

R Notebook

Parametros:

```
Measure = F1 measure
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.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          :1800 SMOTE : 360 TRUE :360
## G-mean              : 0 NA's :0
## Matthews correlation coefficient: 0
##
##
## tuning_measure  holdout_measure  holdout_measure_residual
## Min. :0.0000 Min. :0.0000 Min. :0.00000
## 1st Qu.:0.1475 1st Qu.:0.0000 1st Qu.:0.02254
## Median :0.8030 Median :0.3333 Median :0.20700
## Mean :0.6194 Mean :0.4107 Mean :0.32309
## 3rd Qu.:0.9986 3rd Qu.:0.8000 3rd Qu.:0.58363
## Max. :1.0000 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. :0.01
## 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 0.0000000
## 2 0.0000000
## 3 0.0000000
## 4 0.4333333
## 5 0.0000000
## 6 0.1666667
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.0000000
## 2 NA
## 3 0.0000000
## 4 1.0000000
## 5 0.4920635
## 6 1.0000000
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.0000000 0.00000000
## 2 0.2353875 0.01886792
## 3 0.0000000 0.00000000
## 4 0.9523810 0.83333333
## 5 0.5833333 0.57777778
## 6 0.7746032 0.70000000
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.0000000
## 2 0.4247251
```

```

## 3          0.0000000
## 4          1.0000000
## 5          0.2666667
## 6          0.9523810
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.0000000          0.02321195
## 2          0.3636878          0.06800533
## 3          0.0000000          0.04408668
## 4          1.0000000          1.00000000
## 5          0.4000000          0.00000000
## 6          0.9047619          0.43333333
## FALSE, FALSE, TRUE, classif.randomForest
## 1          0.02668038
## 2          0.07357207
## 3          0.08902246
## 4          0.63419913
## 5          0.16893085
## 6          0.67222222
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.03142433          0.00000000
## 2          0.08240072          0.01626016
## 3          0.07465224          0.00000000
## 4          0.20767860          0.83333333
## 5          0.12934386          0.57777778
## 6          0.37662338          0.70000000
## FALSE, TRUE, FALSE, classif.randomForest
## 1          0.000000
## 2          NA
## 3          0.000000
## 4          1.000000
## 5          0.400000
## 6          0.952381
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.0000000          0.00000000
## 2          0.4037524          0.02145474
## 3          0.0000000          0.00000000
## 4          1.0000000          0.60000000
## 5          0.6412698          0.13333333
## 6          0.9047619          0.16666667
## SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.0000000
## 2          0.1192507
## 3          0.0000000
## 4          1.0000000
## 5          0.4444444
## 6          0.9333333
## SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.0000000
## 2          0.2480555
## 3          0.0000000
## 4          1.0000000
## 5          0.5873016
## 6          0.8380952

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.0000
## 1st Qu.:0.0000
## Median :0.0000
## Mean :0.1799
## 3rd Qu.:0.2830
## Max. :0.9432
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.1000
## Median :0.4921
## Mean :0.5016
## 3rd Qu.:0.9333
## Max. :1.0000
## NA's :9
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.0000
## 1st Qu.:0.2766 1st Qu.:0.0000
## Median :0.6364 Median :0.2004
## Mean :0.5846 Mean :0.3350
## 3rd Qu.:0.9480 3rd Qu.:0.6750
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.0000
## Median :0.5152
## Mean :0.5027
## 3rd Qu.:0.9060
## Max. :1.0000
## NA's :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.00000 Min. :0.00000
## 1st Qu.:0.09226 1st Qu.:0.04327
## Median :0.56399 Median :0.13333
## Mean :0.51197 Mean :0.31390
## 3rd Qu.:0.89771 3rd Qu.:0.56534
## Max. :1.00000 Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.02136
## 1st Qu.:0.07850
## Median :0.20627
## Mean :0.31027
## 3rd Qu.:0.51707
## Max. :1.00000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.01227 Min. :0.0000
## 1st Qu.:0.07766 1st Qu.:0.0000
## Median :0.15082 Median :0.1574
```

```
## Mean :0.23621 Mean :0.3322
## 3rd Qu.:0.28159 3rd Qu.:0.6750
## Max. :1.00000 Max. :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.0000
## Median :0.5788
## Mean :0.5105
## 3rd Qu.:0.9083
## Max. :1.0000
## NA's :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.00000
## 1st Qu.:0.1551 1st Qu.:0.00000
## Median :0.6391 Median :0.03174
## Mean :0.5282 Mean :0.21525
## 3rd Qu.:0.8744 3rd Qu.:0.42013
## Max. :1.0000 Max. :1.00000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.1583
## Median :0.5028
## Mean :0.5325
## 3rd Qu.:0.9333
## Max. :1.0000
## NA's :4
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.0000
## 1st Qu.:0.2731
## Median :0.6591
## Mean :0.5947
## 3rd Qu.:0.9246
## Max. :1.0000
##
```

