Performance Assessment

David Meyer

16.3.2022

Contents

```
library("mosaicData")
library("caret")
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library("e1071")
library("ROCR")
```

---

1 Create data partitions

```
library(caret)
data("RailTrail")
set.seed(4711)
```

1.1 Train/Test-split

```
part = createDataPartition(RailTrail$volume, times = 2, p = 2/3)
part
```

```
## $Resample1
##  [1]  1  3  5  7  8  9 10 11 12 14 15 16 17 18 19 20 21 23 24 25 26 27 28 29 30
## [26] 34 35 39 41 42 43 45 46 47 48 49 50 51 52 53 54 56 57 58 59 61 62 65 66 67
## [51] 68 70 71 73 75 79 80 84 87 88 89 90
##
## $Resample2
##  [1]  2  3  4  5  6  7  8  9 10 11 15 17 18 20 21 22 25 27 28 29 30 31 33 34 35
## [26] 36 38 39 40 42 43 44 45 46 47 48 49 52 56 57 58 60 66 67 68 69 70 71 72 73
## [51] 74 75 77 78 79 80 81 82 83 84 87 88
```

```
train = RailTrail[part$Resample1,]
test  = RailTrail[-part$Resample1,]
```

1.2 Cross-Validation

```r
createFolds(RailTrail$volume, k = 5)
```

```
## $Fold1
##  [1]  4 13 16 17 25 28 31 32 34 38 51 52 53 57 59 60 65 85
##
## $Fold2
##  [1]  2 12 15 21 22 27 30 39 40 49 55 66 69 80 81 84 86
##
## $Fold3
##  [1]  3  5  6  8 11 19 24 37 44 47 58 61 63 64 75 77 79 82 83 87
##
## $Fold4
##  [1]  7  9 10 18 20 35 43 45 46 48 50 54 62 68 70 71 78 88 90
##
## $Fold5
##  [1]  1 14 23 26 29 33 36 41 42 56 67 72 73 74 76 89
```

With repetitions:

```r
createMultiFolds(RailTrail$volume, k = 2, times = 3)
```

```
## $Fold1.Rep1
##  [1]  2  3  6  7  8 13 14 17 19 23 24 25 26 31 33 34 35 37 38 43 47 51 53 54 56
## [26] 58 62 63 67 68 69 71 72 73 74 75 76 80 81 82 83 84 86 89
##
## $Fold2.Rep1
##  [1]  1  4  5  9 10 11 12 15 16 18 20 21 22 27 28 29 30 32 36 39 40 41 42 44 45
## [26] 46 48 49 50 52 55 57 59 60 61 64 65 66 70 77 78 79 85 87 88 90
##
## $Fold1.Rep2
##  [1]  1  2  3 13 15 16 19 21 24 25 27 28 34 36 37 38 40 41 42 47 48 50 53 56 57
## [26] 58 59 60 63 64 65 66 68 71 72 73 74 80 82 84 85 86 87 88
##
## $Fold2.Rep2
##  [1]  4  5  6  7  8  9 10 11 12 14 17 18 20 22 23 26 29 30 31 32 33 35 39 43 44
## [26] 45 46 49 51 52 54 55 61 62 67 69 70 75 76 77 78 79 81 83 89 90
##
## $Fold1.Rep3
##  [1]  2  4  8 13 14 17 18 20 21 25 27 28 29 31 32 34 38 41 42 44 45 47 48 49 51
## [26] 53 54 56 57 58 59 60 63 64 65 68 71 72 74 75 77 82 84 90
##
## $Fold2.Rep3
##  [1]  1  3  5  6  7  9 10 11 12 15 16 19 22 23 24 26 30 33 35 36 37 39 40 43 46
## [26] 50 52 55 61 62 66 67 69 70 73 76 78 79 80 81 83 85 86 87 88 89
```

1.3 Bootstrap

```r
createResample(RailTrail$volume, times = 3)
```

```
## $Resample1
##  [1]  1  1  1  2  3  4  5  5  6  7  7  9 10 11 11 11 12 14 15 17 19 20 22 23 24
## [26] 25 26 29 30 30 31 31 32 33 33 33 35 35 36 39 39 40 40 40 41 43 45 46 47 48
## [51] 49 50 51 51 51 54 54 54 55 56 59 59 61 61 64 64 67 68 68 72 73 74 76 76 76
## [76] 76 77 78 80 81 82 83 83 83 83 86 86 87 88 90
##
## $Resample2
##  [1]  3  4  5  6  6  7  9  9 12 15 16 18 18 19 20 22 23 24 25 26 27 29 29 31 32
## [26] 34 35 36 36 41 41 41 42 42 43 44 45 45 46 46 47 47 48 48 51 52 52 53 56 57
## [51] 58 58 59 60 60 61 61 62 62 63 63 64 65 65 68 68 70 70 70 70 70 71 71 74 76
## [76] 77 78 79 80 81 81 82 82 83 85 85 86 87 88 88
##
## $Resample3
##  [1]  1  1  1  1  3  4  6  6  9  9 11 11 15 15 16 16 17 18 18 19 19 19 20 21 21
## [26] 22 25 25 25 28 29 30 30 30 31 33 34 35 36 39 40 42 42 42 42 45 47 48 48 49
## [51] 51 51 51 52 53 53 54 54 55 55 55 56 56 57 57 58 59 60 61 61 64 65 65 68 69
## [76] 72 73 74 74 75 75 75 77 77 77 82 85 86 87 87
```

2 Regression

2.1 Create train/test samples

```r
part = createDataPartition(RailTrail$volume, times = 2, p = 2/3)
train = RailTrail[part$Resample1,]
test  = RailTrail[-part$Resample1,]
```

