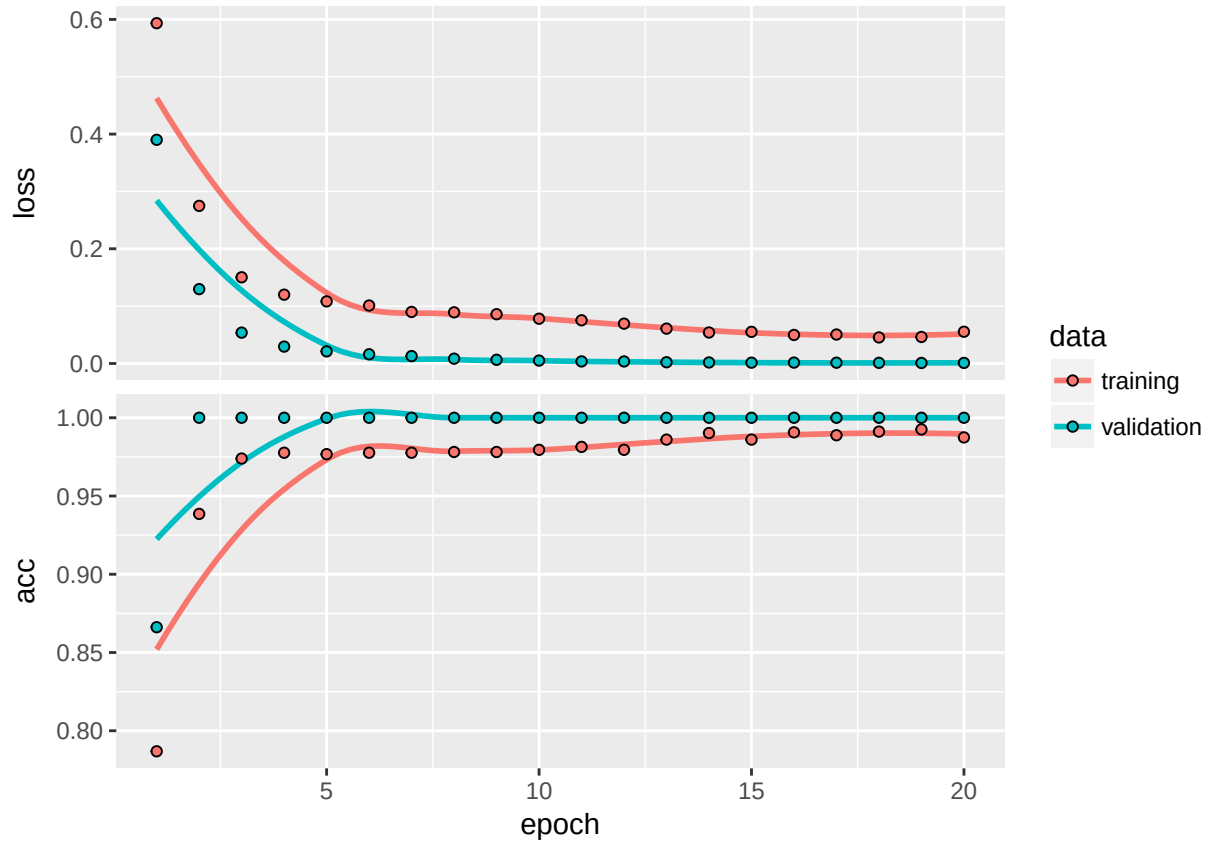
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
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 = 0.2,
```

```
  epochs = 20
)
plot(lstm.before)
```



3. Evaluate the model.

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

```
## The loss value is 0.09204124 .
```

```
## The metric value (in this case 'accuracy') is 0.9449405 .
```

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

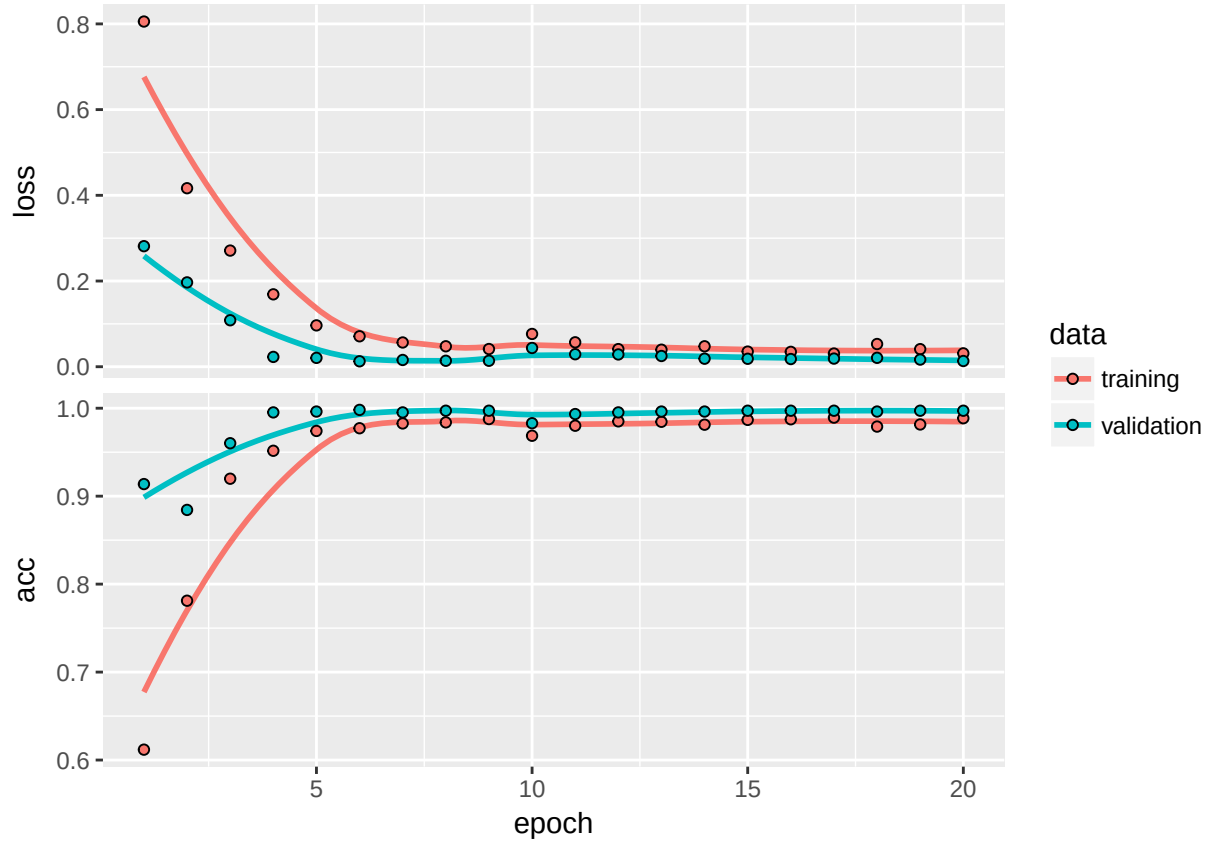
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
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 = 20
)
plot(lstm.after)
```



