

R Notebook

Parametros:

Measure = Matthews correlation coefficient
Columns = sampling, weight_space, underbagging, learner
Performance = tuning_measure
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           : 0 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:1800
##
##
## tuning_measure      holdout_measure      holdout_measure_residual
## Min. : -0.00646 Min. : -0.1370 Min. : -0.06817
## 1st Qu.: 0.23511 1st Qu.: 0.0000 1st Qu.: 0.02210
## Median : 0.81997 Median : 0.3764 Median : 0.19355
## Mean : 0.64034 Mean : 0.4305 Mean : 0.29627
## 3rd Qu.: 0.99727 3rd Qu.: 0.8152 3rd Qu.: 0.49996
## Max. : 1.00000 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. :0.001
## 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 :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 0.9711549
## 2 0.9840938
## 3 0.9976886
## 4 1.0000000
## 5 1.0000000
## 6 0.9973459
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9639089
## 2 0.9843147
## 3 0.9913813
## 4 1.0000000
## 5 0.9947464
## 6 0.9984513
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9729427 -0.00331073
## 2 0.9882984 0.03130559
## 3 0.9865396 0.00000000
## 4 1.0000000 0.84986516
## 5 0.9949727 0.53692615
## 6 0.9986729 0.66437344
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.0000000
## 2 0.3792653
```

```

## 3          0.0000000
## 4          1.0000000
## 5          0.5461331
## 6          0.8932206
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.0000000          0.03775174
## 2          0.39809263          0.09124314
## 3          0.03498764          0.07291374
## 4          0.99019903          0.83204223
## 5          0.70251830          0.42292522
## 6          0.85240511          0.86395957
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.04138595
## 2          0.16003091
## 3          0.17236753
## 4          0.49950982
## 5          0.26010542
## 6          0.65564107
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.05606272         -0.002739151
## 2          0.15758863          0.043297878
## 3          0.14976876          0.000000000
## 4          0.50474181          0.849865157
## 5          0.21629241          0.536926146
## 6          0.46284332          0.664373439
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.0000000
## 2          NA
## 3          0.0000000
## 4          1.0000000
## 5          0.5413920
## 6          0.8577859
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.0000000          0.9694672
## 2          0.41553378          0.9861198
## 3          0.03464031          0.9973734
## 4          0.99019903          1.0000000
## 5          0.63325625          1.0000000
## 6          0.85240511          0.9973574
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9644783
## 2          NA
## 3          0.9899064
## 4          1.0000000
## 5          0.9966342
## 6          0.9993362
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9745849
## 2          0.9895432
## 3          0.9885378
## 4          0.9994734
## 5          0.9968504
## 6          0.9988938

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.8777
## 1st Qu.:0.9916
## Median :0.9977
## Mean :0.9914
## 3rd Qu.:0.9994
## Max. :1.0000
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9639
## 1st Qu.:0.9915
## Median :0.9968
## Mean :0.9938
## 3rd Qu.:0.9990
## Max. :1.0000
## NA's :6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.6780 Min. : -0.003311
## 1st Qu.:0.9879 1st Qu.: 0.000000
## Median :0.9962 Median : 0.231339
## Mean :0.9853 Mean : 0.329084
## 3rd Qu.:0.9988 3rd Qu.: 0.566816
## Max. :1.0000 Max. : 1.000000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.00000
## 1st Qu.:0.06637
## Median :0.50140
## Mean :0.49511
## 3rd Qu.:0.87767
## Max. :1.00000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.0000 Min. :0.03775
## 1st Qu.:0.1496 1st Qu.:0.12823
## Median :0.5837 Median :0.36387
## Mean :0.5355 Mean :0.39029
## 3rd Qu.:0.8633 3rd Qu.:0.65133
## Max. :1.0000 Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.04139
## 1st Qu.:0.17380
## Median :0.27765
## Mean :0.35533
## 3rd Qu.:0.46782
## Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.02596 Min. : -0.002739
## 1st Qu.:0.15586 1st Qu.: 0.000000
## Median :0.25622 Median : 0.231339
```