2.2 Train

```r
model_lm = lm(volume ~ hightemp, data = train)
model_knnreg = gknn(volume ~ hightemp, data = train)
```

2.3 Predict test set data

```r
pred_lm = predict(model_lm, test)
pred_knnreg = predict(model_knnreg, test)
```

2.4 Evaluate

```r
rbind(lm  = postResample(pred_lm, test$volume),
      knn = postResample(pred_knnreg, test$volume))
```

```
##         RMSE   Rsquared       MAE
## lm  126.5919 0.18265927  93.62582
## knn 154.8042 0.06529704 117.16369
```

3 Classification

3.1 Create train/test samples

Bootstrap

```
ind   = createResample(iris$Species, times = 1)
train = iris[ind$Resample1,] ## 150 cases!
test  = iris[-ind$Resample1,] ## only those not in train set
nrow(train)
```

```
## [1] 150
```

```
nrow(test)
```

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

3.2 Train models

```
model_nb = naiveBayes(Species ~ ., data = train)
model_knn = gknn(Species ~ ., data = train)
```

3.3 Predict test set data

```
pred_nb = predict(model_nb, test)
pred_knn = predict(model_knn, test, type = "class")
```

3.4 Evaluate

```
rbind(nb  = postResample(pred_nb, test$Species),
      knn = postResample(pred_knn, test$Species))
```

```
##      Accuracy     Kappa
## nb  0.9333333 0.8984772
## knn 0.9333333 0.8984772
```

4 Performance Evaluation for Classifiers

4.1 Confusion Matrix for true/predicted values

```
confusionMatrix(pred_knn, test$Species, mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         13         1
##   virginica       0          3        21
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.838, 0.9815)
##     No Information Rate : 0.3667
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8985
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                        Class: setosa Class: versicolor Class: virginica
## Precision                     1.0000            0.9286           0.8750
## Recall                        1.0000            0.8125           0.9545
## F1                            1.0000            0.8667           0.9130
## Prevalence                    0.3667            0.2667           0.3667
## Detection Rate                0.3667            0.2167           0.3500
## Detection Prevalence          0.3667            0.2333           0.4000
## Balanced Accuracy             1.0000            0.8949           0.9378
```

```
confusionMatrix(pred_knn, test$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         13         1
##   virginica       0          3        21
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.838, 0.9815)
##     No Information Rate : 0.3667
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8985
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.8125           0.9545
## Specificity                 1.0000            0.9773           0.9211
## Pos Pred Value              1.0000            0.9286           0.8750
## Neg Pred Value              1.0000            0.9348           0.9722
## Prevalence                  0.3667            0.2667           0.3667
## Detection Rate              0.3667            0.2167           0.3500
## Detection Prevalence        0.3667            0.2333           0.4000
## Balanced Accuracy           1.0000            0.8949           0.9378
```

4.2 Confusion matrix for a given fourfold-table

```r
pred = c(T, T, F, F, T, T, F, F, T, F)
true = c(T, T, F, F, F, F, F, T, T, F)
tab = table(pred, true)
confusionMatrix(tab, positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##        true
## pred    FALSE TRUE
##   FALSE     4    1
##   TRUE      2    3
##
##                Accuracy : 0.7
##                  95% CI : (0.3475, 0.9333)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.3823
##
##                   Kappa : 0.4
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.7500
##             Specificity : 0.6667
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 0.8000
##              Prevalence : 0.4000
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7083
##
##        'Positive' Class : TRUE
##
```

5 Calibration plots for probability-based classifiers

5.1 Bank marketing data

Bank data: Response to marketing campaign for some bank product (term deposit)

```
dat = read.table("bank.csv", sep = ";", header = TRUE, stringsAsFactors = TRUE)
head(dat)
```

```
##   age          job marital education default balance housing loan  contact day
## 1  30   unemployed married   primary      no    1787      no   no cellular  19
## 2  33     services married secondary      no    4789     yes  yes cellular  11
## 3  35   management  single  tertiary      no    1350     yes   no cellular  16
## 4  30   management married  tertiary      no    1476     yes  yes  unknown   3
## 5  59  blue-collar married secondary      no       0     yes   no  unknown   5
## 6  35   management  single  tertiary      no     747      no   no cellular  23
##   month duration campaign pdays previous poutcome  y
## 1   oct       79        1    -1        0  unknown no
## 2   may      220        1   339        4  failure no
## 3   apr      185        1   330        1  failure no
## 4   jun      199        4    -1        0  unknown no
## 5   may      226        1    -1        0  unknown no
## 6   feb      141        2   176        3  failure no
```

```
summary(dat)
```

```
##       age                  job          marital         education    default
##  Min.   :19.00   management :969   divorced: 528   primary  : 678   no :4445
##  1st Qu.:33.00   blue-collar:946   married :2797   secondary:2306   yes:  76
##  Median :39.00   technician :768   single  :1196   tertiary :1350
##  Mean   :41.17   admin.     :478                   unknown  : 187
##  3rd Qu.:49.00   services   :417
##  Max.   :87.00   retired    :230
##                  (Other)    :713
##     balance       housing     loan           contact          day
##  Min.   :-3313   no :1962   no :3830   cellular :2896   Min.   : 1.00
##  1st Qu.:   69   yes:2559   yes: 691   telephone: 301   1st Qu.: 9.00
##  Median :  444                         unknown  :1324   Median :16.00
##  Mean   : 1423                                          Mean   :15.92
##  3rd Qu.: 1480                                          3rd Qu.:21.00
##  Max.   :71188                                          Max.   :31.00
##
##      month          duration       campaign          pdays
##  may    :1398   Min.   :   4   Min.   : 1.000   Min.   : -1.00
##  jul    : 706   1st Qu.: 104   1st Qu.: 1.000   1st Qu.: -1.00
##  aug    : 633   Median : 185   Median : 2.000   Median : -1.00
##  jun    : 531   Mean   : 264   Mean   : 2.794   Mean   : 39.77
##  nov    : 389   3rd Qu.: 329   3rd Qu.: 3.000   3rd Qu.: -1.00
##  apr    : 293   Max.   :3025   Max.   :50.000   Max.   :871.00
##  (Other): 571
##     previous          poutcome      y
##  Min.   : 0.0000   failure: 490   no :4000
```

```
##  1st Qu.: 0.0000   other  : 197   yes: 521
##  Median : 0.0000   success: 129
##  Mean   : 0.5426   unknown:3705
##  3rd Qu.: 0.0000
##  Max.   :25.0000
##
```

```
part = createDataPartition(dat$y, times = 1, p = 2/3)
train = dat[part$Resample1,]
test  = dat[-part$Resample1,]
```