3. Evaluate the model.

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

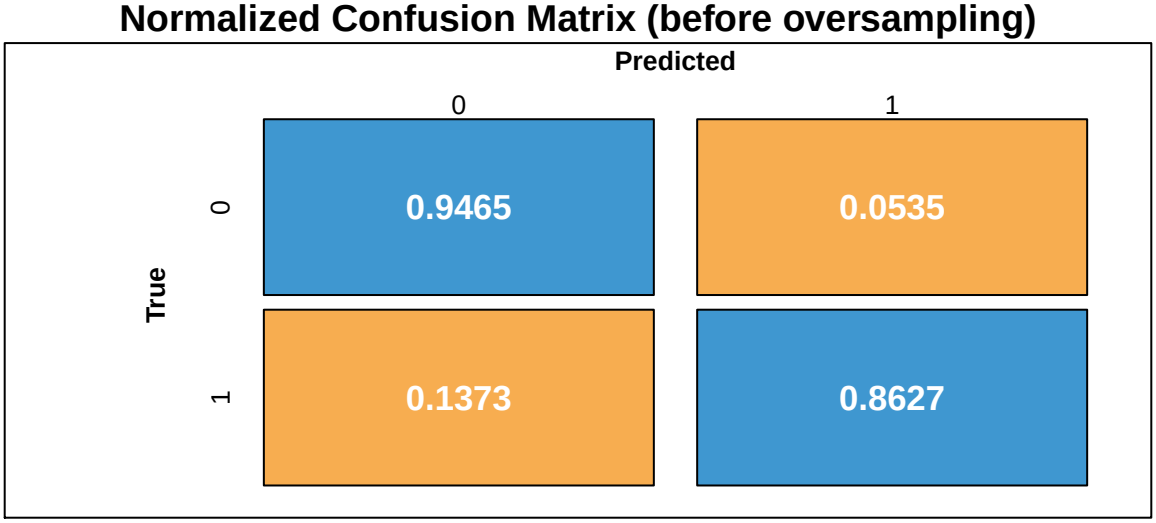
```
## The loss value is 0.369708 .
```

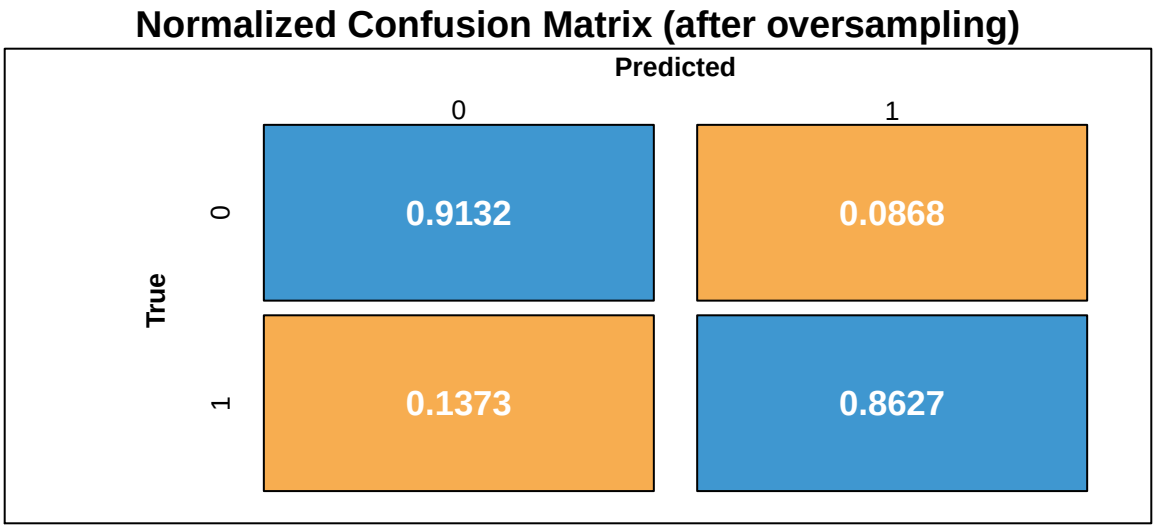
```
## The metric value (in this case 'accuracy') is 0.9122024 .
```

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)
```

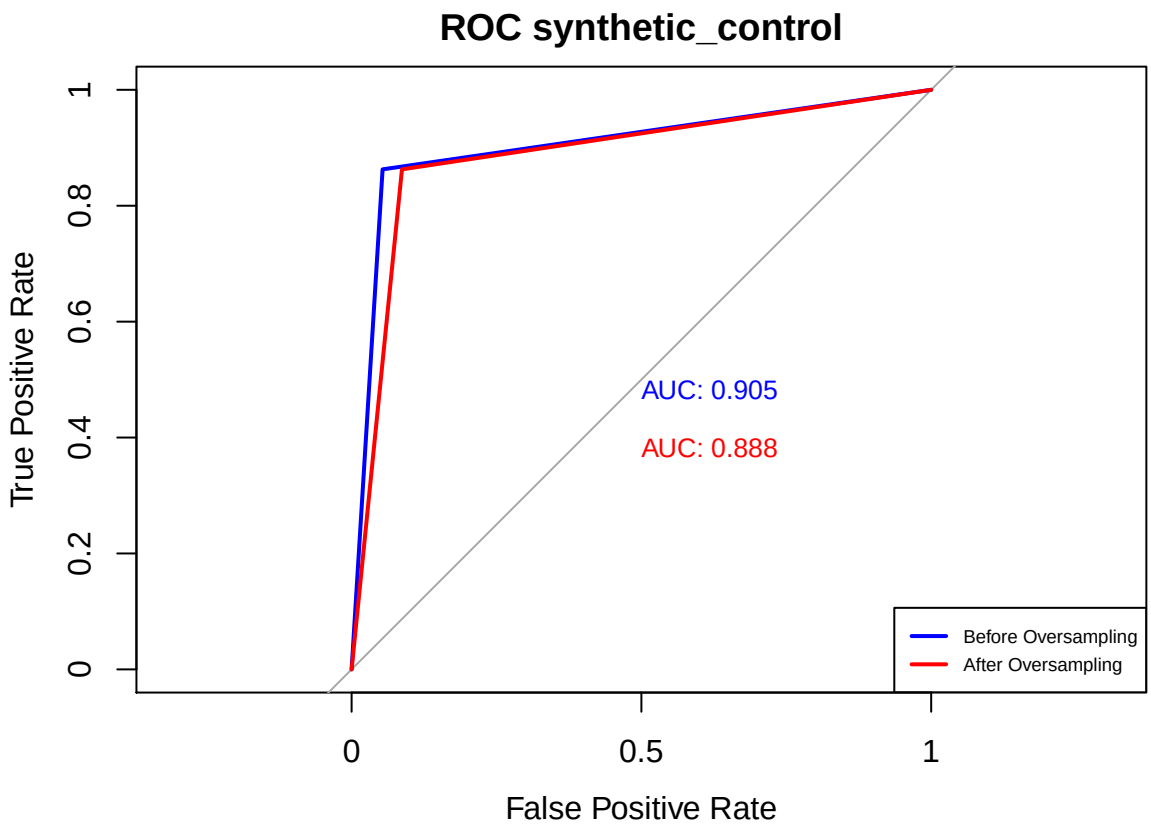
```
cm.before <- table(test.label, pred.label)
cm.after <- table(test.label, pred.label.over)
```





The ROC plot tells the same.

```
library(pROC)
plot.roc(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 synthetic_control")
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 fully balanced.

