

R Notebook

Parametros:

```
Measure = Accuracy
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          :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.:0.38526
## Median :0.9938 Median :0.99163 Median :0.75447
## Mean :0.9691 Mean :0.96664 Mean :0.66878
## 3rd Qu.:0.9990 3rd Qu.:0.99687 3rd Qu.:0.95350
## Max. :1.0000 Max. :1.00000 Max. :1.00000
## NA's :57 NA's :57 NA's :57
## 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 :57 (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 0.9853969
## 2 0.9920423
## 3 0.9987904
## 4 1.0000000
## 5 1.0000000
## 6 0.9986705
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9818597
## 2 0.9921523
## 3 0.9956889
## 4 1.0000000
## 5 0.9973705
## 6 0.9992247
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9864477 0.9878042
## 2 0.9941462 0.9891070
## 3 0.9932604 0.9900002
## 4 1.0000000 0.9976581
## 5 0.9986319 0.9933338
## 6 0.9993355 0.9943033
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.9897371
## 2 0.9913157
```

```

## 3          0.9900002
## 4          1.0000000
## 5          0.9931254
## 6          0.9975915
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.9897373          0.5674864
## 2          0.9915333          0.8833724
## 3          0.9901044          0.4496844
## 4          0.9997404          0.9976611
## 5          0.9945831          0.9895862
## 6          0.9967127          0.9973719
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.5220884
## 2          0.7848836
## 3          0.8140604
## 4          0.9656603
## 5          0.8943835
## 6          0.9857571
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.5574877          0.9883986
## 2          0.8011477          0.9896592
## 3          0.8115578          0.9900002
## 4          0.9282338          0.9976581
## 5          0.8864670          0.9933338
## 6          0.9627420          0.9943033
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.9897373
## 2          0.9913659
## 3          0.9900002
## 4          1.0000000
## 5          0.9931254
## 6          0.9971510
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.9897373          0.9845604
## 2          0.9916671          0.9930487
## 3          0.9900003          0.9986848
## 4          0.9997404          1.0000000
## 5          0.9941681          1.0000000
## 6          0.9967127          0.9986707
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9821508
## 2          NA
## 3          0.9949495
## 4          1.0000000
## 5          0.9983165
## 6          0.9996678
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9872722
## 2          0.9947695
## 3          0.9942656
## 4          0.9997364
## 5          0.9977904
## 6          0.9994461

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.9337
## 1st Qu.:0.9958
## Median :0.9988
## Mean :0.9955
## 3rd Qu.:0.9997
## Max. :1.0000
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9819
## 1st Qu.:0.9957
## Median :0.9983
## Mean :0.9968
## 3rd Qu.:0.9995
## Max. :1.0000
## NA's :7
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.9864 Min. :0.9878
## 1st Qu.:0.9940 1st Qu.:0.9899
## Median :0.9982 Median :0.9905
## Mean :0.9965 Mean :0.9921
## 3rd Qu.:0.9994 3rd Qu.:0.9937
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.9893
## 1st Qu.:0.9902
## Median :0.9931
## Mean :0.9938
## 3rd Qu.:0.9976
## Max. :1.0000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.9896 Min. :0.4327
## 1st Qu.:0.9904 1st Qu.:0.7905
## Median :0.9933 Median :0.9835
## Mean :0.9940 Mean :0.8819
## 3rd Qu.:0.9970 3rd Qu.:0.9934
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.5221
## 1st Qu.:0.7952
## Median :0.9132
## Mean :0.8731
## 3rd Qu.:0.9652
## Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.5026 Min. :0.9883
## 1st Qu.:0.7948 1st Qu.:0.9900
## Median :0.8964 Median :0.9906
```

```
## Mean :0.8549 Mean :0.9922
## 3rd Qu.:0.9450 3rd Qu.:0.9937
## Max. :1.0000 Max. :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min. :0.9893
## 1st Qu.:0.9901
## Median :0.9925
## Mean :0.9937
## 3rd Qu.:0.9974
## Max. :1.0000
## NA's :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min. :0.9896 Min. :0.9337
## 1st Qu.:0.9905 1st Qu.:0.9979
## Median :0.9936 Median :0.9990
## Mean :0.9941 Mean :0.9961
## 3rd Qu.:0.9973 3rd Qu.:1.0000
## Max. :1.0000 Max. :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min. :0.9822
## 1st Qu.:0.9971
## Median :0.9993
## Mean :0.9974
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :7
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.9850
## 1st Qu.:0.9951
## Median :0.9982
## Mean :0.9967
## 3rd Qu.:0.9995
## Max. :1.0000
##
```