5.2 Fit NaiveBayes-Model

```
head(train)
```

```
##   age         job marital education default balance housing loan  contact day
## 2  33    services married secondary      no    4789     yes  yes cellular  11
## 3  35  management  single  tertiary      no    1350     yes   no cellular  16
## 4  30  management married  tertiary      no    1476     yes  yes  unknown   3
## 5  59 blue-collar married secondary      no       0     yes   no  unknown   5
## 6  35  management  single  tertiary      no     747      no   no cellular  23
## 8  39  technician married secondary      no     147     yes   no cellular   6
##   month duration campaign pdays previous poutcome  y
## 2   may      220        1   339        4  failure no
## 3   apr      185        1   330        1  failure no
## 4   jun      199        4    -1        0  unknown no
## 5   may      226        1    -1        0  unknown no
## 6   feb      141        2   176        3  failure no
## 8   may      151        2    -1        0  unknown no
```

```
model = naiveBayes(y ~ ., data = train)

## Performance on test set
confusionMatrix(predict(model, test), test$y, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1175   81
##        yes  158   92
##
##                Accuracy : 0.8413
##                  95% CI : (0.8219, 0.8594)
##     No Information Rate : 0.8851
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3462
##
##  Mcnemar's Test P-Value : 8.832e-07
```

```
##
##             Sensitivity : 0.53179
##             Specificity : 0.88147
##          Pos Pred Value : 0.36800
##          Neg Pred Value : 0.93551
##              Prevalence : 0.11487
##          Detection Rate : 0.06109
##    Detection Prevalence : 0.16600
##       Balanced Accuracy : 0.70663
##
##        'Positive' Class : yes
##
```

Note: model useless since accuracy ~ NIR

5.3 Calibrate classifier using ROC

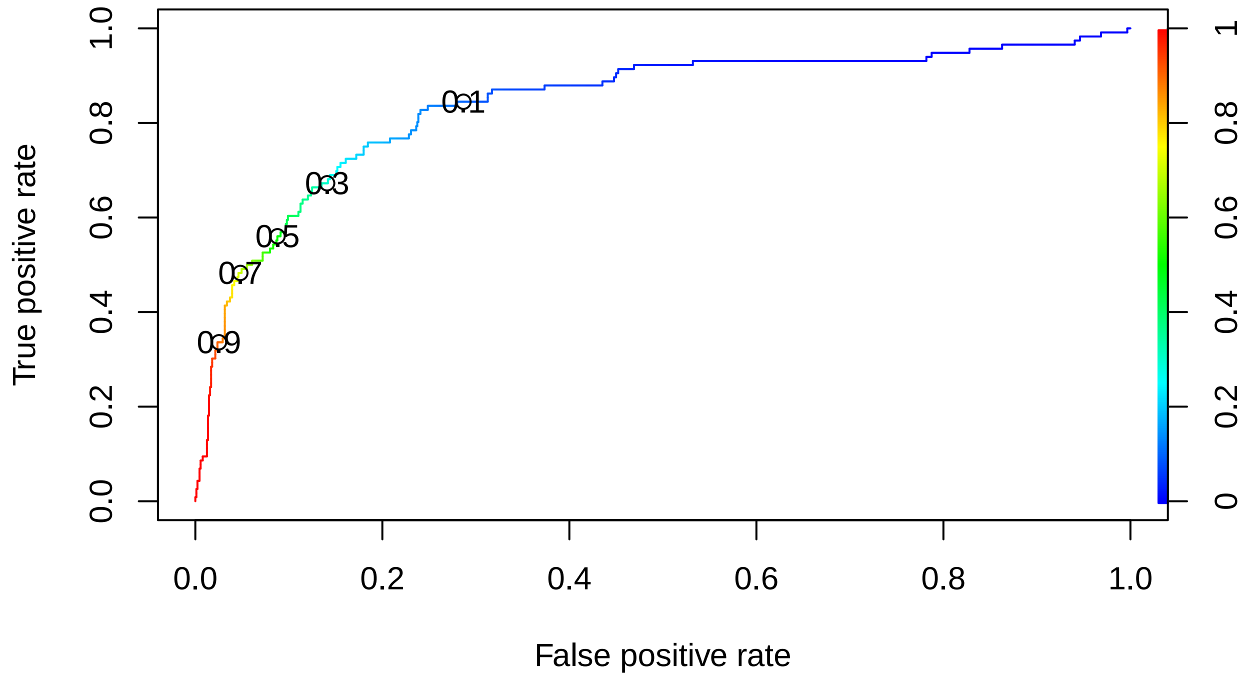
Create probabilities for predictions on *validation* set:

```r
part = createDataPartition(train$y, times = 1, p = 2/3)
train_sub = train[part$Resample1,]
validation = train[-part$Resample1,]
model = naiveBayes(y ~ ., data = train_sub)

## use "yes" column
prob = predict(model, validation, type = "raw")[,"yes"]
```