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

table(over.label)
```

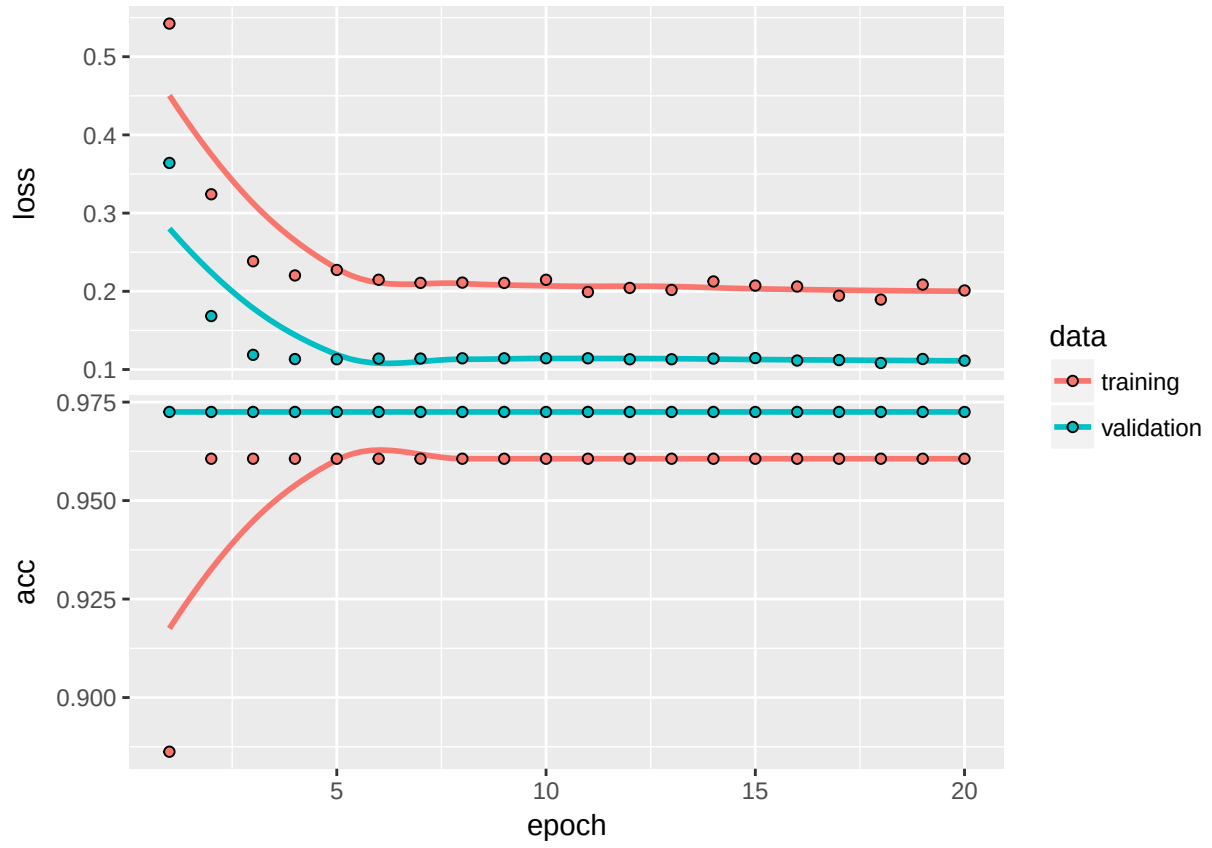
And then we test the oversampling performance on LSTM.

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.

```
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 = 0.2,
  epochs = 20
)
plot(lstm.before)
```



3. Evaluate the model.

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

```
## The loss value is 0.1752301 .
```

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

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

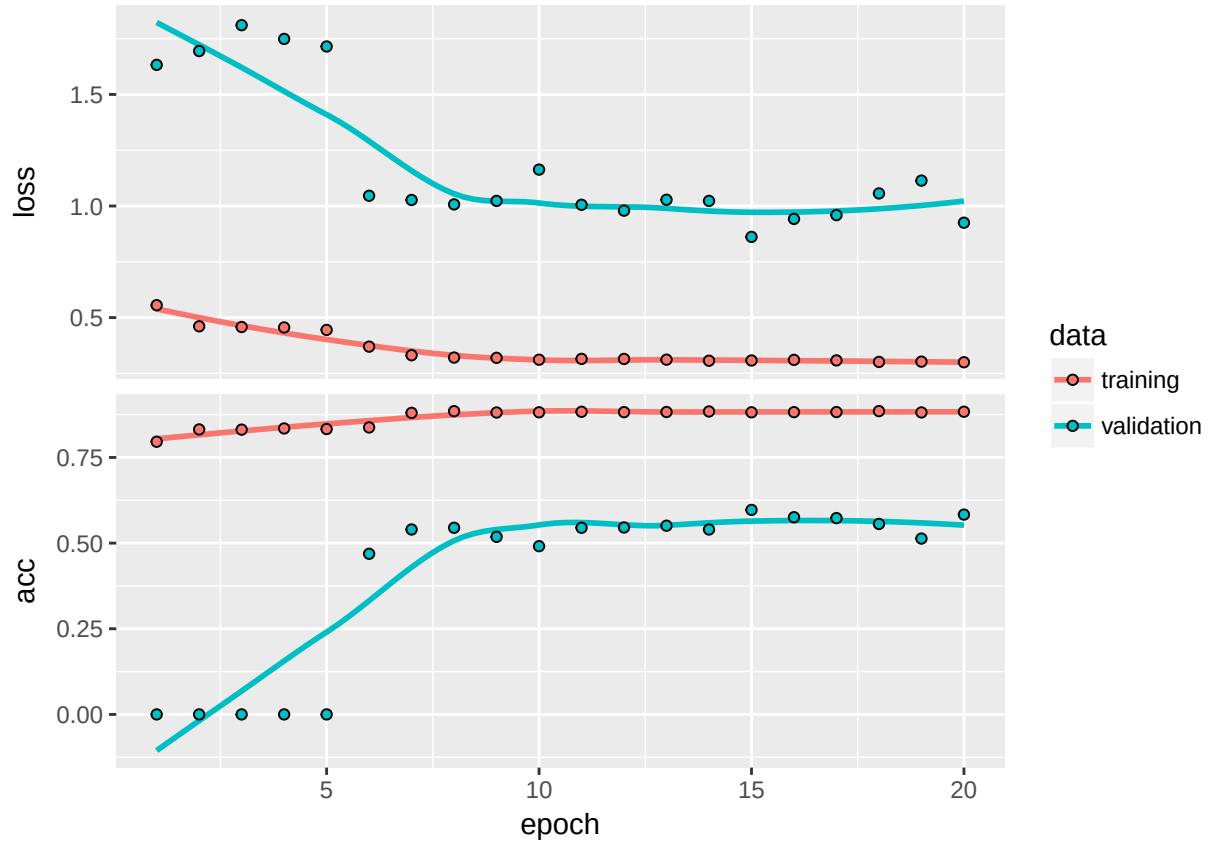
1. One-hot encode the label vectors into binary class matrices using the Keras to_categorical() function. And transform the sample array to 3-dimension for LSTM.

