

R Notebook

Parametros:

```
Measure = F1 measure
Columns = sampling, weight_space, underbagging, learner
Performance = tuning_measure
Filter keys = imba.rate
Filter values = 0.01
```

```
library("scmamp")
library(dplyr)
```

Tratamento dos dados

Carregando data set compilado

```
ds = read.csv("/home/rodrigo/Dropbox/UNICAMP/IC/estudo_cost_learning/SummaryResults/summary_compilation.csv")
ds = filter(ds, learner != "classif.rusboost")
summary(ds)
```

```
##           learner      weight_space
## classif.ksvm      :17100  Mode :logical
## classif.randomForest:17100 FALSE:41040
## classif.rusboost   :    0  TRUE :10260
## classif.xgboost    :17100  NA's :0
##
##
##
##           measure      sampling      underbagging
## Accuracy              :10260  ADASYN:10260  Mode :logical
## Area under the curve    :10260  FALSE :30780  FALSE:41040
## F1 measure              :10260  SMOTE :10260  TRUE :10260
## G-mean                 :10260              NA's :0
## Matthews correlation coefficient:10260
##
##
## tuning_measure  holdout_measure  holdout_measure_residual
## Min.      :-0.1277  Min.      :-0.2120  Min.      :-0.4658
## 1st Qu.: 0.6911  1st Qu.: 0.4001  1st Qu.: 0.1994
## Median : 0.9700  Median : 0.8571  Median : 0.5581
## Mean   : 0.7903  Mean   : 0.6718  Mean   : 0.5298
## 3rd Qu.: 0.9975  3rd Qu.: 0.9900  3rd Qu.: 0.8755
## Max.    : 1.0000  Max.    : 1.0000  Max.    : 1.0000
## NA's    :1077    NA's    :1077    NA's    :1077
## iteration_count      dataset      imba.rate
## Min.      :1         abalone      : 900  Min.      :0.0010
## 1st Qu.:1          adult         : 900  1st Qu.:0.0100
## Median :2          bank          : 900  Median :0.0300
## Mean   :2          car           : 900  Mean   :0.0286
```

```
## 3rd Qu.:3      cardiocography-10clases: 900 3rd Qu.:0.0500
## Max. :3      cardiocography-3clases : 900 Max. :0.0500
## NA's :1077 (Other) :45900
```

Filtrando pela metrica

```
ds = filter(ds, measure == params$measure)
```

Filtrando o data set

```
if(params$filter_keys != 'NULL' && !is.null(params$filter_keys)){
  dots = paste0(params$filter_keys, " == '",params$filter_values, "'")
  ds = filter_(ds, .dots = dots)
}
```

```
summary(ds)
```

```
##          learner      weight_space
## classif.ksvm      :600 Mode :logical
## classif.randomForest:600 FALSE:1440
## classif.rusboost   : 0 TRUE :360
## classif.xgboost    :600 NA's :0
##
##
##
##          measure      sampling      underbagging
## Accuracy          : 0 ADASYN: 360 Mode :logical
## Area under the curve : 0 FALSE :1080 FALSE:1440
## F1 measure          :1800 SMOTE : 360 TRUE :360
## G-mean              : 0 NA's :0
## Matthews correlation coefficient: 0
##
##
## tuning_measure  holdout_measure  holdout_measure_residual
## Min. :0.0000 Min. :0.0000 Min. :0.00000
## 1st Qu.:0.1475 1st Qu.:0.0000 1st Qu.:0.02254
## Median :0.8030 Median :0.3333 Median :0.20700
## Mean :0.6194 Mean :0.4107 Mean :0.32309
## 3rd Qu.:0.9986 3rd Qu.:0.8000 3rd Qu.:0.58363
## Max. :1.0000 Max. :1.0000 Max. :1.00000
## NA's :54 NA's :54 NA's :54
## iteration_count      dataset      imba.rate
## Min. :1 abalone : 45 Min. :0.01
## 1st Qu.:1 adult : 45 1st Qu.:0.01
## Median :2 bank : 45 Median :0.01
## Mean :2 car : 45 Mean :0.01
## 3rd Qu.:3 cardiocography-10clases: 45 3rd Qu.:0.01
## Max. :3 cardiocography-3clases : 45 Max. :0.01
## NA's :54 (Other) :1530
```

Computando as médias das iteracoes

```
ds = group_by(ds, learner , weight_space , measure , sampling , underbagging , dataset , imba.rate)
ds = summarise(ds, tuning_measure = mean(tuning_measure), holdout_measure = mean(holdout_measure),
               holdout_measure_residual = mean(holdout_measure_residual))

ds = as.data.frame(ds)
```

