

# R Notebook

## Parametros:

Measure = Matthews correlation coefficient  
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 : 0  FALSE :1080 FALSE:1440
## F1 measure          : 0  SMOTE : 360  TRUE :360
## G-mean              : 0              NA's :0
## Matthews correlation coefficient:1800
##
##
## tuning_measure      holdout_measure      holdout_measure_residual
## Min. : -0.00646      Min. : -0.1370      Min. : -0.06817
## 1st Qu.: 0.23511      1st Qu.: 0.0000      1st Qu.: 0.02210
## Median : 0.81997      Median : 0.3764      Median : 0.19355
## Mean : 0.64034      Mean : 0.4305      Mean : 0.29627
## 3rd Qu.: 0.99727      3rd Qu.: 0.8152      3rd Qu.: 0.49996
## Max. : 1.00000      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. :0.001
## 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 :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

summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. : -0.002393
## 1st Qu.: 0.000000
## Median : 0.058874
## Mean : 0.138420
## 3rd Qu.: 0.196052
## Max. : 0.851072
## NA's : 2
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. : -0.005847
## 1st Qu.: 0.064201
## Median : 0.263335
## Mean : 0.322226
## 3rd Qu.: 0.498946
## Max. : 0.950347
## NA's : 6
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. : -0.03745 Min. : -0.0008605
## 1st Qu.: 0.09532 1st Qu.: 0.0000000
## Median : 0.39846 Median : 0.0786579
## Mean : 0.38898 Mean : 0.1900848
## 3rd Qu.: 0.52592 3rd Qu.: 0.2433956
## Max. : 0.95928 Max. : 0.8585106
##
```

```

## FALSE, FALSE, FALSE, classif.randomForest
## Min.      :-0.01023
## 1st Qu.: 0.01831
## Median : 0.20011
## Mean    : 0.28128
## 3rd Qu.: 0.50264
## Max.    : 0.89881
## NA's    :1
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :-0.03888      Min.      :-0.003467
## 1st Qu.: 0.05539      1st Qu.: 0.095746
## Median : 0.25404      Median : 0.324663
## Mean    : 0.30227      Mean    : 0.316728
## 3rd Qu.: 0.48833      3rd Qu.: 0.525981
## Max.    : 0.90474      Max.    : 0.914968
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.006653
## 1st Qu.:0.151386
## Median :0.474659
## Mean    :0.458142
## 3rd Qu.:0.731017
## Max.    :0.916350
##
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.01815      Min.      :-0.0008605
## 1st Qu.:0.18818      1st Qu.: 0.0000000
## Median :0.46926      Median : 0.0758148
## Mean    :0.45167      Mean    : 0.1866649
## 3rd Qu.:0.67906      3rd Qu.: 0.2433956
## Max.    :0.89511      Max.    : 0.8585106
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :-0.02457
## 1st Qu.: 0.02669
## Median : 0.17019
## Mean    : 0.26935
## 3rd Qu.: 0.49242
## Max.    : 0.88606
## NA's    :2
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :-0.04521      Min.      :-0.002561
## 1st Qu.: 0.05402      1st Qu.: 0.001628
## Median : 0.24338      Median : 0.063074
## Mean    : 0.30232      Mean    : 0.142549
## 3rd Qu.: 0.49302      3rd Qu.: 0.178964
## Max.    : 0.89582      Max.    : 0.709685
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :-0.004377
## 1st Qu.: 0.065042
## Median : 0.226775
## Mean    : 0.296105
## 3rd Qu.: 0.482570

```

```
## Max.      : 0.938403
## NA's      :7
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :-0.06687
## 1st Qu.: 0.10196
## Median : 0.38049
## Mean      : 0.39147
## 3rd Qu.: 0.59879
## Max.      : 0.93815
##
```

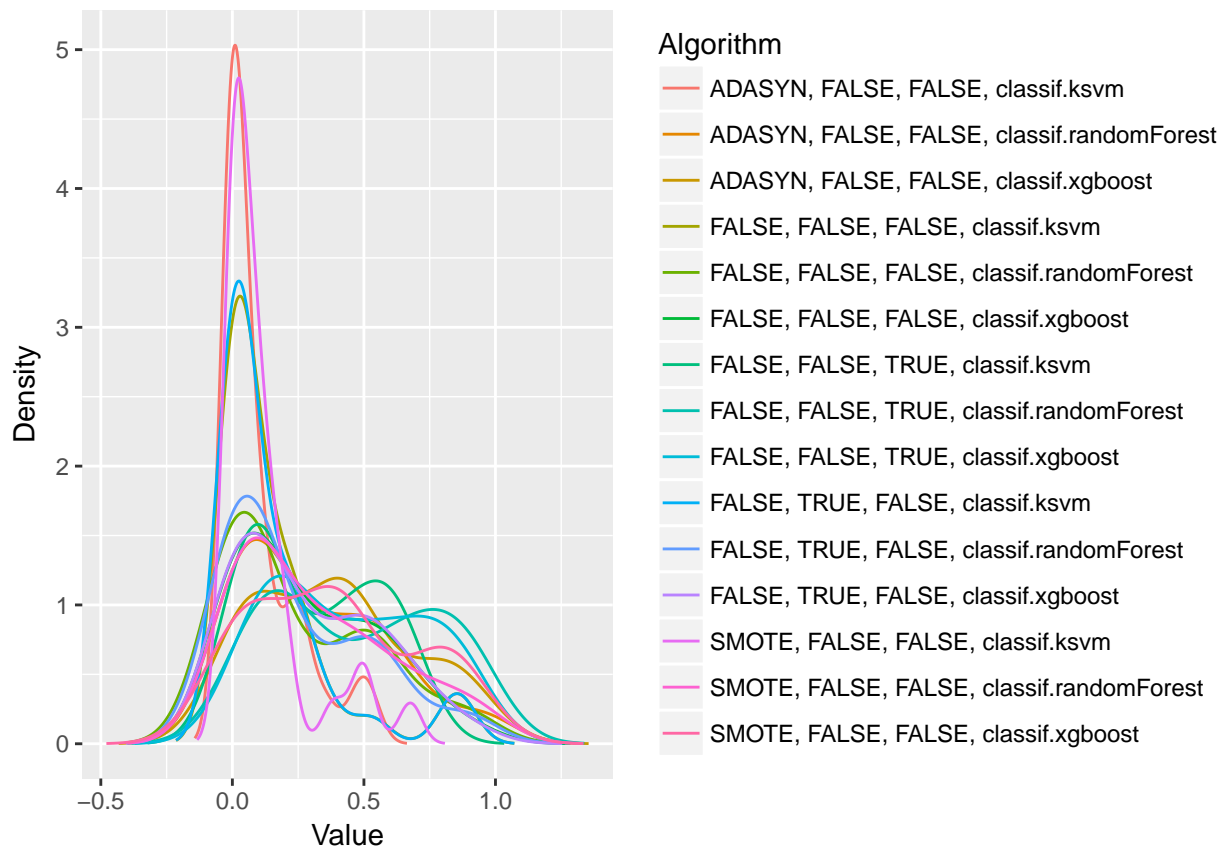
## Verificando a média de cada coluna selecionada

```
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 = 0.13841968187075"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.322225743960371"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.388975025643202"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.190084830988913"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.281281776315392"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.302270429895642"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.316727703199023"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.458142335331858"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.451670509813634"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.186664904350653"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.269347827594748"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.302317138155214"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.142549319028629"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.296104842944565"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.391465990700945"
```

## 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 = 177.5, 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,]                                TRUE
```

```

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

```

```

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

```



```

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

```

## Plotando grafico de Critical Difference

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