Verificando a média de cada coluna selecionada

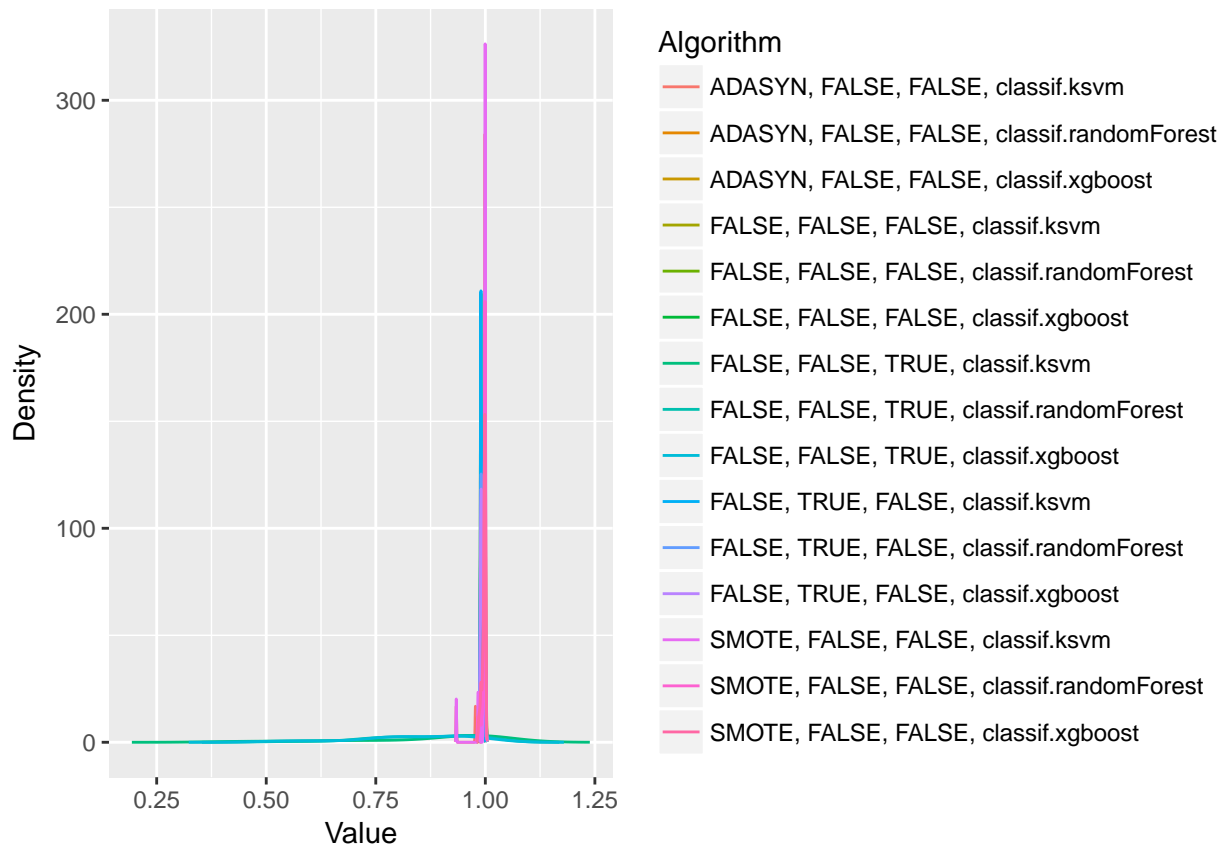
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.995511570830375"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.996796214277836"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.996542793985662"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.992104909528212"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.993814923112317"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.994020154009022"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.881932503018145"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.873131594878238"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.854945168435446"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.992194197714511"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.993661685120893"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.994098941840879"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.996092752637003"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.997362289446068"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.996667273471713"
```

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 = 279.93, 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,]                                     FALSE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    FALSE
## [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,]                                    FALSE
## [12,]                                    FALSE
## [13,]                                    FALSE
## [14,]                                    FALSE
## [15,]                                    FALSE
##      ADASYN, FALSE, FALSE, classif.xgboost
## [1,]                                     FALSE
## [2,]                                     FALSE
## [3,]                                     FALSE
## [4,]                                     TRUE
## [5,]                                     FALSE
## [6,]                                     FALSE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    FALSE
## [12,]                                    FALSE
## [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,]     FALSE
## [4,]     FALSE
## [5,]     FALSE
## [6,]     FALSE
## [7,]     TRUE
## [8,]     TRUE
## [9,]     TRUE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]    FALSE
## [15,]    FALSE
##      FALSE, FALSE, FALSE, classif.xgboost
## [1,]     FALSE
## [2,]     FALSE
## [3,]     FALSE
## [4,]     FALSE
## [5,]     FALSE
## [6,]     FALSE
## [7,]     TRUE
## [8,]     TRUE
## [9,]     TRUE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]    FALSE
## [15,]    FALSE
##      FALSE, FALSE, TRUE, classif.ksvm
## [1,]      TRUE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]      TRUE

```

```

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

```

```

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

Plotando os ranks

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

```
##      ADASYN, FALSE, FALSE, classif.ksvm
##      4.5875
## ADASYN, FALSE, FALSE, classif.randomForest
##      6.1375
##      ADASYN, FALSE, FALSE, classif.xgboost
##      5.1750
##      FALSE, FALSE, FALSE, classif.ksvm
##      10.2875
## FALSE, FALSE, FALSE, classif.randomForest
##      8.0875
##      FALSE, FALSE, FALSE, classif.xgboost
##      7.2375
##      FALSE, FALSE, TRUE, classif.ksvm
##      11.6250
## FALSE, FALSE, TRUE, classif.randomForest
##      13.2250
##      FALSE, FALSE, TRUE, classif.xgboost
##      13.5875
##      FALSE, TRUE, FALSE, classif.ksvm
##      10.1375
## FALSE, TRUE, FALSE, classif.randomForest
##      8.5125
##      FALSE, TRUE, FALSE, classif.xgboost
##      7.0750
##      SMOTE, FALSE, FALSE, classif.ksvm
##      3.6000
## SMOTE, FALSE, FALSE, classif.randomForest
##      5.8250
##      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) {})
```

