

R Notebook

Parametros:

Measure = Area under the curve
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           : 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.:0.9553 1st Qu.:0.8921 1st Qu.:0.7377
## Median :0.9993 Median :0.9916 Median :0.9069
## Mean :0.9502 Mean :0.9126 Mean :0.8460
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:0.9840
## Max. :1.0000 Max. :1.0000 Max. :1.0000
## NA's :42 NA's :42 NA's :42
## 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 :42 (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 0.5136216
## 2 NA
## 3 0.7583631
## 4 0.9337170
## 5 0.9224985
## 6 0.8857046
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.6617140
## 2 NA
## 3 0.8439311
## 4 0.7496955
## 5 0.9729017
## 6 0.7728464
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.6206414 0.5133720
## 2 0.8708447 0.4549243
## 3 0.8147400 0.7469161
## 4 0.6692242 0.8590001
## 5 0.9755048 0.9703766
## 6 0.8442439 0.8672659
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.6224639
## 2 0.8601656
```

```

## 3          0.8316542
## 4          0.7703462
## 5          0.9830413
## 6          0.7798361
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.6375083          0.6259505
## 2          0.9023521          0.7943273
## 3          0.8204698          0.7440871
## 4          0.7190811          0.8088235
## 5          0.9625229          0.6252906
## 6          0.8821091          0.6192346
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.6705058
## 2          0.8795716
## 3          0.8439155
## 4          0.8157274
## 5          0.9665499
## 6          0.7516526
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.6359470          0.5137510
## 2          0.8885925          0.5371265
## 3          0.8108122          0.7469161
## 4          0.7154199          0.8560772
## 5          0.9734690          0.9703766
## 6          0.7777879          0.8672659
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.6178971
## 2          NA
## 3          0.8268784
## 4          0.7814974
## 5          0.9817732
## 6          0.7700843
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.6375879          0.5167415
## 2          0.9015632          0.6257369
## 3          0.8206660          0.7616514
## 4          0.7242875          0.9305505
## 5          0.9627232          0.9551922
## 6          0.8864817          0.8325538
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.6600339
## 2          0.8664327
## 3          0.8518726
## 4          0.6328705
## 5          0.9740197
## 6          0.7605478
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.6217660
## 2          0.8685851
## 3          0.8095652
## 4          0.6792032
## 5          0.9779298
## 6          0.7812734

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.4882
## 1st Qu.:0.7250
## Median :0.8243
## Mean :0.8102
## 3rd Qu.:0.9394
## Max. :1.0000
## NA's :3
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.6035
## 1st Qu.:0.8469
## Median :0.9576
## Mean :0.8935
## 3rd Qu.:0.9805
## Max. :1.0000
## NA's :6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.5704 Min. :0.4549
## 1st Qu.:0.7888 1st Qu.:0.7339
## Median :0.9261 Median :0.8486
## Mean :0.8619 Mean :0.8197
## 3rd Qu.:0.9854 3rd Qu.:0.9742
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.4953
## 1st Qu.:0.8044
## Median :0.9342
## Mean :0.8644
## 3rd Qu.:0.9797
## Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.5704 Min. :0.5009
## 1st Qu.:0.8118 1st Qu.:0.6258
## Median :0.9225 Median :0.7851
## Mean :0.8761 Mean :0.7641
## 3rd Qu.:0.9855 3rd Qu.:0.8949
## Max. :1.0000 Max. :0.9986
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.5199
## 1st Qu.:0.7513
## Median :0.9048
## Mean :0.8571
## 3rd Qu.:0.9837
## Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.5153 Min. :0.4988
## 1st Qu.:0.7695 1st Qu.:0.7339
## Median :0.9026 Median :0.8471
```

