

R Notebook

Parametros:

```
Measure = G-mean
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           : 0 ADASYN: 360 Mode :logical
## Area under the curve : 0 FALSE :1080 FALSE:1440
## F1 measure          : 0 SMOTE : 360 TRUE :360
## G-mean              :1800 NA's :0
## Matthews correlation coefficient: 0
##
##
## tuning_measure holdout_measure holdout_measure_residual
## Min. :0.0000 Min. :0.0000 Min. :0.0000
## 1st Qu.:0.5895 1st Qu.:0.0000 1st Qu.:0.1104
## Median :0.9629 Median :0.7066 Median :0.4187
## Mean :0.7515 Mean :0.5605 Mean :0.4386
## 3rd Qu.:0.9987 3rd Qu.:0.9645 3rd Qu.:0.7566
## Max. :1.0000 Max. :1.0000 Max. :1.0000
## NA's :54 NA's :54 NA's :54
## 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 :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 0.9853254
## 2 0.9920394
## 3 0.9988425
## 4 1.0000000
## 5 1.0000000
## 6 0.9986692
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9815988
## 2 NA
## 3 0.9956872
## 4 1.0000000
## 5 0.9973684
## 6 0.9992244
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9864427 0.0000000
## 2 0.9941441 0.1484512
## 3 0.9938971 0.0000000
## 4 1.0000000 0.8504800
## 5 0.9979991 0.5827798
## 6 0.9993355 0.6661380
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.0000000
## 2 NA
```

```

## 3          0.0000000
## 4          1.0000000
## 5          0.5479681
## 6          0.9302375
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.0000000          0.5502782
## 2          0.43718890          0.6354148
## 3          0.03513642          0.6021962
## 4          0.99986859          0.8711040
## 5          0.72523102          0.8260606
## 6          0.92968345          0.9012045
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.5895215
## 2          0.8106157
## 3          0.8303013
## 4          0.9824784
## 5          0.9134414
## 6          0.9928862
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.6168405          0.0000000
## 2          0.8202719          0.1661451
## 3          0.8083963          0.0000000
## 4          0.9628047          0.8504800
## 5          0.8657657          0.5827798
## 6          0.9696503          0.6661380
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.0000000
## 2          NA
## 3          0.03513642
## 4          1.0000000
## 5          0.55592733
## 6          0.93398195
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.0000000          0.9845000
## 2          0.43375517          0.9930521
## 3          0.03513642          0.9986838
## 4          0.99986859          1.0000000
## 5          0.60713092          1.0000000
## 6          0.92968345          0.9986654
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9817433
## 2          NA
## 3          0.9949476
## 4          1.0000000
## 5          0.9983162
## 6          0.9996676
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9872697
## 2          0.9947643
## 3          0.9942639
## 4          0.9997369
## 5          0.9977880
## 6          0.9994456

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.9299
## 1st Qu.:0.9958
## Median :0.9988
## Mean :0.9954
## 3rd Qu.:0.9997
## Max. :1.0000
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9816
## 1st Qu.:0.9962
## Median :0.9986
## Mean :0.9971
## 3rd Qu.:0.9995
## Max. :1.0000
## NA's :6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.6792 Min. :0.0000
## 1st Qu.:0.9938 1st Qu.:0.0000
## Median :0.9981 Median :0.2587
## Mean :0.9886 Mean :0.3558
## 3rd Qu.:0.9994 3rd Qu.:0.6391
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.00000
## 1st Qu.:0.05815
## Median :0.53894
## Mean :0.51766
## 3rd Qu.:0.89738
## Max. :1.00000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.0000 Min. :0.5133
## 1st Qu.:0.1802 1st Qu.:0.6441
## Median :0.6153 Median :0.7699
## Mean :0.5594 Mean :0.7620
## 3rd Qu.:0.9162 3rd Qu.:0.8724
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.5895
## 1st Qu.:0.8378
## Median :0.9144
## Mean :0.8932
## 3rd Qu.:0.9813
## Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.5341 Min. :0.0000
## 1st Qu.:0.8257 1st Qu.:0.0000
## Median :0.9041 Median :0.2587
```

```
## Mean      :0.8770                      Mean      :0.3548
## 3rd Qu.   :0.9634                      3rd Qu.   :0.6252
## Max.      :1.0000                      Max.      :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.00000
## 1st Qu.   :0.08042
## Median    :0.52815
## Mean      :0.50085
## 3rd Qu.   :0.90303
## Max.      :1.00000
## NA's      :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.0000                      Min.      :0.9299
## 1st Qu.   :0.1967                      1st Qu.   :0.9979
## Median    :0.5925                      Median    :0.9990
## Mean      :0.5543                      Mean      :0.9960
## 3rd Qu.   :0.9098                      3rd Qu.   :1.0000
## Max.      :1.0000                      Max.      :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.9817
## 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.      :0.9849
## 1st Qu.   :0.9948
## Median    :0.9982
## Mean      :0.9966
## 3rd Qu.   :0.9995
## Max.      :1.0000
##
```

