

# R Notebook

## Parametros:

Measure = Area under the curve  
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.  
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 :1800 FALSE :1080 FALSE:1440
## F1 measure          : 0 SMOTE : 360 TRUE :360
## G-mean              : 0 NA's :0
## Matthews correlation coefficient: 0
##
##
## tuning_measure  holdout_measure  holdout_measure_residual
## Min. :0.3866 Min. :0.2139 Min. :0.3092
## 1st Qu.:0.9529 1st Qu.:0.8909 1st Qu.:0.7392
## Median :0.9993 Median :0.9916 Median :0.9067
## Mean :0.9498 Mean :0.9120 Mean :0.8469
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:0.9842
## Max. :1.0000 Max. :1.0000 Max. :1.0000
## NA's :51 NA's :51 NA's :51
## 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 :51 (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 0.9965924
## 2 NA
## 3 0.9999865
## 4 1.0000000
## 5 1.0000000
## 6 1.0000000
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9986175
## 2 0.9995864
## 3 0.9999160
## 4 1.0000000
## 5 0.9999556
## 6 1.0000000
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9988635 0.5777335
## 2 0.9995594 0.6022761
## 3 0.9998386 0.7565083
## 4 1.0000000 1.0000000
## 5 0.9999904 0.9757015
## 6 0.9993349 0.9996823
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.5577409
## 2 0.8632699
```

```

## 3          0.8534540
## 4          1.0000000
## 5          0.9877876
## 6          0.9999631
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.6580300          0.6095754
## 2          0.9166584          0.8041582
## 3          0.8535358          0.7367271
## 4          1.0000000          0.9634617
## 5          0.9852483          0.8600168
## 6          0.9997785          0.9310388
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.6527221
## 2          0.8879571
## 3          0.8806780
## 4          1.0000000
## 5          0.9750210
## 6          0.9992987
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.6450893          0.5776046
## 2          0.8979356          0.6355048
## 3          0.8636885          0.7565083
## 4          0.9997042          1.0000000
## 5          0.9658880          0.9757015
## 6          0.9983724          0.9996823
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.5350271
## 2          NA
## 3          0.8502324
## 4          1.0000000
## 5          0.9890923
## 6          0.9999188
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.6601316          0.9977386
## 2          0.9156899          0.9974304
## 3          0.8519776          0.9999889
## 4          1.0000000          1.0000000
## 5          0.9895202          1.0000000
## 6          0.9996198          1.0000000
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9984723
## 2          NA
## 3          0.9999023
## 4          1.0000000
## 5          0.9999751
## 6          1.0000000
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9987191
## 2          0.9995941
## 3          0.9998625
## 4          1.0000000
## 5          0.9999510
## 6          0.9997769

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.9966
## 1st Qu.:1.0000
## Median :1.0000
## Mean :0.9999
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :3
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9986
## 1st Qu.:0.9999
## Median :1.0000
## Mean :0.9999
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.9988 Min. :0.5777
## 1st Qu.:0.9998 1st Qu.:0.7526
## Median :1.0000 Median :0.9372
## Mean :0.9998 Mean :0.8739
## 3rd Qu.:1.0000 3rd Qu.:0.9983
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.5577
## 1st Qu.:0.8992
## Median :0.9894
## Mean :0.9428
## 3rd Qu.:0.9996
## Max. :1.0000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.6580 Min. :0.6096
## 1st Qu.:0.9171 1st Qu.:0.8032
## Median :0.9873 Median :0.8804
## Mean :0.9449 Mean :0.8782
## 3rd Qu.:0.9987 3rd Qu.:0.9516
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.5873
## 1st Qu.:0.9052
## Median :0.9899
## Mean :0.9401
## 3rd Qu.:0.9992
## Max. :1.0000
## NA's :1
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.4862 Min. :0.5776
## 1st Qu.:0.9041 1st Qu.:0.7528
## Median :0.9708 Median :0.9372
```