5.4 ROC-curve

```r
predobj = prediction(prob, validation$y, label.ordering = c("no", "yes"))
perf = performance(predobj, "tpr", "fpr")
plot(perf, colorize = TRUE, print.cutoffs.at = seq(0.1, 1, 0.2))
```




```r
## AUC-value:
performance(predobj, "auc")@y.values
```

```
## [[1]]
## [1] 0.841453
```

Choose Cutoff 0.2 and compute performance again on test set:

```r
pred = predict(model, test, type = "raw")[,2] > 0.2
confusionMatrix(table(pred, test$y == "yes"), positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##
## pred    FALSE TRUE
##   FALSE  1046   49
##   TRUE    287  124
##
##                Accuracy : 0.7769
##                  95% CI : (0.755, 0.7977)
##     No Information Rate : 0.8851
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3137
```

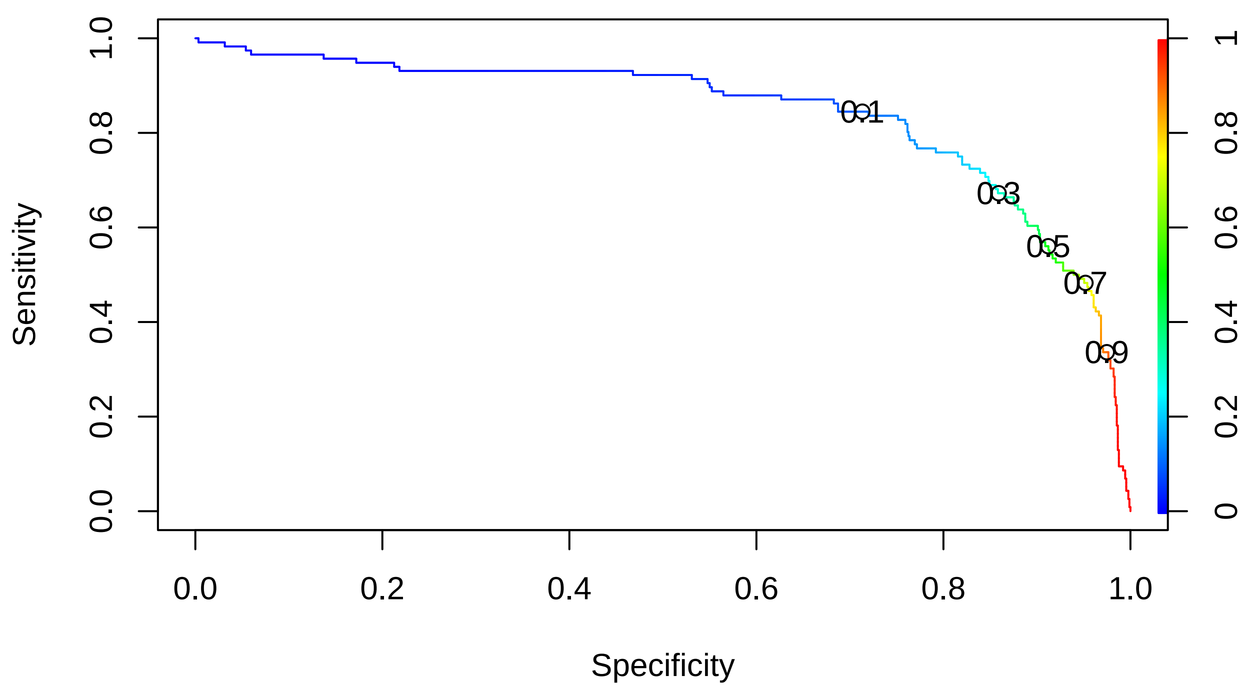
```
##
## Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.71676
##             Specificity : 0.78470
##          Pos Pred Value : 0.30170
##          Neg Pred Value : 0.95525
##              Prevalence : 0.11487
##          Detection Rate : 0.08234
##    Detection Prevalence : 0.27291
##       Balanced Accuracy : 0.75073
##
##        'Positive' Class : TRUE
##
```

Better tradeoff between sensitivity and specificity.