```
## Mean :0.31237 Mean : 0.331758
## 3rd Qu.:0.38715 3rd Qu.: 0.566816
## Max. :1.00000 Max. : 1.000000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min. :0.00000
## 1st Qu.:0.07377
## Median :0.47971
## Mean :0.48795
## 3rd Qu.:0.85687
## Max. :1.00000
## NA's :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.8777
## 1st Qu.:0.1929 1st Qu.:0.9958
## Median :0.5722 Median :0.9981
## Mean :0.5383 Mean :0.9925
## 3rd Qu.:0.8664 3rd Qu.:1.0000
## Max. :1.0000 Max. :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min. :0.9645
## 1st Qu.:0.9927
## Median :0.9983
## Mean :0.9943
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :7
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.9702
## 1st Qu.:0.9903
## Median :0.9964
## Mean :0.9934
## 3rd Qu.:0.9989
## Max. :1.0000
##
```

Verificando a média de cada coluna selecionada

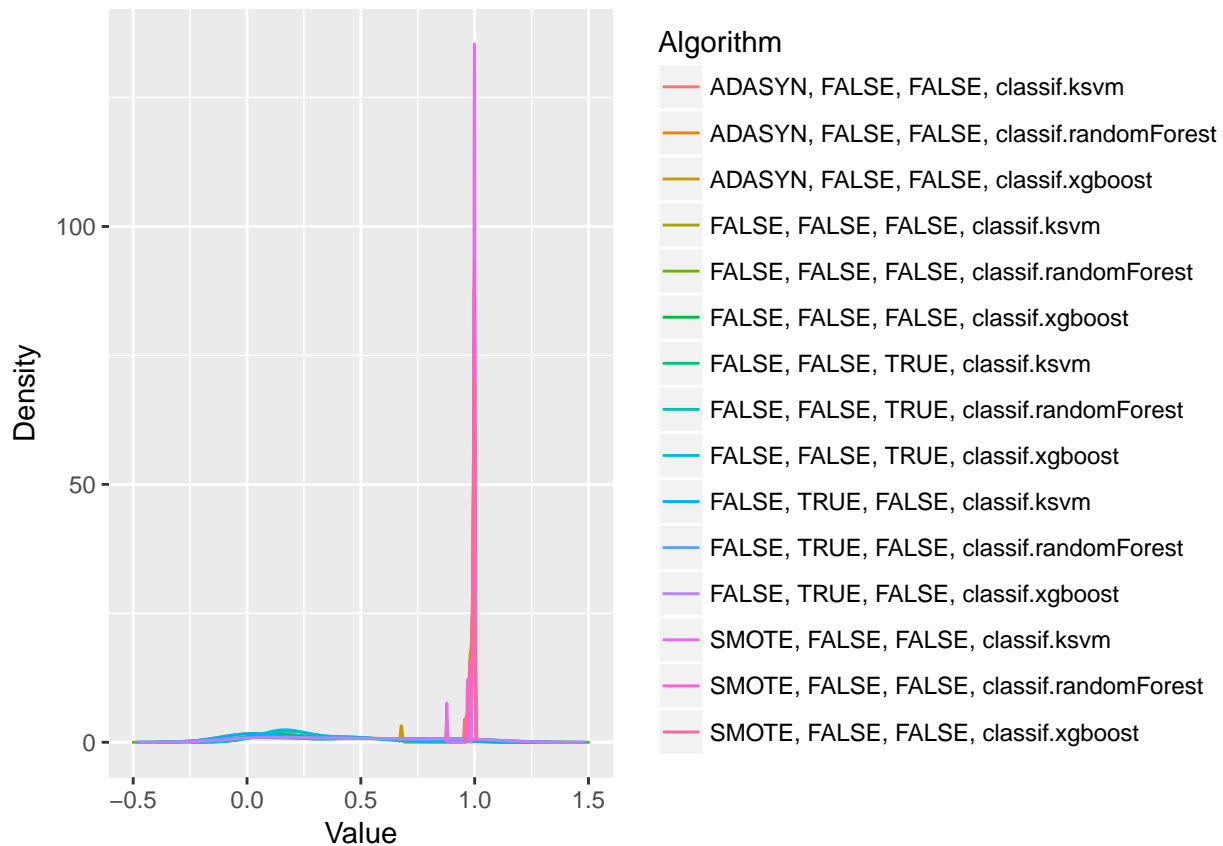
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.991406546238978"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.993765684643667"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.985296703454835"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.329084266711667"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.495112060290362"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.535458277855064"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.390291162248843"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.355333707183092"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.312372123809922"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.331757755779061"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.487946202223982"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.538346898451686"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.992481978663669"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.994253396373547"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.993375106025774"
```

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 = 306.14, 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,]                                     TRUE
## [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,]      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.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,]    FALSE
## [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,] FALSE
## [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,] FALSE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] TRUE
## [12,] FALSE
## [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
##      5.4625
##      ADASYN, FALSE, FALSE, classif.xgboost
##      4.4000
##      FALSE, FALSE, FALSE, classif.ksvm
##      12.3125
## FALSE, FALSE, FALSE, classif.randomForest
##      9.5375
##      FALSE, FALSE, FALSE, classif.xgboost
##      8.6375
##      FALSE, FALSE, TRUE, classif.ksvm
##      10.4000
## FALSE, FALSE, TRUE, classif.randomForest
##      10.8750
##      FALSE, FALSE, TRUE, classif.xgboost
##      11.3375
##      FALSE, TRUE, FALSE, classif.ksvm
##      12.1625
## FALSE, TRUE, FALSE, classif.randomForest
##      10.1625
##      FALSE, TRUE, FALSE, classif.xgboost
##      8.4000
##      SMOTE, FALSE, FALSE, classif.ksvm
##      2.9250
## SMOTE, FALSE, FALSE, classif.randomForest
##      5.5000
##      SMOTE, FALSE, FALSE, classif.xgboost
##      4.1250
```

Plotando grafico de Critical Difference

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