```
## Mean      :0.8594                      Mean      :0.8207
## 3rd Qu.:0.9755                      3rd Qu.:0.9742
## Max.      :1.0000                      Max.      :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.5008
## 1st Qu.:0.7815
## Median :0.9607
## Mean      :0.8795
## 3rd Qu.:0.9935
## Max.      :1.0000
## NA's      :3
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.5437                      Min.      :0.4858
## 1st Qu.:0.7935                      1st Qu.:0.6569
## Median :0.9184                      Median :0.8160
## Mean      :0.8725                      Mean      :0.7930
## 3rd Qu.:0.9794                      3rd Qu.:0.9451
## Max.      :1.0000                      Max.      :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.5061
## 1st Qu.:0.7787
## Median :0.9259
## Mean      :0.8645
## 3rd Qu.:0.9862
## Max.      :1.0000
## NA's      :2
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.5112
## 1st Qu.:0.7790
## Median :0.9260
## Mean      :0.8617
## 3rd Qu.:0.9868
## Max.      :1.0000
##
```

Verificando a média de cada coluna selecionada

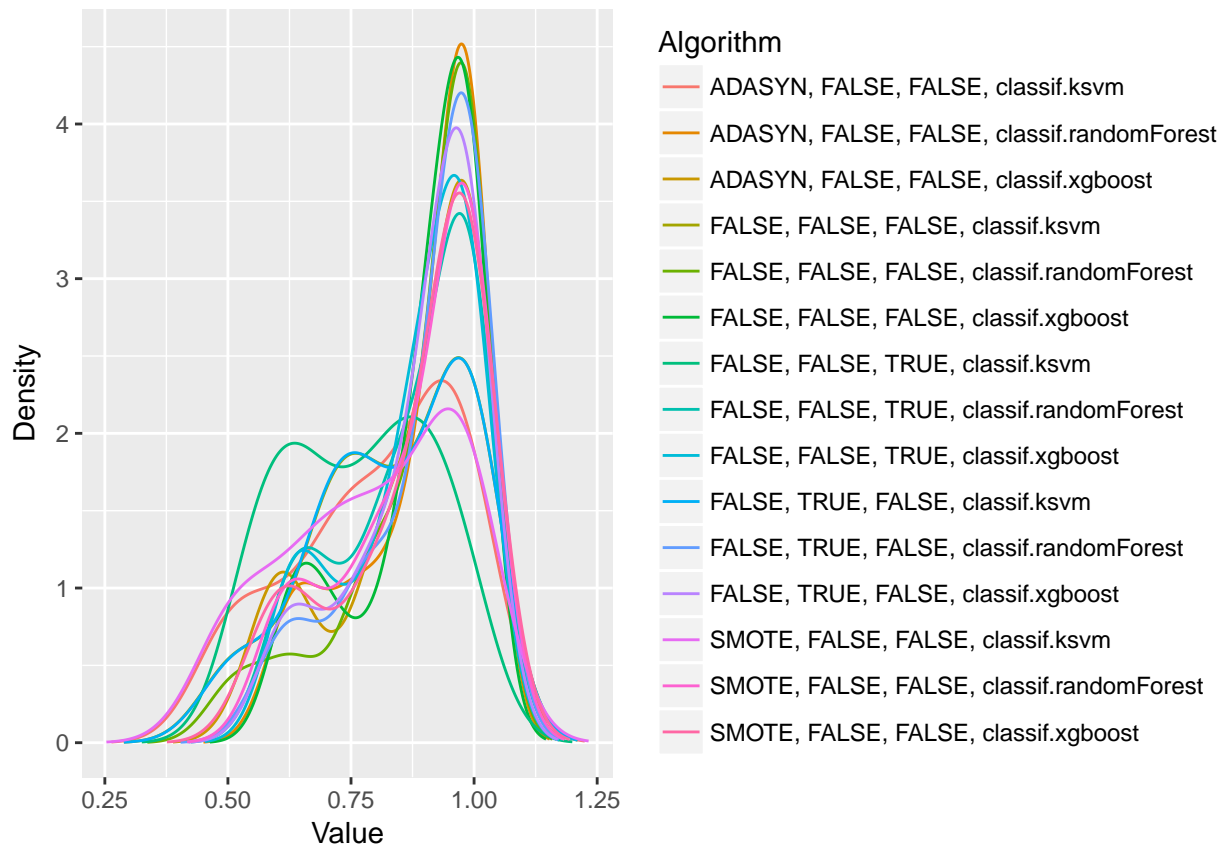
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.810192517620008"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.893498076021079"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.861931463889398"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.819684267348074"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.864391237411296"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.876095504804726"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.764052695360382"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.857140840312252"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.859448061063698"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.820687355828643"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.879516684013695"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.872496664911349"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.792984752389399"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.864526774876087"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.861650382410199"
```

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 = 88.768, df = 14, p-value = 6.481e-13
```

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



```

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

```

```

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

```

```

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

```

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

Plotando os ranks

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

```
##      ADASYN, FALSE, FALSE, classif.ksvm
##      10.3375
## ADASYN, FALSE, FALSE, classif.randomForest
##      6.9250
##      ADASYN, FALSE, FALSE, classif.xgboost
##      7.0750
##      FALSE, FALSE, FALSE, classif.ksvm
##      8.9375
## FALSE, FALSE, FALSE, classif.randomForest
##      6.2750
##      FALSE, FALSE, FALSE, classif.xgboost
##      6.3750
##      FALSE, FALSE, TRUE, classif.ksvm
##      12.2750
## FALSE, FALSE, TRUE, classif.randomForest
##      7.4250
##      FALSE, FALSE, TRUE, classif.xgboost
##      8.4125
##      FALSE, TRUE, FALSE, classif.ksvm
##      8.9625
## FALSE, TRUE, FALSE, classif.randomForest
##      6.2500
##      FALSE, TRUE, FALSE, classif.xgboost
##      6.7000
##      SMOTE, FALSE, FALSE, classif.ksvm
##      9.9875
## SMOTE, FALSE, FALSE, classif.randomForest
##      6.7750
##      SMOTE, FALSE, FALSE, classif.xgboost
##      7.2875
```

Plotando grafico de Critical Difference

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