```
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 = 20
)
plot(lstm.after)
```



3. Evaluate the model.

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

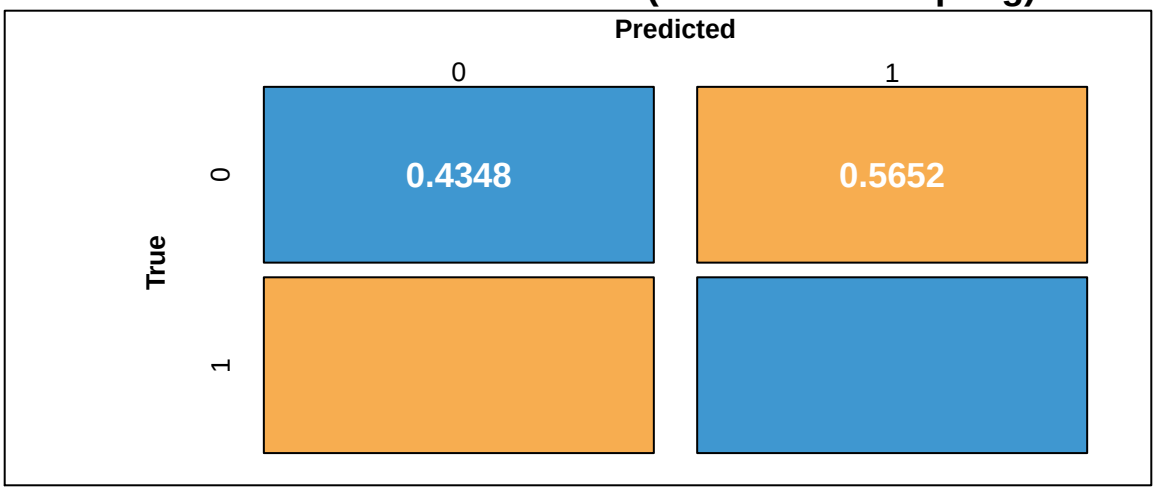
```
## The loss value is 1.049027 .
```

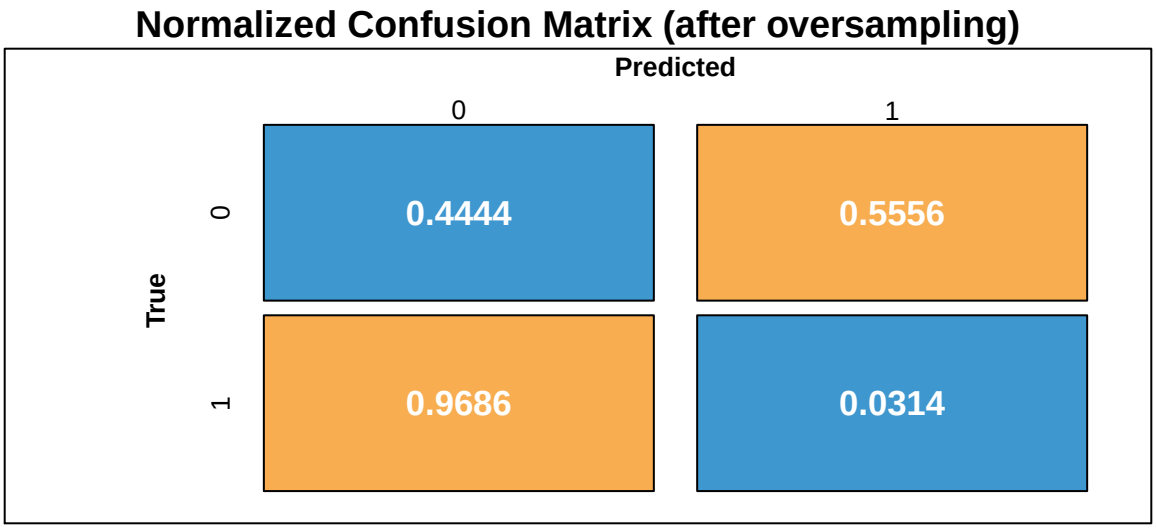
```
## The metric value (in this case 'accuracy') is 0.53 .
```

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.

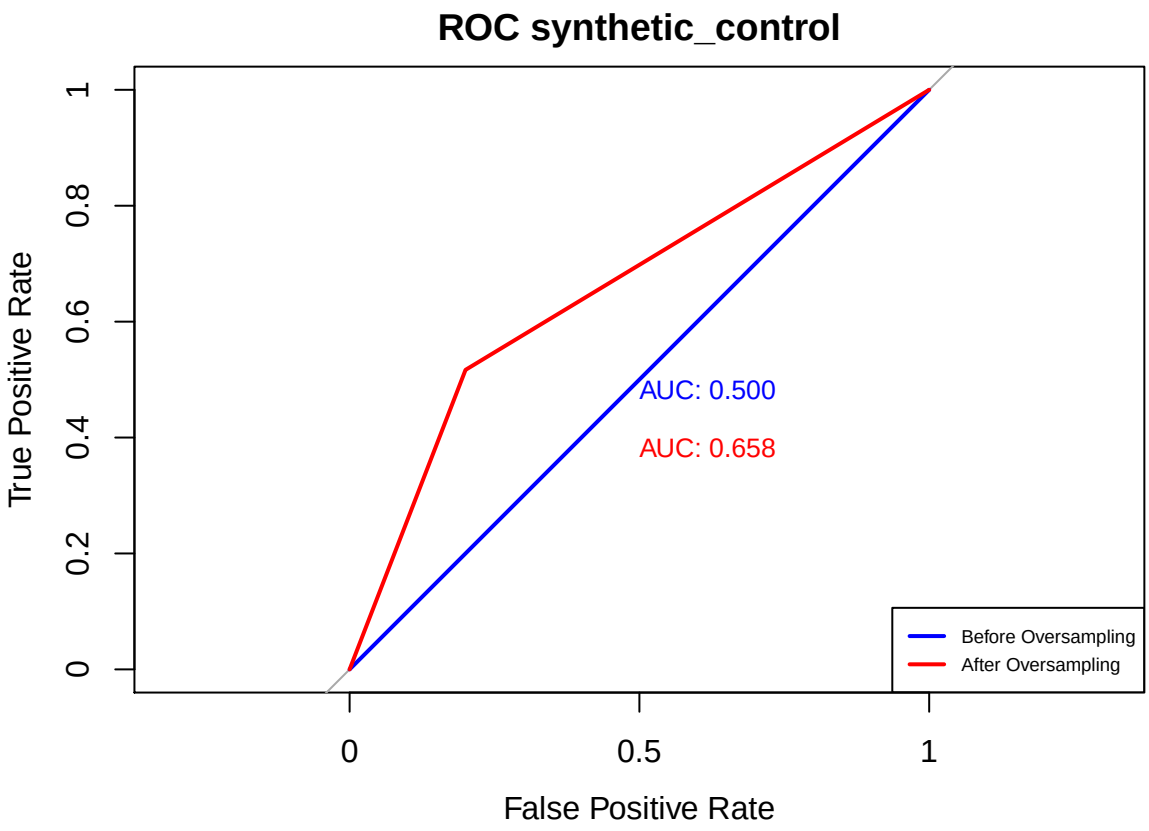
```
library(pROC)
plot.roc(as.vector(test.label), pred.label, legacy.axes = TRUE, col = "blue", print.auc = TRUE,
         print.auc.cex= .8, xlab = 'False Positive Rate', ylab = 'True Positive Rate',
         main="ROC synthetic_control")
```