5.5 Sensitivity-Specificity-Curve

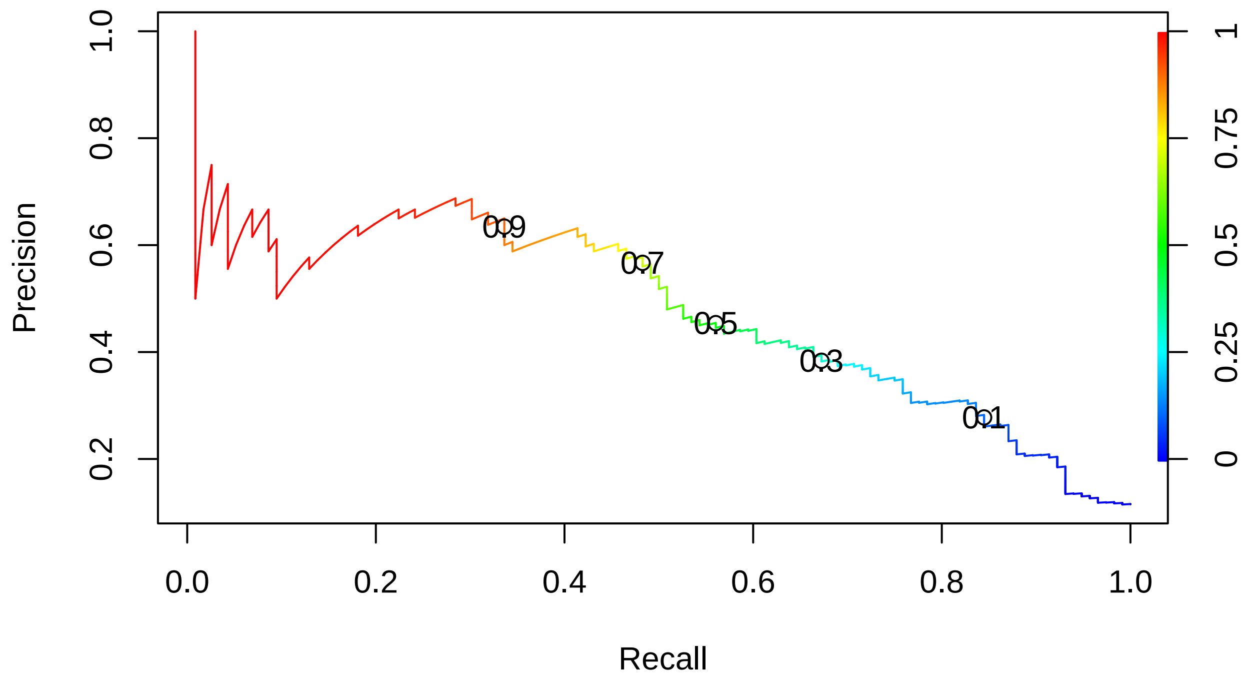
Actually, same than ROC-curve (X-axis flipped)

```r
perf = performance(predobj, "sens", "spec")
plot(perf, colorize = TRUE, print.cutoffs.at = seq(0.1, 1, 0.2))
```



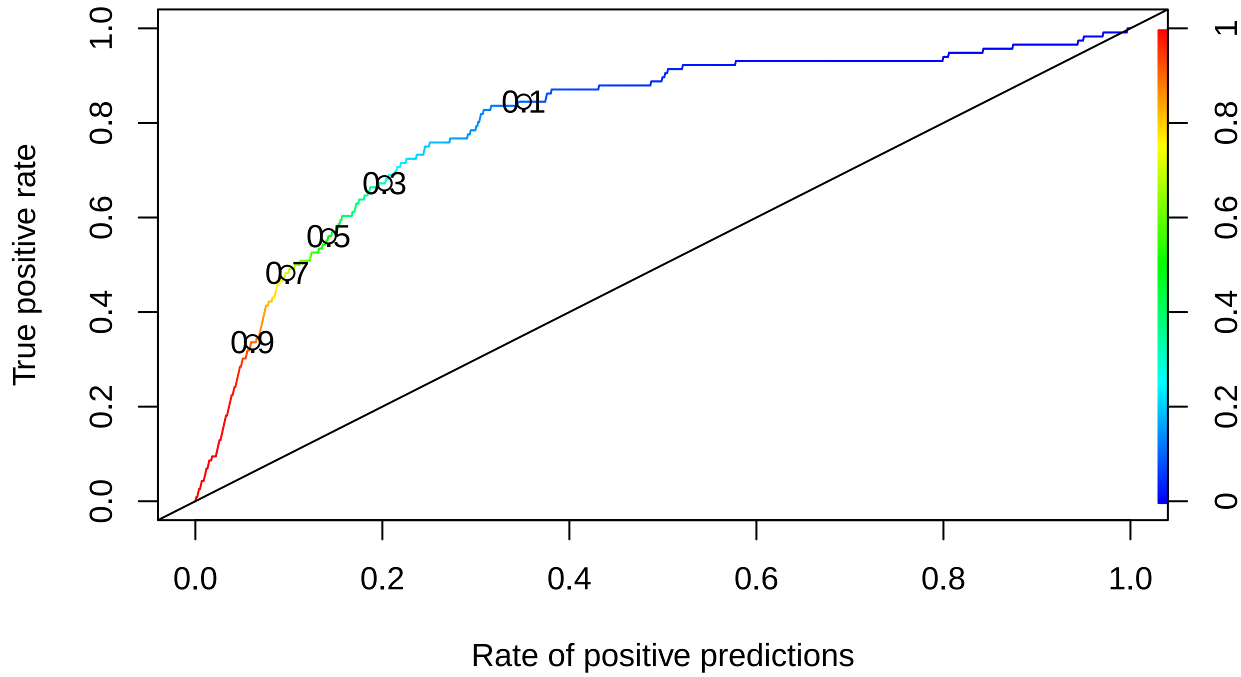
5.6 Recall-Precision-Curve

```
perf = performance(predobj, "prec", "rec")
plot(perf, colorize = TRUE, print.cutoffs.at = seq(0.1, 1, 0.2))
```