Verificando a média de cada coluna selecionada

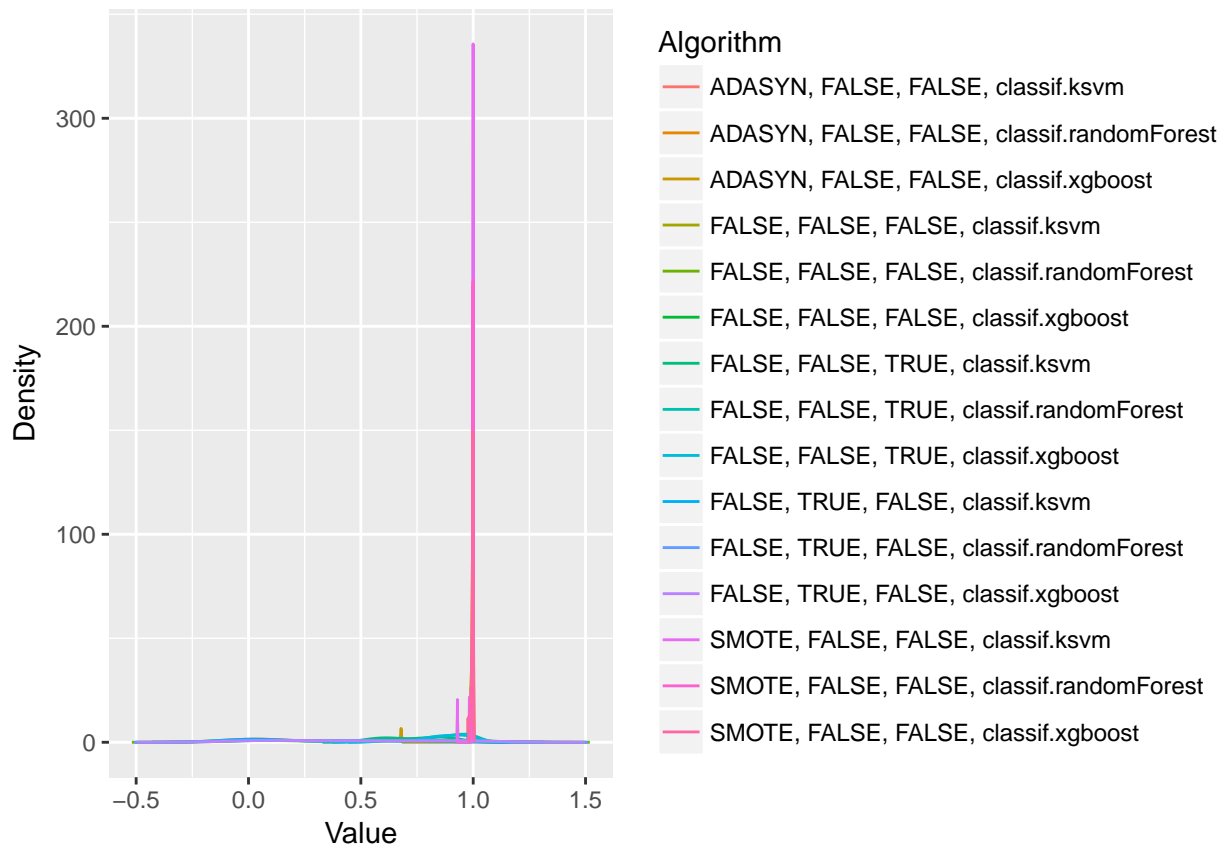
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.99541846799912"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.997053635129182"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.988610737787024"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.355765188875386"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.517657641904766"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.559381422542501"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.761964765268466"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.893244538922893"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.87704054021134"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.354778268780595"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.500850813174842"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.554292240137955"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.995991057108736"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.997355706649823"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.996637147042397"
```

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 = 329.99, 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,]                                     TRUE
## [7,]                                     TRUE
## [8,]                                     FALSE
## [9,]                                     FALSE
## [10,]                                    TRUE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [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,]                                     FALSE
## [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,]     FALSE
## [7,]     FALSE
## [8,]      TRUE
## [9,]      TRUE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [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,]      TRUE
## [9,]     FALSE
## [10,]    FALSE
## [11,]    FALSE
## [12,]    FALSE
## [13,]     TRUE
## [14,]     TRUE
## [15,]     TRUE
##      FALSE, FALSE, FALSE, classif.xgboost
## [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.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,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] TRUE
## [12,] FALSE
## [13,] TRUE
## [14,] FALSE
## [15,] FALSE
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## [1,] TRUE TRUE
## [2,] FALSE TRUE
## [3,] TRUE TRUE
## [4,] TRUE FALSE
## [5,] FALSE FALSE
## [6,] FALSE FALSE
## [7,] FALSE FALSE
## [8,] FALSE TRUE
## [9,] FALSE TRUE
## [10,] TRUE FALSE
## [11,] TRUE FALSE
## [12,] FALSE FALSE
## [13,] TRUE TRUE
## [14,] FALSE TRUE
## [15,] TRUE TRUE
## FALSE, TRUE, FALSE, classif.randomForest
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
## [4,] FALSE
## [5,] FALSE
## [6,] FALSE
## [7,] FALSE
## [8,] TRUE
## [9,] TRUE
## [10,] FALSE
## [11,] FALSE