```
## 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)
```

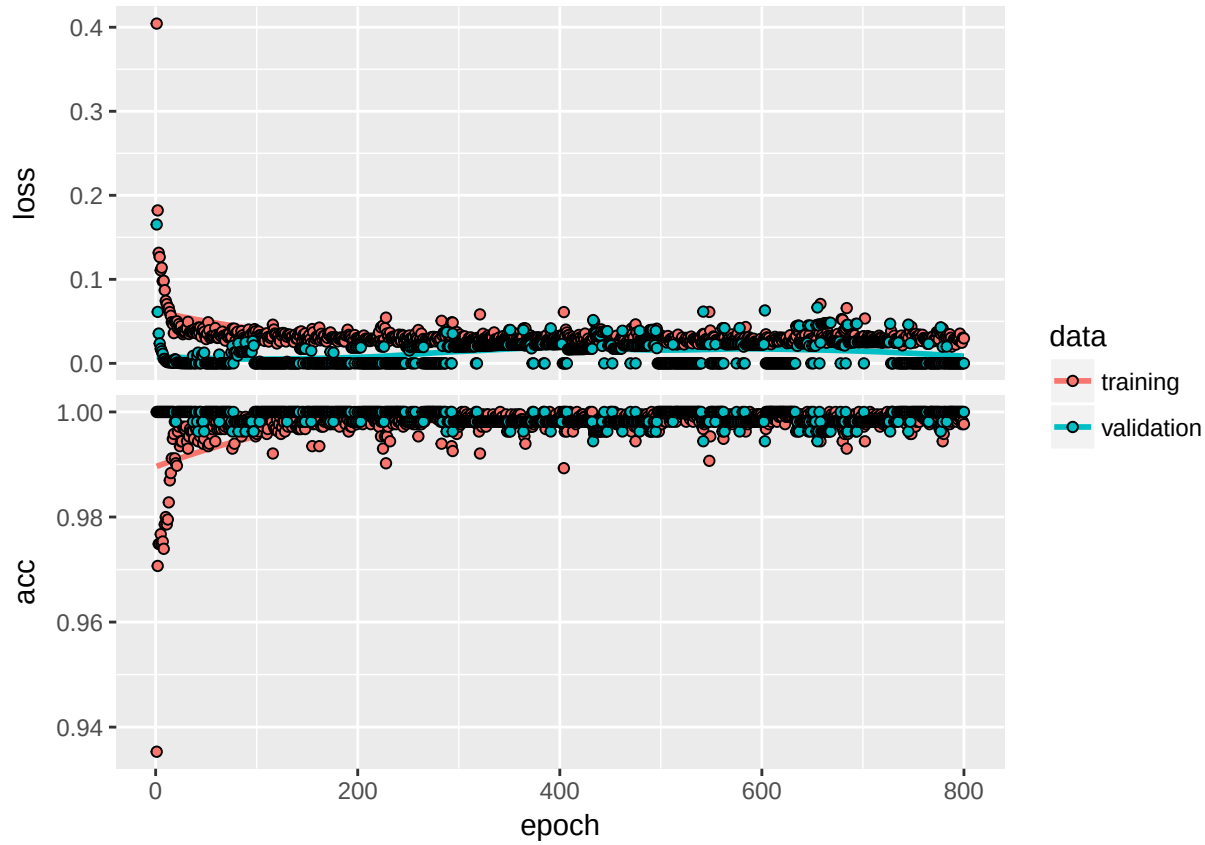


Evaluating OSTSC

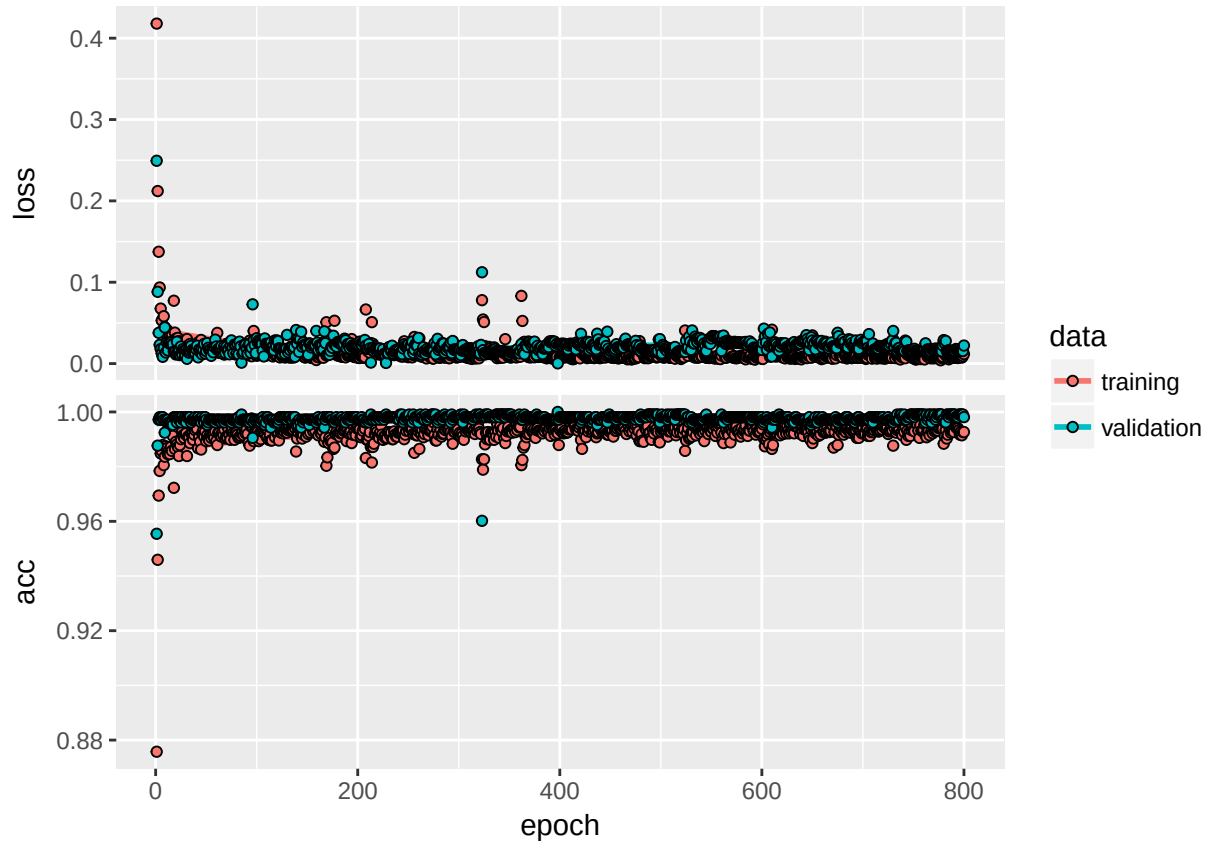
In the evaluation section, we uses the MHEALTH dataset and HFT dataset again. But for evaluation, we run LSTM to converge.