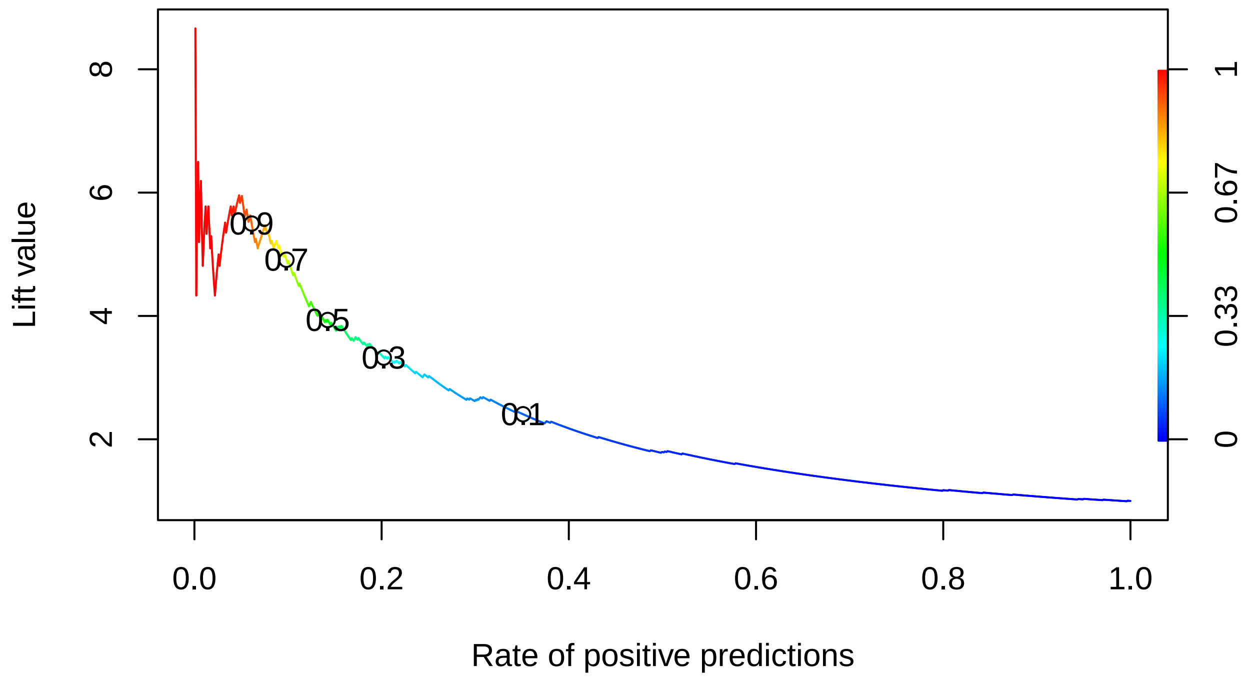
5.7 Cumulative Response Curve

```
perf = performance(predobj, "tpr", "rpp")
plot(perf, colorize = TRUE, print.cutoffs.at = seq(0.1, 1, 0.2))
abline(a = 0, b = 1)
```



5.8 Lift chart

```
perf = performance(predobj, "lift", "rpp")
plot(perf, colorize = TRUE, print.cutoffs.at = seq(0.1, 1, 0.2))
```



6 Tuning hyperparameters

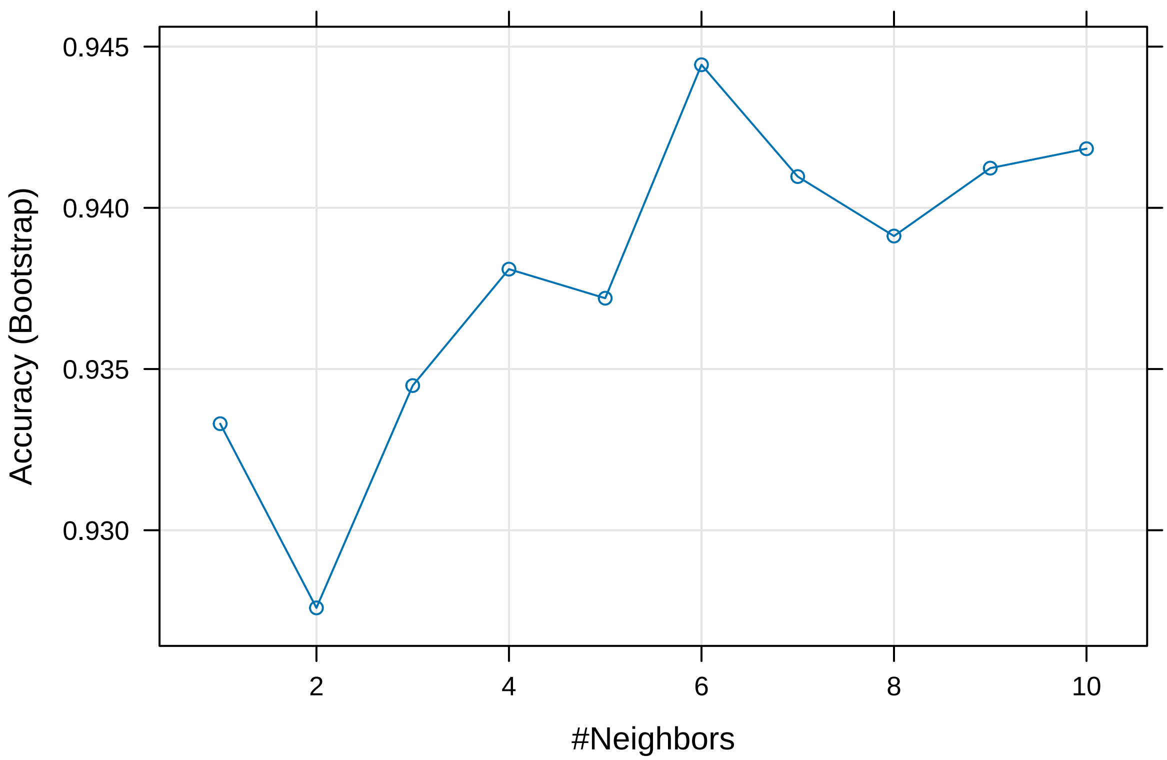
Note: In the following examples, we use the whole data set for tuning for simplicity. In real applications, create training/test sets first and perform tuning only on *training* set, using part of it as *validation* set!

```
set.seed(4711)
model_knn = train(Species ~ ., data = iris, "knn",
                  preProcess = c("scale", "center"),
                  trControl = trainControl(method = "boot"),
                  tuneGrid = data.frame(k = 1:10))
model_knn
```

```
## k-Nearest Neighbors
##
## 150 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## Pre-processing: scaled (4), centered (4)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 150, 150, 150, 150, 150, 150, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
```

```
##    1  0.9333046  0.8989173
##    2  0.9275919  0.8903665
##    3  0.9344876  0.9008575
##    4  0.9380984  0.9061775
##    5  0.9371970  0.9048937
##    6  0.9444377  0.9159108
##    7  0.9409696  0.9106566
##    8  0.9391245  0.9078966
##    9  0.9412302  0.9110822
##   10  0.9418332  0.9119600
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 6.
```

```
plot(model_knn)
```



