

R Notebook

Parametros:

```
Measure = Accuracy
Columns = sampling, weight_space, underbagging, learner
Performance = holdout_measure
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.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.:0.38633
## Median :0.9938 Median :0.99163 Median :0.76435
## Mean :0.9692 Mean :0.96680 Mean :0.67114
## 3rd Qu.:0.9990 3rd Qu.:0.99687 3rd Qu.:0.95470
## Max. :1.0000 Max. :1.00000 Max. :1.00000
## NA's :48 NA's :48 NA's :48
## 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 :48 (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 0.9677996
## 2 0.9819241
## 3 0.9875000
## 4 0.9937304
## 5 0.9900000
## 6 0.9929639
## ADASYN, FALSE, FALSE, classif.randomForest
## 1 0.9683959
## 2 NA
## 3 0.9887500
## 4 1.0000000
## 5 0.9916667
## 6 0.9991205
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## 1 0.9785331 0.9898629
## 2 0.9884180 0.9892883
## 3 0.9812500 0.9900000
## 4 0.9989551 0.9979101
## 5 0.9908333 0.9941667
## 6 0.9956025 0.9964820
## FALSE, FALSE, FALSE, classif.randomForest
## 1 0.9910555
## 2 NA
```

```

## 3          0.9900000
## 4          1.0000000
## 5          0.9916667
## 6          1.0000000
##  FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## 1          0.9910555          0.5086464
## 2          0.9918993          0.8815023
## 3          0.9900000          0.4887500
## 4          1.0000000          1.0000000
## 5          0.9925000          0.9900000
## 6          0.9982410          0.9947230
##  FALSE, FALSE, TRUE, classif.randomForest
## 1          0.4818128
## 2          0.7864364
## 3          0.8191667
## 4          0.9864159
## 5          0.8991667
## 6          0.9920844
##  FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## 1          0.4812165          0.9880739
## 2          0.8075919          0.9898239
## 3          0.8166667          0.9900000
## 4          0.9592476          0.9979101
## 5          0.9033333          0.9941667
## 6          0.9700967          0.9964820
##  FALSE, TRUE, FALSE, classif.randomForest
## 1          0.9910555
## 2          0.9914307
## 3          0.9900000
## 4          1.0000000
## 5          0.9925000
## 6          0.9991205
##  FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## 1          0.9910555          0.9666070
## 2          0.9914307          0.9822588
## 3          0.9900000          0.9879167
## 4          1.0000000          0.9958203
## 5          0.9950000          0.9908333
## 6          0.9982410          0.9929639
##  SMOTE, FALSE, FALSE, classif.randomForest
## 1          0.9737627
## 2          0.9841334
## 3          0.9870833
## 4          1.0000000
## 5          0.9916667
## 6          0.9991205
##  SMOTE, FALSE, FALSE, classif.xgboost
## 1          0.9779368
## 2          0.9894892
## 3          0.9816667
## 4          1.0000000
## 5          0.9933333
## 6          0.9973615

```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.9678
## 1st Qu.:0.9900
## Median :0.9909
## Mean :0.9903
## 3rd Qu.:0.9926
## Max. :0.9990
## NA's :2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.9684
## 1st Qu.:0.9887
## Median :0.9926
## Mean :0.9924
## 3rd Qu.:0.9990
## Max. :1.0000
## NA's :8
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.9767 Min. :0.9893
## 1st Qu.:0.9878 1st Qu.:0.9902
## Median :0.9939 Median :0.9916
## Mean :0.9919 Mean :0.9931
## 3rd Qu.:0.9990 3rd Qu.:0.9960
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, FALSE, classif.randomForest
## Min. :0.9883
## 1st Qu.:0.9911
## Median :0.9947
## Mean :0.9947
## 3rd Qu.:0.9991
## Max. :1.0000
## NA's :2
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min. :0.9850 Min. :0.1817
## 1st Qu.:0.9916 1st Qu.:0.7045
## Median :0.9943 Median :0.9807
## Mean :0.9945 Mean :0.8455
## 3rd Qu.:0.9981 3rd Qu.:0.9944
## Max. :1.0000 Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min. :0.4818
## 1st Qu.:0.7981
## Median :0.9246
## Mean :0.8825
## 3rd Qu.:0.9714
## Max. :1.0000
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min. :0.4812 Min. :0.9881
## 1st Qu.:0.8058 1st Qu.:0.9903
## Median :0.9091 Median :0.9917
```