Criando dataframe

```
# Dividindo o ds em n, um para cada técnica
splited_df = ds %>% group_by_at(.vars = params$columns) %>% do(vals = as.data.frame(.)) %>% select(vals)

# Juntando cada uma das partes horizontalmente em um data set
df_tec_wide = do.call("cbind", splited_df)

# Renomeando duplicacao de nomes
colnames(df_tec_wide) = make.unique(colnames(df_tec_wide))

# Selecionando apenas as medidas da performance escolhida
df_tec_wide_residual = select(df_tec_wide, matches(paste("^", params$performance, "$|", params$performance)))

# Renomeando colunas
new_names = NULL
for(i in (1:length(splited_df))){
  id = toString(sapply(splited_df[[i]][1, params$columns], as.character))
  new_names = c(new_names, id)
}
colnames(df_tec_wide_residual) = new_names

# Verificando a dimensao do df
dim(df_tec_wide_residual)
```

```
## [1] 40 15
```

```
# Renomeando a variavel
df = df_tec_wide_residual

head(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## 1 0.9855714
## 2 0.9920578
## 3 0.9988447
## 4 1.0000000
## 5 1.0000000
## 6 0.9986680
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9819986
## 2 NA
## 3 0.9956934
## 4 1.0000000
## 5 0.9973732
## 6 0.9992243
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9864756 0.00000000
## 2 0.9941432 0.03745141
## 3 0.9932733 0.00000000
## 4 1.0000000 0.82222222
## 5 0.9974878 0.50176367
## 6 0.9993358 0.61022928
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.0000000
## 2 0.3147376
```

```

## 3          0.0000000
## 4          1.0000000
## 5          0.4629630
## 6          0.8841190
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.00000000          0.02691361
## 2          0.31226618          0.06037610
## 3          0.02020202          0.03542331
## 4          0.98989899          0.82892416
## 5          0.59497354          0.34858986
## 6          0.84118042          0.85101010
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.02829182
## 2          0.07459643
## 3          0.08788294
## 4          0.41177845
## 5          0.15178778
## 6          0.60674901
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.03101175          0.00000000
## 2          0.07794380          0.04852524
## 3          0.07613248          0.00000000
## 4          0.43310820          0.82222222
## 5          0.11965033          0.50176367
## 6          0.37664313          0.61022928
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.00000000
## 2          NA
## 3          0.02020202
## 4          1.00000000
## 5          0.45634921
## 6          0.89955107
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.00000000          0.9847324
## 2          0.30327271          0.9930476
## 3          0.02020202          0.9986867
## 4          0.98989899          1.0000000
## 5          0.58783069          1.0000000
## 6          0.84118042          0.9986613
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9822771
## 2          0.9921117
## 3          0.9949524
## 4          1.0000000
## 5          0.9983175
## 6          0.9996681
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9872957
## 2          0.9947465
## 3          0.9942732
## 4          0.9997361
## 5          0.9977955
## 6          0.9994470

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.9262
## 1st Qu.:0.9958
## Median :0.9988
## Mean :0.9953
## 3rd Qu.:0.9997
## Max. :1.0000
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9820
## 1st Qu.:0.9964
## Median :0.9983
## Mean :0.9970
## 3rd Qu.:0.9994
## Max. :1.0000
## NA's :9
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.6463 Min. :0.0000
## 1st Qu.:0.9940 1st Qu.:0.0000
## Median :0.9981 Median :0.2007
## Mean :0.9878 Mean :0.3118
## 3rd Qu.:0.9994 3rd Qu.:0.5289
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.00000
## 1st Qu.:0.03704
## Median :0.46296
## Mean :0.46524
## 3rd Qu.:0.81561
## Max. :1.00000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.0000 Min. :0.02691
## 1st Qu.:0.1184 1st Qu.:0.06207
## Median :0.5549 Median :0.29549
## Mean :0.5140 Mean :0.35232
## 3rd Qu.:0.8489 3rd Qu.:0.62406
## Max. :1.0000 Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.02829
## 1st Qu.:0.08315
## Median :0.17611
## Mean :0.27851
## 3rd Qu.:0.37452
## Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.02172 Min. :0.0000
## 1st Qu.:0.07767 1st Qu.:0.0000
## Median :0.14925 Median :0.2007
```

```
## Mean :0.23227 Mean :0.3140
## 3rd Qu.:0.27661 3rd Qu.:0.5289
## Max. :1.00000 Max. :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.0419
## Median :0.4535
## Mean :0.4725
## 3rd Qu.:0.8742
## Max. :1.0000
## NA's :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.9262
## 1st Qu.:0.1265 1st Qu.:0.9979
## Median :0.5195 Median :0.9990
## Mean :0.5108 Mean :0.9959
## 3rd Qu.:0.8573 3rd Qu.:1.0000
## Max. :1.0000 Max. :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min. :0.9823
## 1st Qu.:0.9969
## Median :0.9993
## Mean :0.9974
## 3rd Qu.:0.9999
## Max. :1.0000
## NA's :4
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.9851
## 1st Qu.:0.9951
## Median :0.9982
## Mean :0.9967
## 3rd Qu.:0.9995
## Max. :1.0000
##
```

