

R Notebook

Parametros:

Measure = Area under the curve
Columns = sampling, weight_space, underbagging, learner
Performance = holdout_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 0.5908544
## 2 NA
## 3 0.6696128
## 4 1.0000000
## 5 0.8696338
## 6 1.0000000
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.6115523
## 2 0.8760552
## 3 0.8433817
## 4 1.0000000
## 5 0.9885311
## 6 1.0000000
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.5329723 0.5048135
## 2 0.8532475 0.4916767
## 3 0.7760417 0.7340067
## 4 1.0000000 1.0000000
## 5 0.9934764 0.9558081
## 6 1.0000000 1.0000000
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.7033093
## 2 0.8886010
```

```

## 3          0.8314657
## 4          1.0000000
## 5          0.9900042
## 6          1.0000000
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.6906137          0.5738869
## 2          0.9002132          0.8178913
## 3          0.9024358          0.7362952
## 4          1.0000000          1.0000000
## 5          0.9619108          0.8627946
## 6          1.0000000          0.9440012
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.6041516
## 2          0.9036622
## 3          0.8363584
## 4          1.0000000
## 5          0.9835859
## 6          1.0000000
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.6801444          0.5037304
## 2          0.9019359          0.5547681
## 3          0.8885732          0.7340067
## 4          0.9992968          1.0000000
## 5          0.9436027          0.9558081
## 6          0.9943853          1.0000000
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.6951264
## 2          NA
## 3          0.8935448
## 4          1.0000000
## 5          0.9800084
## 6          1.0000000
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.6904934          0.5335740
## 2          0.8966787          0.6074513
## 3          0.9065920          0.7012837
## 4          1.0000000          1.0000000
## 5          0.9587542          0.9156145
## 6          1.0000000          1.0000000
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.6535499
## 2          NA
## 3          0.8974905
## 4          1.0000000
## 5          0.9903199
## 6          1.0000000
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.5206980
## 2          0.8799975
## 3          0.7599432
## 4          1.0000000
## 5          0.9903199
## 6          0.9997045

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.5048
## 1st Qu.:0.7857
## Median :0.8908
## Mean :0.8522
## 3rd Qu.:0.9986
## Max. :1.0000
## NA's :3
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.5676
## 1st Qu.:0.9336
## Median :0.9819
## Mean :0.9337
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.5330 Min. :0.4917
## 1st Qu.:0.9130 1st Qu.:0.7393
## Median :0.9914 Median :0.9524
## Mean :0.9331 Mean :0.8654
## 3rd Qu.:0.9997 3rd Qu.:0.9992
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.5522
## 1st Qu.:0.9280
## Median :0.9900
## Mean :0.9392
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.6906 Min. :0.5739
## 1st Qu.:0.9279 1st Qu.:0.8113
## Median :0.9933 Median :0.8733
## Mean :0.9507 Mean :0.8567
## 3rd Qu.:0.9998 3rd Qu.:0.9466
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.5529
## 1st Qu.:0.9175
## Median :0.9909
## Mean :0.9324
## 3rd Qu.:0.9982
## Max. :1.0000
## NA's :1
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.5882 Min. :0.5037
## 1st Qu.:0.9197 1st Qu.:0.7393
## Median :0.9815 Median :0.9524
```