The tuning was performed on the 25 bootstrap samples.

Average confusion matrix for best model:

```
confusionMatrix(model_knn)
```

```
## Bootstrapped (25 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
```

```
##              Reference
## Prediction   setosa versicolor virginica
##   setosa       34.4        0.0       0.0
##   versicolor    0.1       29.4       3.0
##   virginica     0.0        2.5      30.6
##
##  Accuracy (average) : 0.9447
```

Trick to get measures:

```r
tab = trunc(confusionMatrix(model_knn)$table)
confusionMatrix(tab)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         34          0         0
##   versicolor      0         29         2
##   virginica       0          2        30
##
## Overall Statistics
##
##                Accuracy : 0.9588
##                  95% CI : (0.8978, 0.9887)
##     No Information Rate : 0.3505
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9381
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9355           0.9375
## Specificity                 1.0000            0.9697           0.9692
## Pos Pred Value              1.0000            0.9355           0.9375
## Neg Pred Value              1.0000            0.9697           0.9692
## Prevalence                  0.3505            0.3196           0.3299
## Detection Rate              0.3505            0.2990           0.3093
## Detection Prevalence        0.3505            0.3196           0.3299
## Balanced Accuracy           1.0000            0.9526           0.9534
```

Use best model:

```r
predict(model_knn, iris[1:5,-5])
```

```
## [1] setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

7 Automatic sampling, training & tuning

See `?models` for a list of available models in caret.

7.1 Fitting models

```
set.seed(4711) # set seed for all models!
model_lm = train(volume ~ hightemp, method = "lm", data = RailTrail)
model_lm
```

```
## Linear Regression
##
## 90 samples
##  1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 90, 90, 90, 90, 90, 90, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   105.9944  0.3320082  79.12959
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
set.seed(4711) # use same seed for all models!
model_knn = train(volume ~ hightemp, method = "knn", data = RailTrail,
                  preProcess = c("scale", "center"),
                  tuneGrid = data.frame(k = 1:10))
model_knn
```

```
## k-Nearest Neighbors
##
## 90 samples
##  1 predictor
##
## Pre-processing: scaled (1), centered (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 90, 90, 90, 90, 90, 90, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  133.1976  0.1943992  100.10263
##    2  125.4203  0.2437184   92.65906
##    3  120.4376  0.2652129   88.75831
##    4  116.7899  0.2778544   86.00548
##    5  113.5367  0.3026275   83.61075
##    6  110.2958  0.3233051   81.02598
##    7  107.6897  0.3483983   79.06783
##    8  106.2128  0.3570732   77.90331
##    9  105.1751  0.3642035   77.37585
```

```
##   10  104.2709  0.3741905   76.84947
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 10.
```

7.2 Model comparison

7.2.1 side-by-side

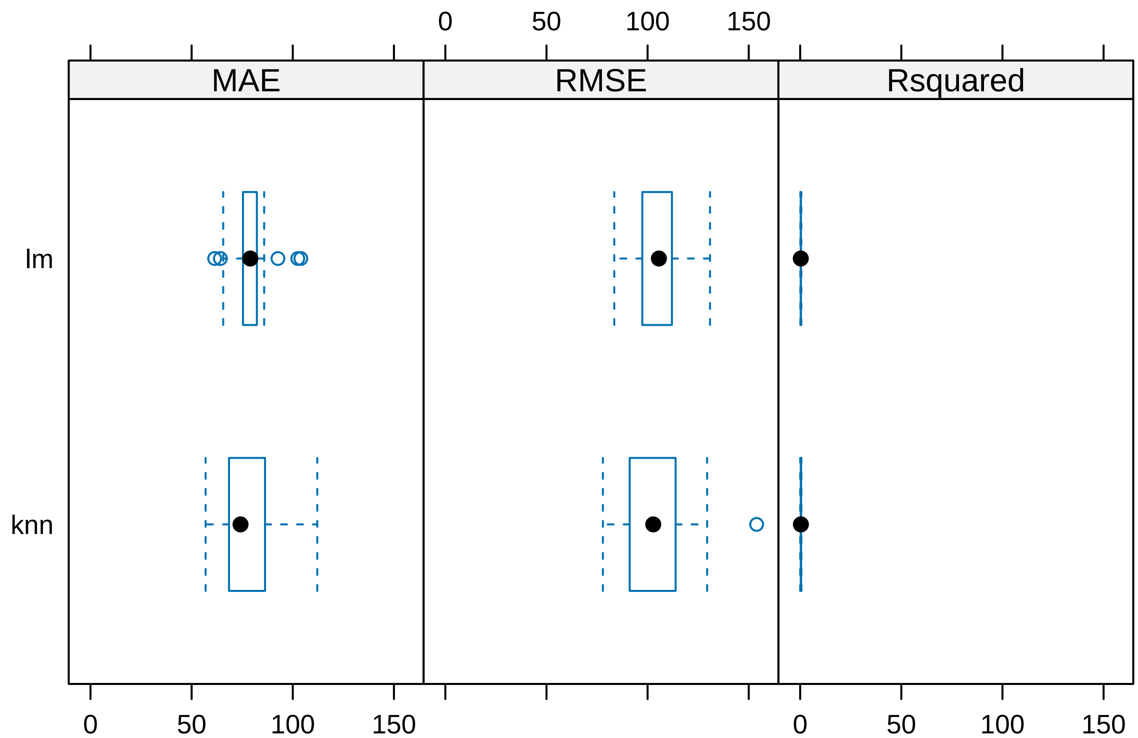
```r
res = resamples(list(knn = model_knn, lm = model_lm))
```