Verificando a média de cada coluna selecionada

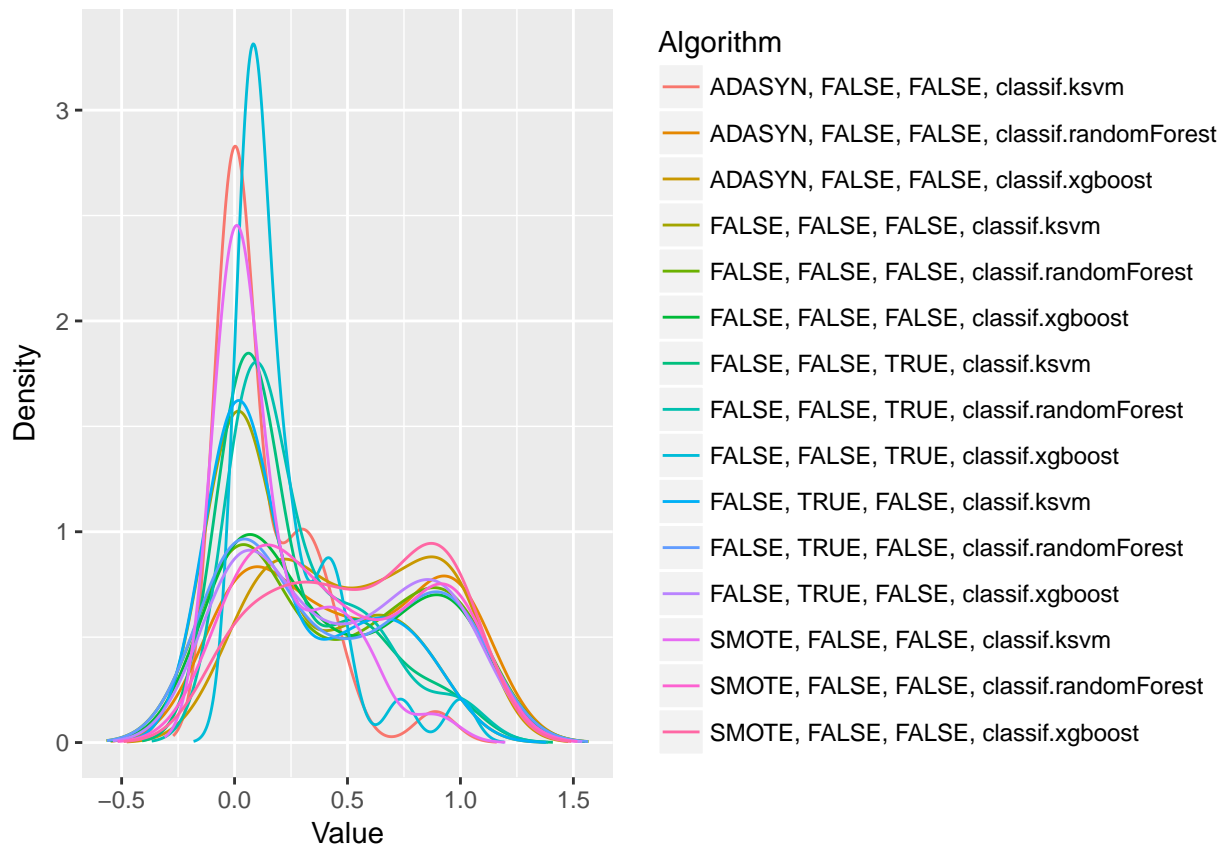
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.179897517848004"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.50162664028821"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.584604602979948"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.334954294254948"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.502683900732488"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.511965799752469"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.313895958425107"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.310266456906157"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.23621256422124"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.332207992944352"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.51045279000618"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.528203570753324"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.215253414568032"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.532462148538269"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.594720225809504"
```

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 = 140.84, 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,]                                     TRUE
## [3,]                                     TRUE
## [4,]                                     FALSE
## [5,]                                     TRUE
## [6,]                                     TRUE
## [7,]                                     FALSE
## [8,]                                     FALSE
## [9,]                                     FALSE
## [10,]                                    FALSE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [13,]                                    FALSE
## [14,]                                    TRUE
## [15,]                                    TRUE
##      ADASYN, FALSE, FALSE, classif.randomForest
## [1,]                                     TRUE
## [2,]                                     FALSE
## [3,]                                     TRUE
## [4,]                                     FALSE
## [5,]                                     FALSE
## [6,]                                     FALSE
## [7,]                                     FALSE
## [8,]                                     FALSE
## [9,]                                     FALSE
## [10,]                                    FALSE
## [11,]                                    FALSE
## [12,]                                    FALSE
## [13,]                                    TRUE
## [14,]                                    FALSE
## [15,]                                    TRUE
##      ADASYN, FALSE, FALSE, classif.xgboost
## [1,]                                     TRUE
## [2,]                                     TRUE
## [3,]                                     FALSE
## [4,]                                     TRUE
## [5,]                                     FALSE
## [6,]                                     FALSE
## [7,]                                     TRUE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    TRUE
## [11,]                                    FALSE
## [12,]                                    FALSE
## [13,]                                    TRUE
## [14,]                                    FALSE
## [15,]                                    FALSE
```



```

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

```

```

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

```

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

Plotando os ranks

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

```
##      ADASYN, FALSE, FALSE, classif.ksvm
##      11.5375
## ADASYN, FALSE, FALSE, classif.randomForest
##      7.8500
##      ADASYN, FALSE, FALSE, classif.xgboost
##      4.2125
##      FALSE, FALSE, FALSE, classif.ksvm
##      9.9250
## FALSE, FALSE, FALSE, classif.randomForest
##      7.2375
##      FALSE, FALSE, FALSE, classif.xgboost
##      6.6125
##      FALSE, FALSE, TRUE, classif.ksvm
##      8.5375
## FALSE, FALSE, TRUE, classif.randomForest
##      8.7000
##      FALSE, FALSE, TRUE, classif.xgboost
##      9.2250
##      FALSE, TRUE, FALSE, classif.ksvm
##      9.9750
## FALSE, TRUE, FALSE, classif.randomForest
##      7.2875
##      FALSE, TRUE, FALSE, classif.xgboost
##      6.2250
##      SMOTE, FALSE, FALSE, classif.ksvm
##      11.5625
## SMOTE, FALSE, FALSE, classif.randomForest
##      6.7500
##      SMOTE, FALSE, FALSE, classif.xgboost
##      4.3625
```

Plotando grafico de Critical Difference

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