The MHEALTH dataset

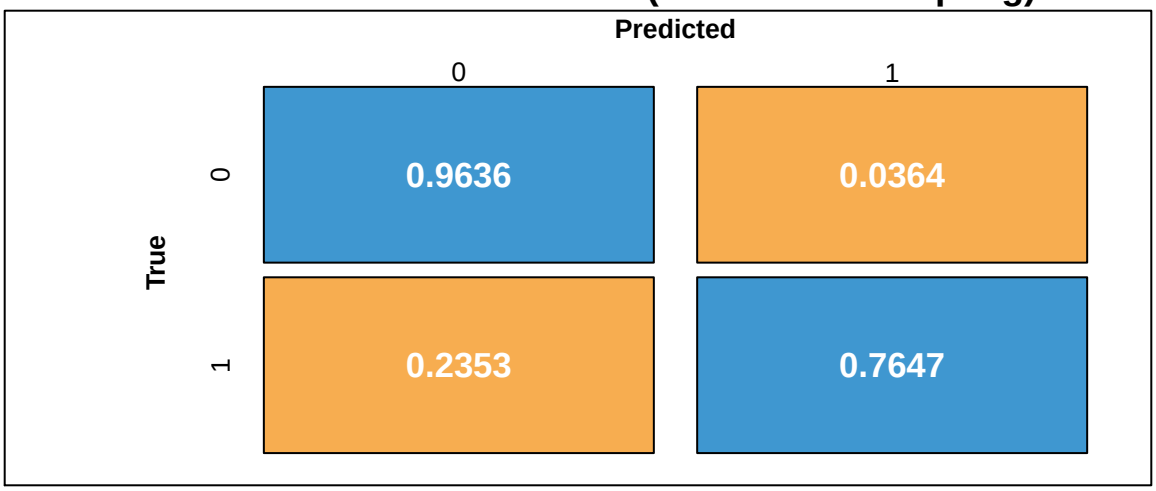
All the settings keep the same. But the epochs number of LSTM sets to 800.

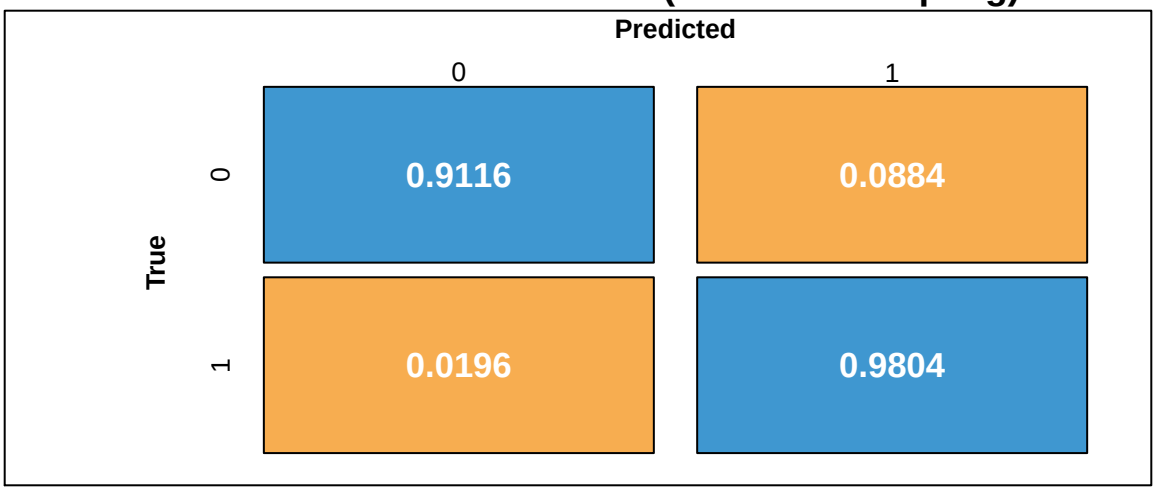


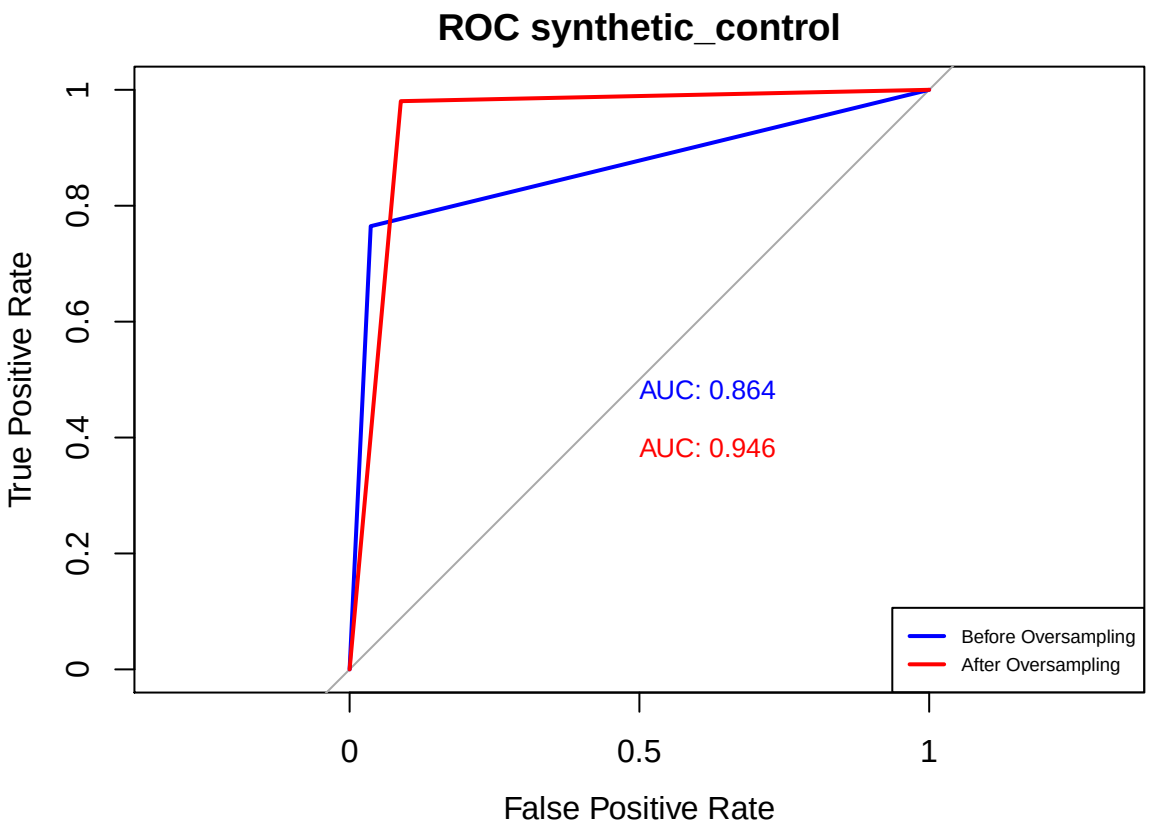