```
## Mean      :0.8703                      Mean      :0.9931
## 3rd Qu.:0.9556                      3rd Qu.:0.9960
## Max.      :1.0000                      Max.      :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.9867
## 1st Qu.:0.9911
## Median :0.9933
## Mean      :0.9945
## 3rd Qu.:0.9988
## Max.      :1.0000
## NA's      :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.9867                      Min.      :0.9666
## 1st Qu.:0.9912                      1st Qu.:0.9900
## Median :0.9946                      Median :0.9910
## Mean      :0.9946                      Mean      :0.9907
## 3rd Qu.:0.9983                      3rd Qu.:0.9935
## Max.      :1.0000                      Max.      :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.9738
## 1st Qu.:0.9884
## Median :0.9953
## Mean      :0.9927
## 3rd Qu.:0.9989
## Max.      :1.0000
## NA's      :2
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.9617
## 1st Qu.:0.9878
## Median :0.9933
## Mean      :0.9915
## 3rd Qu.:0.9991
## Max.      :1.0000
##
```

Verificando a média de cada coluna selecionada

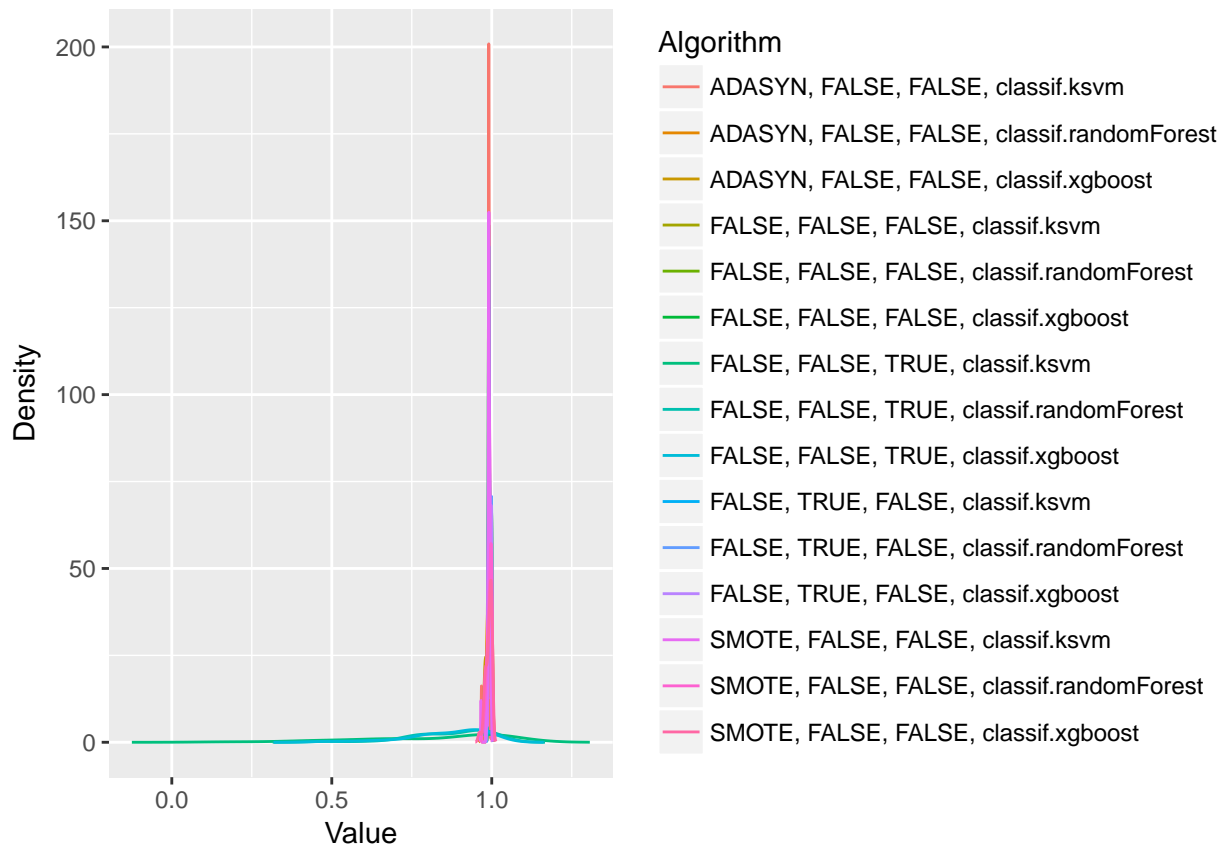
```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}
```