```

```

## [12,] FALSE
## [13,] TRUE
## [14,] TRUE
## [15,] TRUE
## FALSE, TRUE, FALSE, classif.xgboost
## [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
## 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,] TRUE
## [7,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [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,] FALSE
## [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
##      5.4875
##      ADASYN, FALSE, FALSE, classif.xgboost
##      4.4000
##      FALSE, FALSE, FALSE, classif.ksvm
##      12.9500
## FALSE, FALSE, FALSE, classif.randomForest
##      11.0125
##      FALSE, FALSE, FALSE, classif.xgboost
##      10.2000
##      FALSE, FALSE, TRUE, classif.ksvm
##      9.6000
## FALSE, FALSE, TRUE, classif.randomForest
##      7.4250
##      FALSE, FALSE, TRUE, classif.xgboost
##      7.8625
##      FALSE, TRUE, FALSE, classif.ksvm
##      12.9125
## FALSE, TRUE, FALSE, classif.randomForest
##      11.4500
##      FALSE, TRUE, FALSE, classif.xgboost
##      10.2375
##      SMOTE, FALSE, FALSE, classif.ksvm
##      3.0000
## SMOTE, FALSE, FALSE, classif.randomForest
##      5.3750
##      SMOTE, FALSE, FALSE, classif.xgboost
##      4.3250
```

Plotando grafico de Critical Difference

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