Verificando a média de cada coluna selecionada

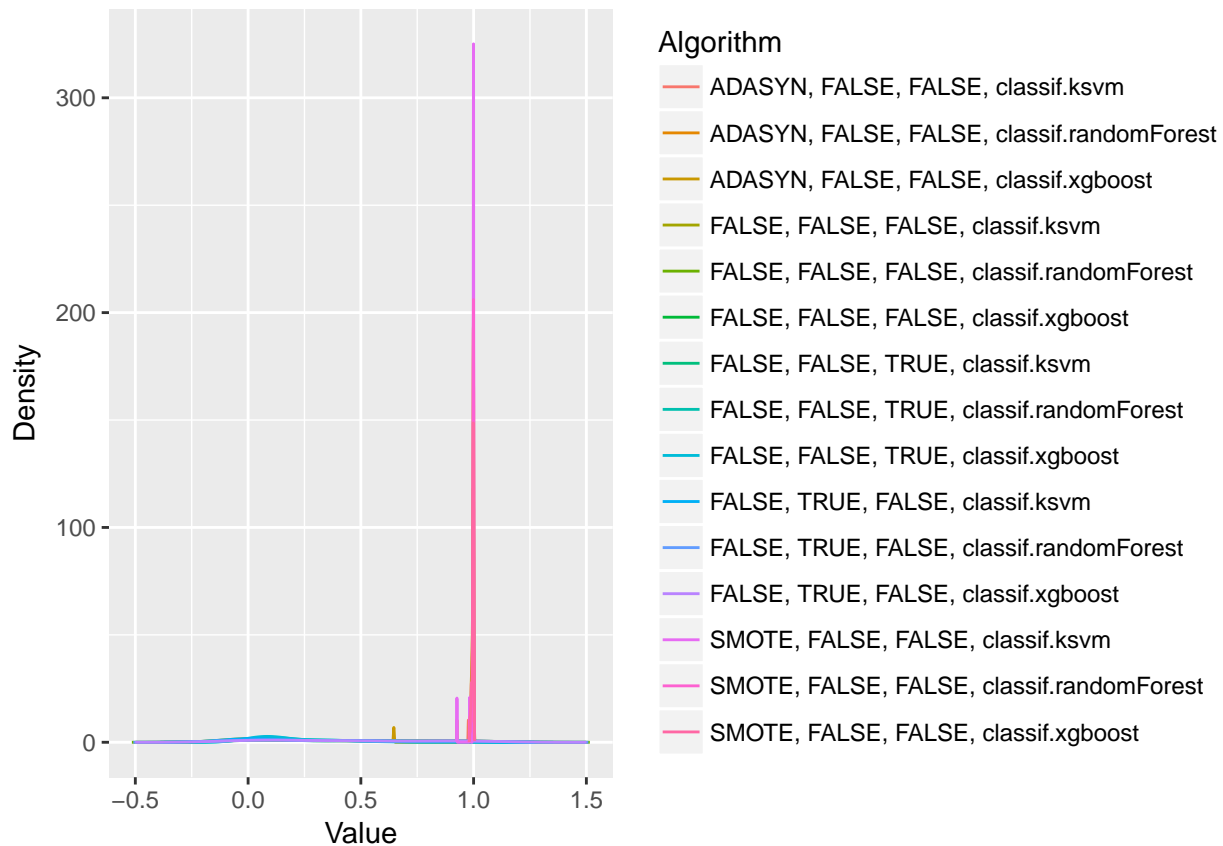
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.995330429724085"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.996999310257464"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.98782122586447"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.311775261644857"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.465235725308096"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.513998364516779"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.352315136446539"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.278511888042848"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.232274034990772"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.313988670601455"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.472481576634501"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.510842978406284"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.995896011179904"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.997357014308556"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.996656410384335"
```

Fazendo teste de normalidade

```
plotDensities(data = na.omit(df))
```



Testando as diferencas

```
friedmanTest(df)
```

```
##
## Friedman's rank sum test
##
## data: df
## Friedman's chi-squared = 308.74, df = 14, p-value < 2.2e-16
```

Testando as diferencas par a par

```
test <- nemenyiTest (df, alpha=0.05)
abs(test$diff.matrix) > test$statistic
```