```
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.990333702309785"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.992435869799196"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.991900833888037"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.993062611875091"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.994689893128772"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.994517387753698"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.845512595070107"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.882542177532805"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.870291997455697"
```

```
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.993085687368942"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.994486656010044"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.99457547143972"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.99071256401121"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.992738670081132"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.991476571878267"
```

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 = 210.14, 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,]                                     FALSE
## [5,]                                     TRUE
## [6,]                                     TRUE
## [7,]                                     FALSE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [10,]                                    FALSE
## [11,]                                    TRUE
## [12,]                                    TRUE
## [13,]                                    FALSE
## [14,]                                    FALSE
## [15,]                                    FALSE
##      ADASYN, FALSE, FALSE, classif.randomForest
## [1,]                                     FALSE
## [2,]                                     FALSE
## [3,]                                     FALSE
## [4,]                                     FALSE
## [5,]                                     FALSE
## [6,]                                     FALSE
## [7,]                                     FALSE
## [8,]                                     TRUE
## [9,]                                     TRUE
## [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,]                                     TRUE
## [9,]                                     TRUE
## [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,]      TRUE
## [8,]      TRUE
## [9,]      TRUE
## [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,]      TRUE
## [9,]      TRUE
## [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,]      TRUE
## [9,]      TRUE
## [10,]     FALSE
## [11,]     FALSE
## [12,]     FALSE
## [13,]      TRUE
## [14,]     FALSE
## [15,]     FALSE
##      FALSE, FALSE, TRUE, classif.ksvm
## [1,]     FALSE
## [2,]     FALSE
## [3,]      TRUE
## [4,]      TRUE
## [5,]      TRUE

```

```

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

Plotando os ranks

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

```
##          ADASYN, FALSE, FALSE, classif.ksvm
##                      9.3625
## ADASYN, FALSE, FALSE, classif.randomForest
##                      8.2250
##          ADASYN, FALSE, FALSE, classif.xgboost
##                      7.0750
##          FALSE, FALSE, FALSE, classif.ksvm
##                      7.0500
## FALSE, FALSE, FALSE, classif.randomForest
##                      5.7875
##          FALSE, FALSE, FALSE, classif.xgboost
##                      4.8375
##          FALSE, FALSE, TRUE, classif.ksvm
##                      11.0500
## FALSE, FALSE, TRUE, classif.randomForest
##                      13.1000
##          FALSE, FALSE, TRUE, classif.xgboost
##                      13.4250
##          FALSE, TRUE, FALSE, classif.ksvm
##                      6.9750
## FALSE, TRUE, FALSE, classif.randomForest
##                      5.5875
##          FALSE, TRUE, FALSE, classif.xgboost
##                      4.8250
##          SMOTE, FALSE, FALSE, classif.ksvm
##                      9.2500
## SMOTE, FALSE, FALSE, classif.randomForest
##                      6.7125
##          SMOTE, FALSE, FALSE, classif.xgboost
##                      6.7375
```

Plotando grafico de Critical Difference

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