```
## For the original data:
## The loss value is 0.3860774 .
## The metric value (in this case 'accuracy') is 0.9598214 .
```




```
## for the oversampled data:
## The loss value is 1.006971 .
## The metric value (in this case 'accuracy') is 0.9129464 .
```

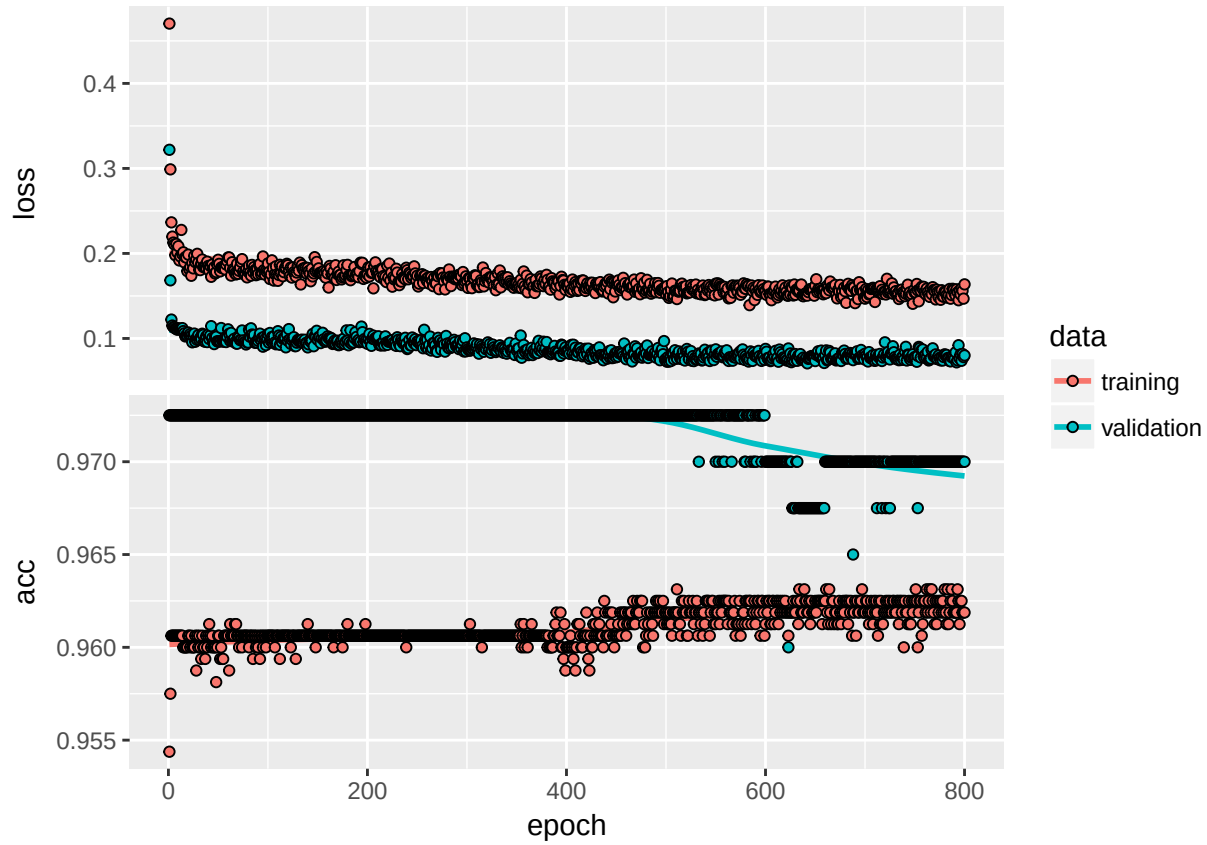

Normalized Confusion Matrix (before oversampling)
Predicted
0
1
True
0
1
0.9636
0.0364
0.2353
0.7647


Normalized Confusion Matrix (after oversampling)
Predicted
0
1
True
0
1
0.9116
0.0884
0.0196
0.9804

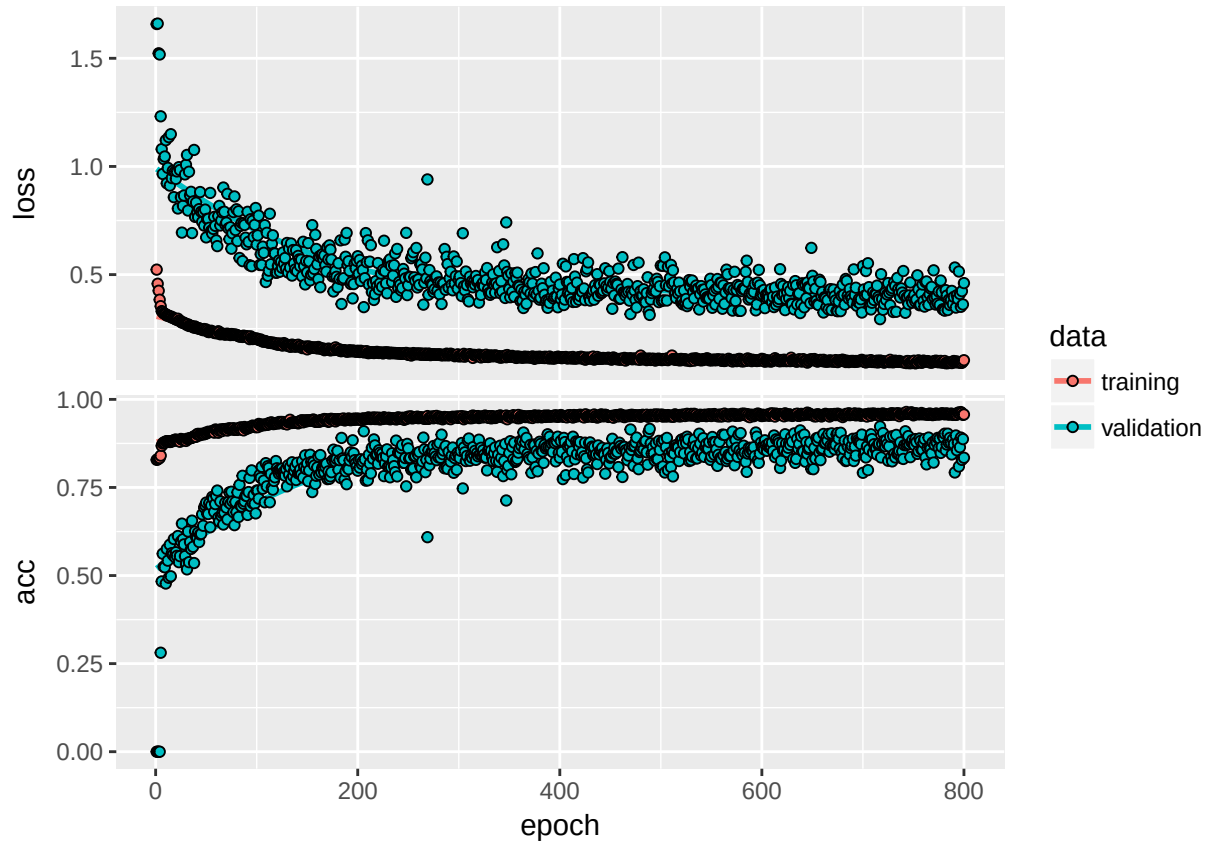


The high frequency trading dataset

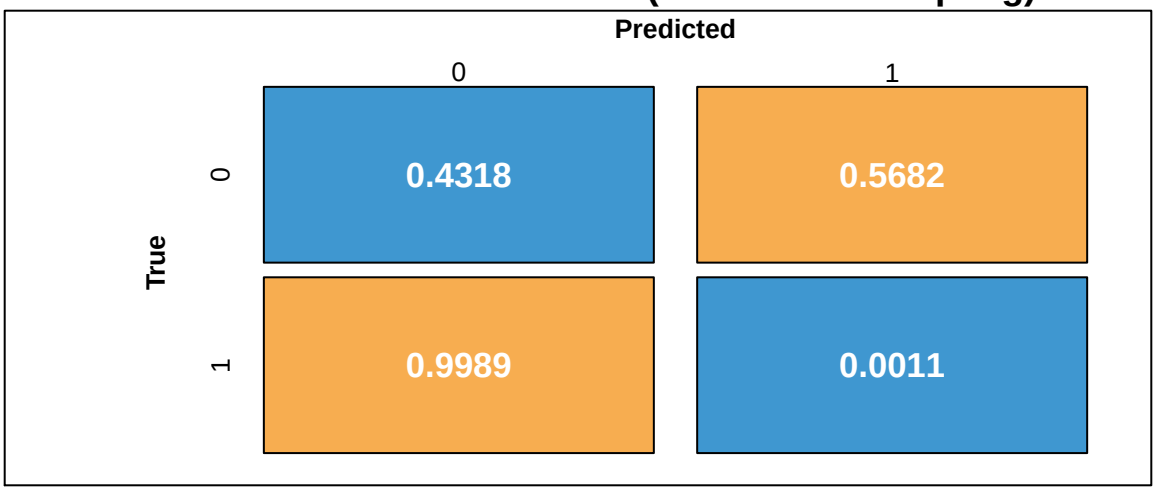
```
## train.label
##   -1    0    1
##   40 1926   34
```

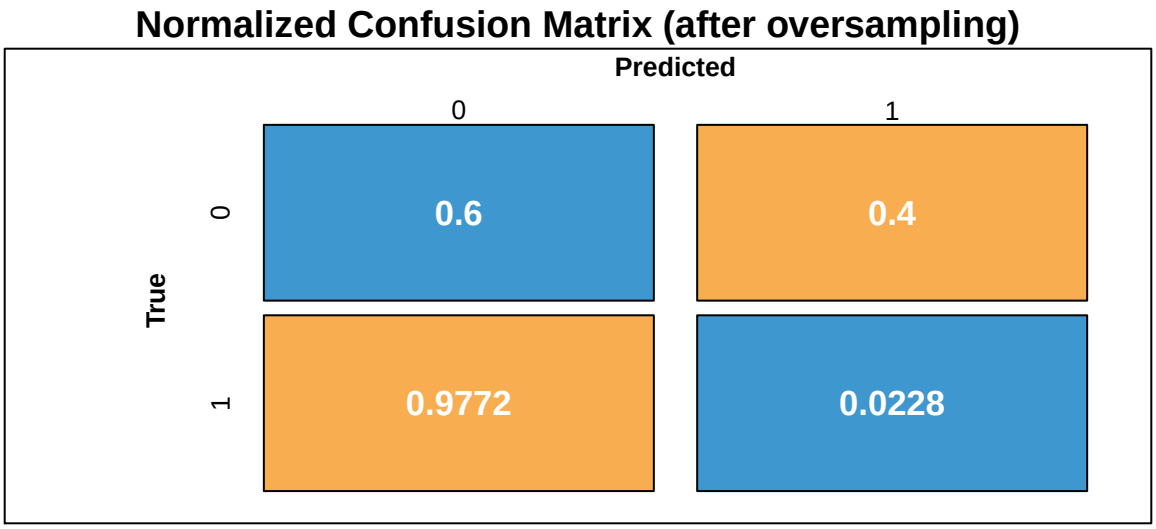