```r
summary(res)
```

```
##
## Call:
## summary.resamples(object = res)
##
## Models: knn, lm
## Number of resamples: 25
##
## MAE
##         Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn 56.90604 68.48944 74.17990 76.84947 86.29575 112.1055    0
## lm  61.37323 75.39249 79.01849 79.12959 82.26355 103.9788    0
##
## RMSE
##         Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn 77.90514 91.13558 102.7691 104.2709 113.8434 153.8869    0
## lm  83.50299 97.36886 105.5822 105.9944 112.0109 130.8544    0
##
## Rsquared
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn 0.01455077 0.2914759 0.3593974 0.3741905 0.4808125 0.6055482    0
## lm  0.09209436 0.2747304 0.3036938 0.3320082 0.4092167 0.5794341    0
```

Compare graphically:

```r
bwplot(res)
```



7.2.2 Pairwise:

t-test for differences:

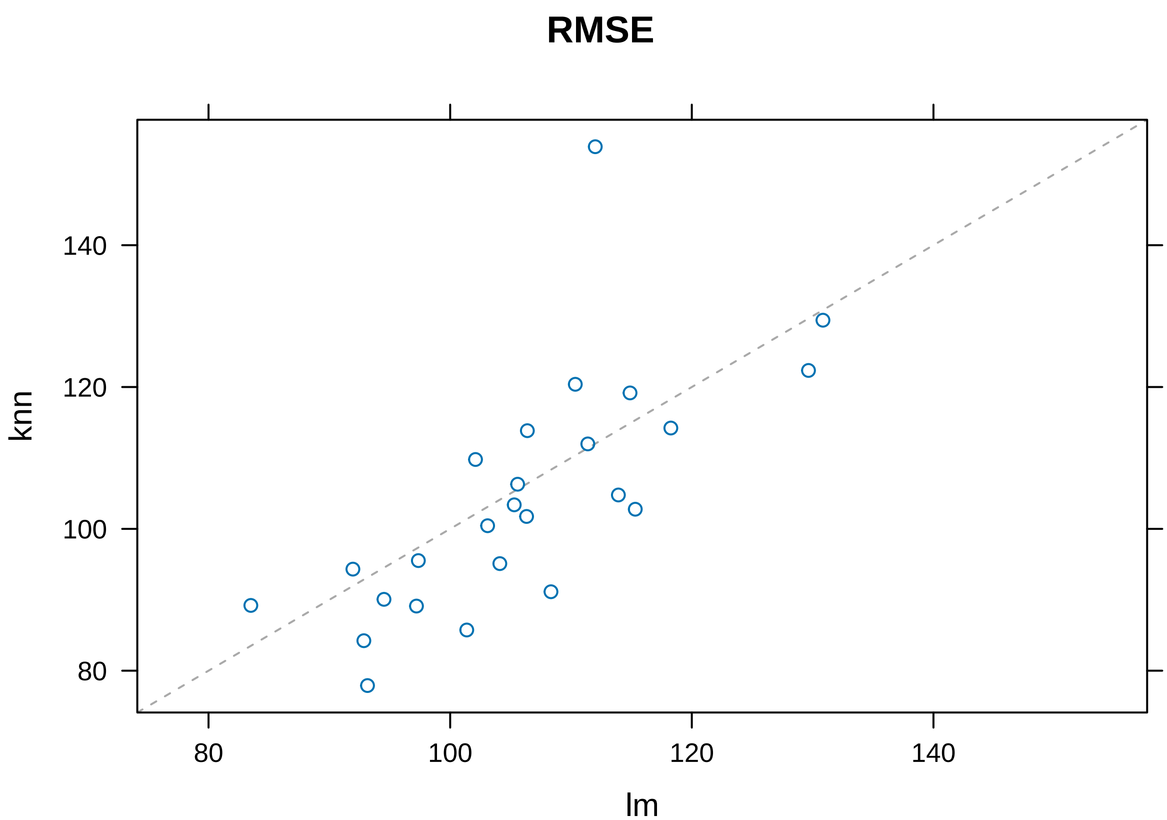
```r
summary(diff(res))
```

```
##
## Call:
## summary.diff.resamples(object = diff(res))
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## MAE
##     knn    lm
## knn        -2.28
## lm  0.2348
##
## RMSE
##     knn    lm
## knn        -1.723
## lm  0.4705
##
## Rsquared
```

```
##      knn      lm
## knn           0.04218
## lm  0.05598
```

Correlation plot:

```r
xyplot(res, metric = "RMSE")
```



RMSE will highly correlate if classifiers have similar performance.

7.3 Use on SLURM cluster

In order to use `caret` on a cluster, it suffices to create a cluster and to register it for the `doParallel` package before calling `train`:

```r
library(slurmR)
cl = makeSlurmCluster(84)

library(doParallel)
registerDoParallel(cl)
```

At the end, do not forget to clean up:

```r
stopCluster(cl)
```