

# R Notebook

## Parametros:

Measure = Accuracy  
Columns = sampling, weight\_space, underbagging, learner  
Performance = holdout\_measure\_residual  
Filter keys = imba.rate  
Filter values = 0.001

```
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          :1800  ADASYN: 360  Mode :logical
## Area under the curve : 0  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. :0.1269  Min. :0.01517  Min. :0.03881
## 1st Qu.:0.9898 1st Qu.:0.98750 1st Qu.:0.38633
## Median :0.9938 Median :0.99163 Median :0.76435
## Mean :0.9692 Mean :0.96680 Mean :0.67114
## 3rd Qu.:0.9990 3rd Qu.:0.99687 3rd Qu.:0.95470
## Max. :1.0000 Max. :1.00000 Max. :1.00000
## NA's :48 NA's :48 NA's :48
## iteration_count      dataset      imba.rate
## Min. :1      abalone      : 45  Min. :0.001
## 1st Qu.:1      adult      : 45  1st Qu.:0.001
## Median :2      bank      : 45  Median :0.001
## Mean :2      car      : 45  Mean :0.001
## 3rd Qu.:3      cardiocography-10clases: 45 3rd Qu.:0.001
## Max. :3      cardiocography-3clases : 45 Max. :0.001
## NA's :48      (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 columnas
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 0.3572658
## 2 0.4186973
## 3 0.6295736
## 4 0.8769575
## 5 0.8764259
## 6 0.7399449
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.3476240
## 2 NA
## 3 0.6320969
## 4 0.7859806
## 5 0.8783270
## 6 0.7955923
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.3374656 0.3329890
## 2 0.5265823 0.4313027
## 3 0.6419379 0.6298259
## 4 0.7404922 0.8337062
## 5 0.9005070 0.8852978
## 6 0.8077135 0.7471074
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.3286846
## 2 NA
```

```

## 3          0.6298259
## 4          0.8337062
## 5          0.8852978
## 6          0.7922865
##  FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.3286846          0.6153581
## 2          0.5114979          0.5874736
## 3          0.6303306          0.6235175
## 4          0.7613721          0.9418345
## 5          0.8846641          0.8776933
## 6          0.8082645          0.7432507
##  FALSE, FALSE, TRUE, classif.randomForest
## 1          0.6559917
## 2          0.8212025
## 3          0.7845067
## 4          0.7427293
## 5          0.9176172
## 6          0.8584022
##  FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.6659780          0.3297176
## 2          0.8204114          0.3964926
## 3          0.7617966          0.6298259
## 4          0.7166294          0.8337062
## 5          0.9163498          0.8852978
## 6          0.8347107          0.7471074
##  FALSE, TRUE, FALSE, classif.randomForest
## 1          0.3286846
## 2          0.5251319
## 3          0.6298259
## 4          0.7740492
## 5          0.8871990
## 6          0.7911846
##  FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.3286846          0.3553719
## 2          0.5170095          0.4211234
## 3          0.6303306          0.6298259
## 4          0.7651007          0.8724832
## 5          0.8840304          0.8770596
## 6          0.8077135          0.7377410
##  SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.3507231
## 2          0.4767405
## 3          0.6361342
## 4          0.8187919
## 5          0.8776933
## 6          0.7939394
##  SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.3376377
## 2          0.5168249
## 3          0.6394146
## 4          0.7442207
## 5          0.8986058
## 6          0.8126722

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.03881
## 1st Qu.:0.33192
## Median :0.68170
## Mean :0.63391
## 3rd Qu.:0.94624
## Max. :0.99989
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.04065
## 1st Qu.:0.42561
## Median :0.79079
## Mean :0.69405
## 3rd Qu.:0.98361
## Max. :0.99986
## NA's :8
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.04525 Min. :0.03881
## 1st Qu.:0.51248 1st Qu.:0.31355
## Median :0.78380 Median :0.71294
## Mean :0.70389 Mean :0.62944
## 3rd Qu.:0.97917 3rd Qu.:0.94182
## Max. :0.99986 Max. :0.99991
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.06542
## 1st Qu.:0.32528
## Median :0.74077
## Mean :0.64753
## 3rd Qu.:0.97617
## Max. :0.99987
## NA's :2
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.04973 Min. :0.04134
## 1st Qu.:0.37904 1st Qu.:0.43701
## Median :0.74121 Median :0.65425
## Mean :0.65896 Mean :0.63780
## 3rd Qu.:0.97695 3rd Qu.:0.88591
## Max. :0.99985 Max. :0.99499
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.2343
## 1st Qu.:0.6994
## Median :0.8547
## Mean :0.7760
## 3rd Qu.:0.9381
## Max. :0.9998
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.2272 Min. :0.03881
## 1st Qu.:0.6912 1st Qu.:0.31355
## Median :0.8397 Median :0.71294
```

```
## Mean      :0.7737                      Mean      :0.62880
## 3rd Qu.   :0.9344                      3rd Qu.   :0.94182
## Max.      :0.9998                      Max.      :0.99991
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.06468
## 1st Qu.   :0.32630
## Median    :0.70976
## Mean      :0.63467
## 3rd Qu.   :0.96623
## Max.      :0.99985
## NA's      :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.05697                      Min.      :0.03881
## 1st Qu.   :0.37442                      1st Qu.   :0.31836
## Median    :0.74837                      Median    :0.67752
## Mean      :0.65848                      Mean      :0.60995
## 3rd Qu.   :0.97751                      3rd Qu.   :0.93342
## Max.      :0.99985                      Max.      :0.99991
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.04019
## 1st Qu.   :0.40451
## Median    :0.76833
## Mean      :0.67437
## 3rd Qu.   :0.97593
## Max.      :0.99992
## NA's      :2
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.04582
## 1st Qu.   :0.50530
## Median    :0.78663
## Mean      :0.70532
## 3rd Qu.   :0.97503
## Max.      :0.99986
##
```