```
##      ADASYN, FALSE, FALSE, classif.ksvm
## [1,]                                     FALSE
## [2,]                                     FALSE
## [3,]                                     FALSE
## [4,]                                     TRUE
## [5,]                                     TRUE
## [6,]                                     TRUE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [13,]                                    FALSE
## [14,]                                    FALSE
## [15,]                                    FALSE
##      ADASYN, FALSE, FALSE, classif.randomForest
## [1,]                                     FALSE
## [2,]                                     FALSE
## [3,]                                     FALSE
## [4,]                                     TRUE
## [5,]                                     FALSE
## [6,]                                     FALSE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    FALSE
## [13,]                                    FALSE
## [14,]                                    FALSE
## [15,]                                    FALSE
##      ADASYN, FALSE, FALSE, classif.xgboost
## [1,]                                     FALSE
## [2,]                                     FALSE
## [3,]                                     FALSE
## [4,]                                     TRUE
## [5,]                                     TRUE
## [6,]                                     TRUE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [13,]                                    FALSE
## [14,]                                    FALSE
## [15,]                                    FALSE
```



```

##      FALSE, FALSE, FALSE, classif.ksvm
## [1,]      TRUE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE
## [6,]      TRUE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]     TRUE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, FALSE, classif.randomForest
## [1,]      TRUE
## [2,]     FALSE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE
## [6,]     FALSE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, FALSE, classif.xgboost
## [1,]      TRUE
## [2,]     FALSE
## [3,]      TRUE
## [4,]      TRUE
## [5,]     FALSE
## [6,]     FALSE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]     TRUE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, TRUE, classif.ksvm
## [1,]      TRUE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE

```

```

## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] FALSE
## [11,] FALSE
## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## FALSE, FALSE, TRUE, classif.randomForest
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
## [4,] FALSE
## [5,] FALSE
## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] FALSE
## [11,] FALSE
## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## [1,] TRUE TRUE
## [2,] TRUE TRUE
## [3,] TRUE TRUE
## [4,] FALSE FALSE
## [5,] FALSE FALSE
## [6,] FALSE TRUE
## [7,] FALSE FALSE
## [8,] FALSE FALSE
## [9,] FALSE FALSE
## [10,] FALSE FALSE
## [11,] FALSE FALSE
## [12,] FALSE TRUE
## [13,] TRUE TRUE
## [14,] TRUE TRUE
## [15,] TRUE TRUE
## FALSE, TRUE, FALSE, classif.randomForest
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
## [4,] FALSE
## [5,] FALSE
## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] FALSE
## [11,] FALSE

```

```

## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## FALSE, TRUE, FALSE, classif.xgboost
## [1,] TRUE
## [2,] FALSE
## [3,] TRUE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] FALSE
## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## SMOTE, FALSE, FALSE, classif.ksvm
## [1,] FALSE
## [2,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [13,] FALSE
## [14,] FALSE
## [15,] FALSE
## SMOTE, FALSE, FALSE, classif.randomForest
## [1,] FALSE
## [2,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [13,] FALSE
## [14,] FALSE
## [15,] FALSE
## SMOTE, FALSE, FALSE, classif.xgboost
## [1,] FALSE

```

```
## [2,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [13,] FALSE
## [14,] FALSE
## [15,] FALSE
```

Plotando os ranks

```
print(colMeans(rankMatrix(df)))
```

```
##          ADASYN, FALSE, FALSE, classif.ksvm
##                               3.7625
## ADASYN, FALSE, FALSE, classif.randomForest
##                               6.3375
##          ADASYN, FALSE, FALSE, classif.xgboost
##                               4.3250
##          FALSE, FALSE, FALSE, classif.ksvm
##                               12.0375
## FALSE, FALSE, FALSE, classif.randomForest
##                               9.5875
##          FALSE, FALSE, FALSE, classif.xgboost
##                               8.4500
##          FALSE, FALSE, TRUE, classif.ksvm
##                               10.3500
## FALSE, FALSE, TRUE, classif.randomForest
##                               11.2250
##          FALSE, FALSE, TRUE, classif.xgboost
##                               11.6125
##          FALSE, TRUE, FALSE, classif.ksvm
##                               11.9750
## FALSE, TRUE, FALSE, classif.randomForest
##                               10.1000
##          FALSE, TRUE, FALSE, classif.xgboost
##                               8.5125
##          SMOTE, FALSE, FALSE, classif.ksvm
##                               2.9500
## SMOTE, FALSE, FALSE, classif.randomForest
##                               4.5750
##          SMOTE, FALSE, FALSE, classif.xgboost
##                               4.2000
```

```
result = tryCatch({
  plotCD(df, alpha=0.05, cex = 0.35)
}, error = function(e) {})
```