```
## Mean      :0.9250                      Mean      :0.8732
## 3rd Qu.:0.9980                      3rd Qu.:0.9983
## Max.      :1.0000                      Max.      :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.5350
## 1st Qu.:0.9081
## Median :0.9905
## Mean      :0.9419
## 3rd Qu.:0.9998
## Max.      :1.0000
## NA's      :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.6601                      Min.      :0.9974
## 1st Qu.:0.9163                      1st Qu.:1.0000
## Median :0.9899                      Median :1.0000
## Mean      :0.9435                      Mean      :0.9998
## 3rd Qu.:0.9987                      3rd Qu.:1.0000
## Max.      :1.0000                      Max.      :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.9985
## 1st Qu.:0.9999
## Median :1.0000
## Mean      :0.9999
## 3rd Qu.:1.0000
## Max.      :1.0000
## NA's      :4
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.9987
## 1st Qu.:0.9998
## Median :1.0000
## Mean      :0.9998
## 3rd Qu.:1.0000
## Max.      :1.0000
##
```

## Verificando a média de cada coluna selecionada

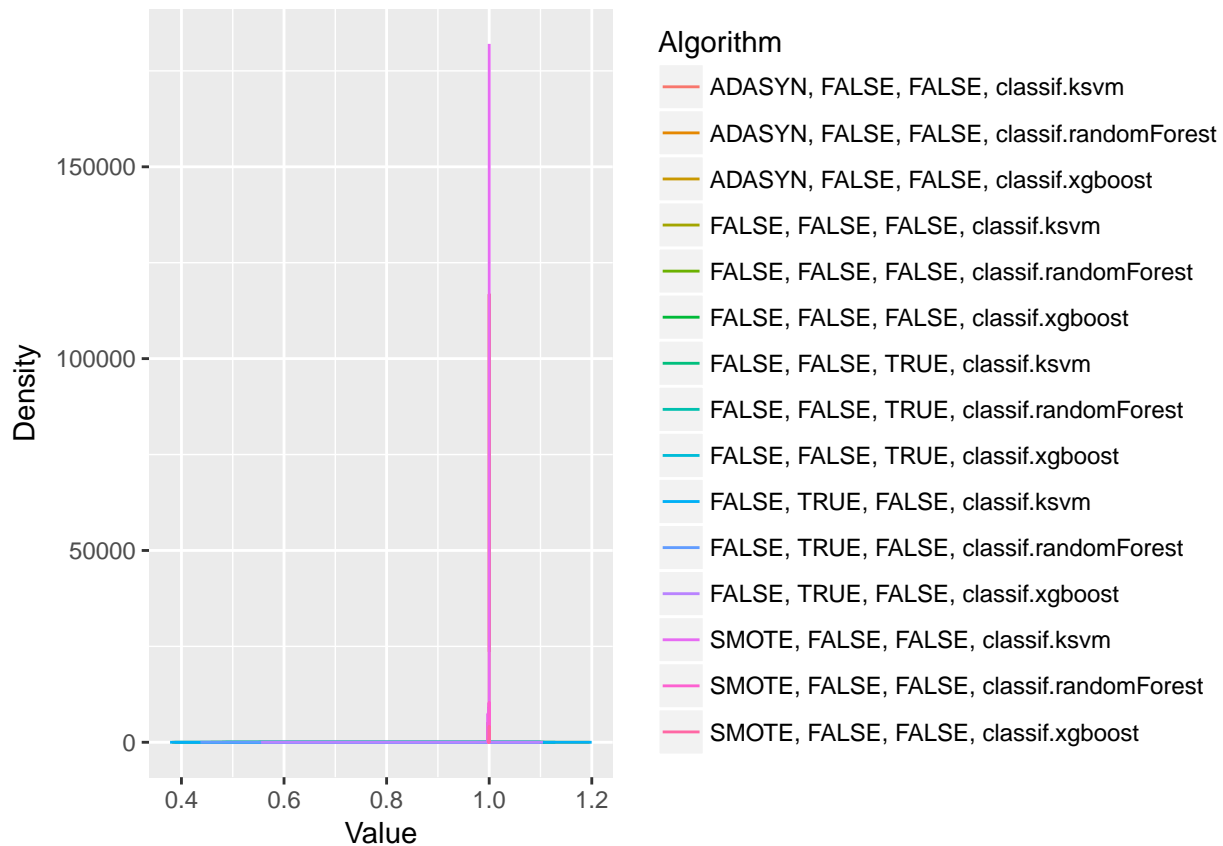
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.999851496094111"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.999858146333752"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.999810136630255"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.873924077627644"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.942821520178951"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.944902952710432"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.878197002995392"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.940111709276027"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.925004512485456"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.873175344440451"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.941895948240033"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.943479700726438"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.999806588603769"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.999878774348496"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.999825248239087"
```

## Fazendo teste de normalidade

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



## Testando as diferenças

```
friedmanTest(df)
```

```
##
## Friedman's rank sum test
##
## data: df
## Friedman's chi-squared = 283, 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,]                                     TRUE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [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,]     FALSE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, FALSE, classif.randomForest
## [1,]      TRUE
## [2,]     FALSE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE
## [6,]     FALSE
## [7,]      TRUE
## [8,]     FALSE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, FALSE, classif.xgboost
## [1,]      TRUE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE
## [6,]     FALSE
## [7,]      TRUE
## [8,]     FALSE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, TRUE, classif.ksvm
## [1,]      TRUE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]      TRUE

```

```

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

```

```

## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## FALSE, TRUE, FALSE, classif.xgboost
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
## [4,] FALSE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] FALSE
## [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
##      3.8500
## ADASYN, FALSE, FALSE, classif.randomForest
##      5.5125
##      ADASYN, FALSE, FALSE, classif.xgboost
##      5.0250
##      FALSE, FALSE, FALSE, classif.ksvm
##      11.0750
## FALSE, FALSE, FALSE, classif.randomForest
##      8.6625
##      FALSE, FALSE, FALSE, classif.xgboost
##      9.0875
##      FALSE, FALSE, TRUE, classif.ksvm
##      13.3750
## FALSE, FALSE, TRUE, classif.randomForest
##      9.9125
##      FALSE, FALSE, TRUE, classif.xgboost
##      10.9125
##      FALSE, TRUE, FALSE, classif.ksvm
##      11.2250
## FALSE, TRUE, FALSE, classif.randomForest
##      9.0875
##      FALSE, TRUE, FALSE, classif.xgboost
##      9.3375
##      SMOTE, FALSE, FALSE, classif.ksvm
##      2.9125
## SMOTE, FALSE, FALSE, classif.randomForest
##      5.1250
##      SMOTE, FALSE, FALSE, classif.xgboost
##      4.9000
```

## Plotando grafico de Critical Difference

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