```
## Mean :0.9349 Mean :0.8663
## 3rd Qu.:0.9974 3rd Qu.:0.9992
## Max. :1.0000 Max. :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min. :0.6774
## 1st Qu.:0.9325
## Median :0.9972
## Mean :0.9480
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min. :0.6905 Min. :0.5026
## 1st Qu.:0.9055 1st Qu.:0.7289
## Median :0.9894 Median :0.9171
## Mean :0.9485 Mean :0.8471
## 3rd Qu.:0.9998 3rd Qu.:0.9982
## Max. :1.0000 Max. :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min. :0.6535
## 1st Qu.:0.9132
## Median :0.9923
## Mean :0.9441
## 3rd Qu.:1.0000
## Max. :1.0000
## NA's :4
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.4858
## 1st Qu.:0.9437
## Median :0.9903
## Mean :0.9327
## 3rd Qu.:0.9997
## Max. :1.0000
##
```

Verificando a média de cada coluna selecionada

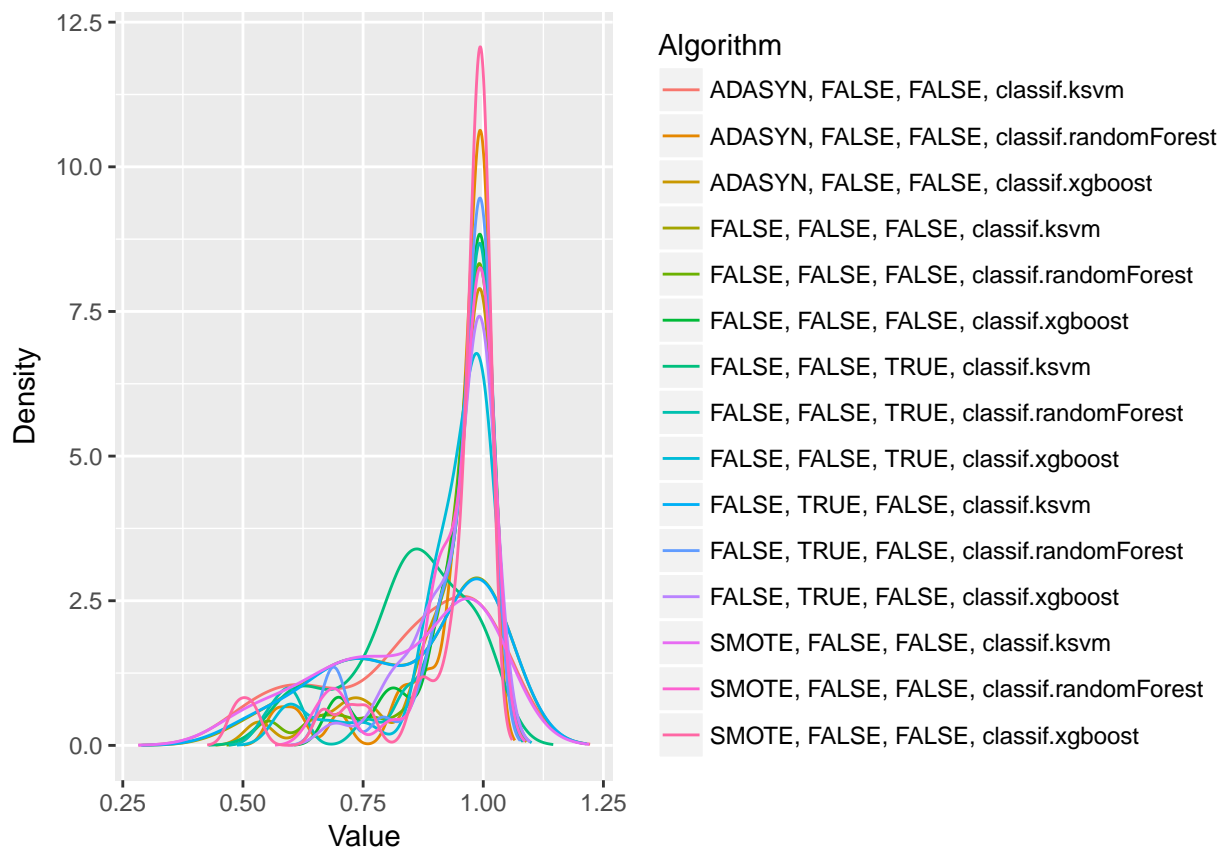
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.852232642605234"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.933734541578364"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.933114321485676"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.865384158603188"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.939241987240982"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.950695748958464"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.856662254538743"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.932435002711275"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.934894712184892"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.866264862403333"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.947967554064456"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.948533061331522"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.847050605409089"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.944119962010488"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.932740026711086"
```

Fazendo teste de normalidade

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



Testando as diferencas

```
friedmanTest(df)
```

```
##
## Friedman's rank sum test
##
## data: df
## Friedman's chi-squared = 91.586, df = 14, p-value = 1.903e-13
```

Testando as diferenças 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,]                                     FALSE
## [5,]                                     TRUE
## [6,]                                     TRUE
## [7,]                                     FALSE
## [8,]                                     FALSE
## [9,]                                     FALSE
## [10,]                                    FALSE
## [11,]                                    TRUE
## [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,]                                     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,]     TRUE
## [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,] FALSE
## [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 TRUE
## [7,] FALSE 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,] TRUE
## [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,] 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,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] FALSE
## [11,] FALSE
## [12,] FALSE
## [13,] FALSE
## [14,] FALSE
## [15,] FALSE
## SMOTE, 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
```

Plotando os ranks

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

```
##          ADASYN, FALSE, FALSE, classif.ksvm
##                               10.3625
## ADASYN, FALSE, FALSE, classif.randomForest
##                               7.3250
##          ADASYN, FALSE, FALSE, classif.xgboost
##                               7.3625
##          FALSE, FALSE, FALSE, classif.ksvm
##                               8.9625
## FALSE, FALSE, FALSE, classif.randomForest
##                               6.2125
##          FALSE, FALSE, FALSE, classif.xgboost
##                               5.6375
##          FALSE, FALSE, TRUE, classif.ksvm
##                               11.5500
## FALSE, FALSE, TRUE, classif.randomForest
##                               8.0250
##          FALSE, FALSE, TRUE, classif.xgboost
##                               8.6500
##          FALSE, TRUE, FALSE, classif.ksvm
##                               9.0875
## FALSE, TRUE, FALSE, classif.randomForest
##                               6.0500
##          FALSE, TRUE, FALSE, classif.xgboost
##                               6.2375
##          SMOTE, FALSE, FALSE, classif.ksvm
##                               10.5375
## SMOTE, FALSE, FALSE, classif.randomForest
##                               7.2125
##          SMOTE, FALSE, FALSE, classif.xgboost
##                               6.7875
```

Plotando grafico de Critical Difference

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