```
## For the original data:
## The loss value is 0.1837748 .
## The metric value (in this case 'accuracy') is 0.948 .
```



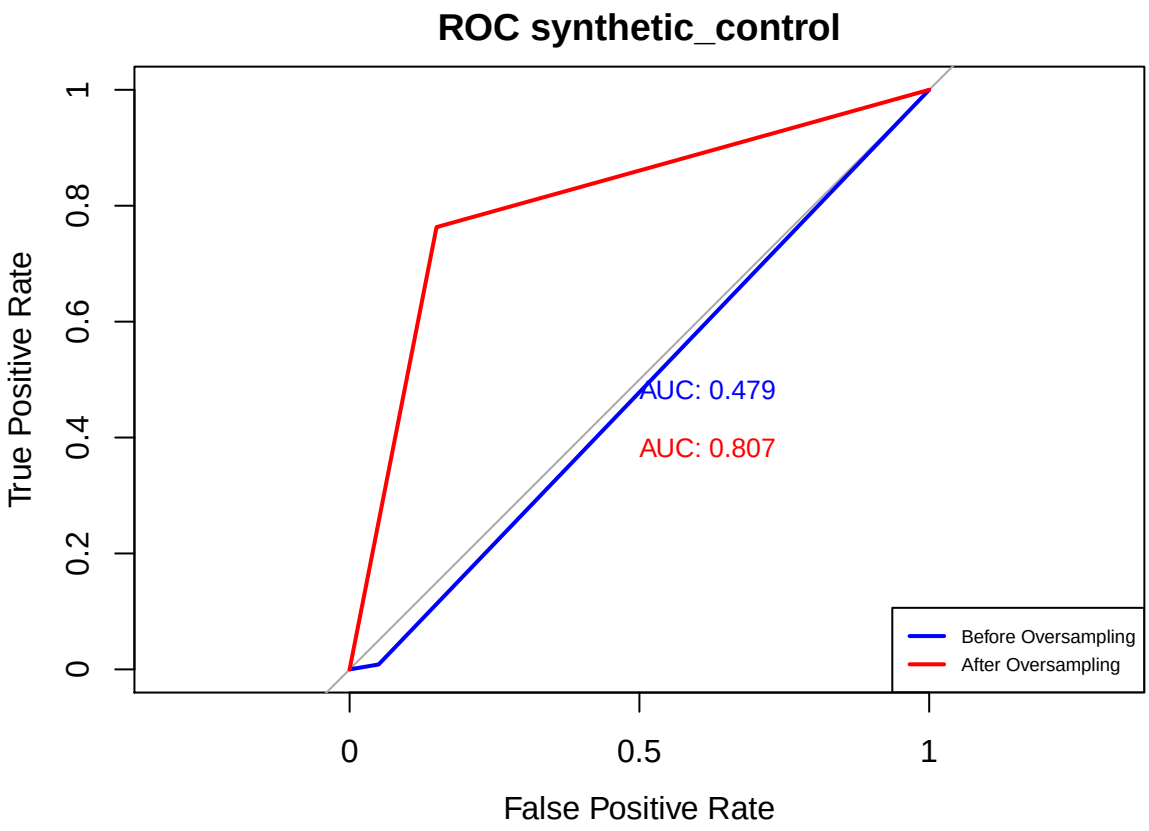

```
## For the oversampled data:
## The loss value is 0.6913902 .
## The metric value (in this case 'accuracy') is 0.769 .
```


Normalized Confusion Matrix (before oversampling)
Predicted
0
1
True
0
1
0.4318
0.5682
0.9989
0.0011




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

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



The evaluation may take hours (about 2h on a 4 cores laptop). There are also the larger datasets downloadable from github by command `OSTSC::MHEALTH` and `OSTSC::HFT`. Evaluations on these larger datasets take longer time.

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.