## Verificando a média de cada coluna selecionada

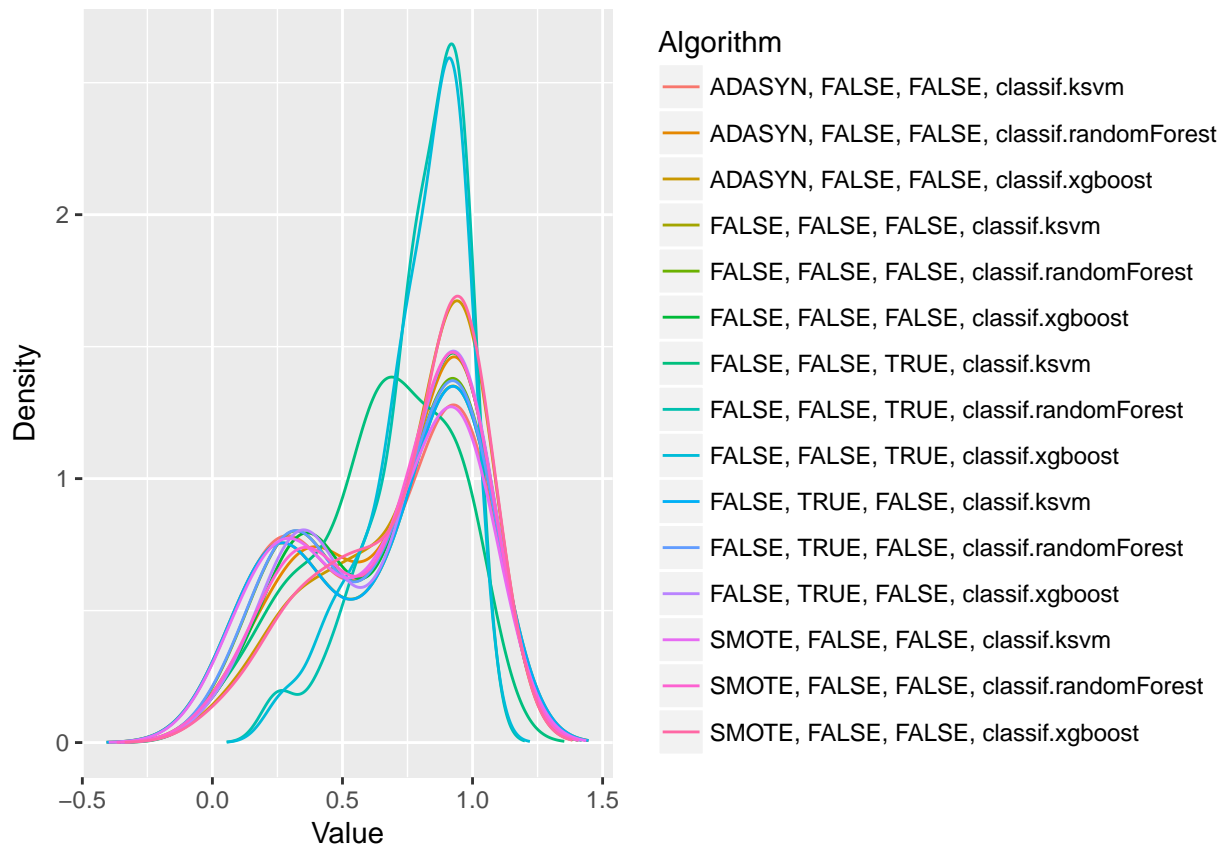
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.633907593305194"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.694053974640758"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.703887285006395"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.629443338349023"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.647525706592087"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.658960156606477"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.637795721759456"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.775998938757073"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.773741725204044"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.628798680357285"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.634672901261654"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.658478176017329"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.609954394657382"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.674370840847573"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.705322232154112"
```

## 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 = 96.283, df = 14, p-value = 2.431e-14
```

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



```

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

```

```

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

```

```

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

```

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

## Plotando os ranks

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

```
##      ADASYN, FALSE, FALSE, classif.ksvm
##      10.9875
## ADASYN, FALSE, FALSE, classif.randomForest
##      8.6750
##      ADASYN, FALSE, FALSE, classif.xgboost
##      5.5000
##      FALSE, FALSE, FALSE, classif.ksvm
##      9.3500
## FALSE, FALSE, FALSE, classif.randomForest
##      8.7625
##      FALSE, FALSE, FALSE, classif.xgboost
##      7.4625
##      FALSE, FALSE, TRUE, classif.ksvm
##      8.9500
## FALSE, FALSE, TRUE, classif.randomForest
##      5.9000
##      FALSE, FALSE, TRUE, classif.xgboost
##      5.6125
##      FALSE, TRUE, FALSE, classif.ksvm
##      9.4000
## FALSE, TRUE, FALSE, classif.randomForest
##      8.8250
##      FALSE, TRUE, FALSE, classif.xgboost
##      7.0750
##      SMOTE, FALSE, FALSE, classif.ksvm
##      10.6625
## SMOTE, FALSE, FALSE, classif.randomForest
##      7.6875
##      SMOTE, FALSE, FALSE, classif.xgboost
##      5.1500
```

### Plotando grafico de Critical Difference

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

