

# R Notebook

## Parametros:

```
Measure = F1 measure
Columns = sampling, weight_space, underbagging, learner
Performance = holdout_measure
Filter keys = NULL
Filter values = NULL
```

```
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)){
  ds = filter_at(ds, .vars = params$filter_keys, .vars_predicate = any_vars(. == params$filter_values))
}
```

```
summary(ds)
```

```
##          learner      weight_space
## classif.ksvm      :3420  Mode :logical
## classif.randomForest:3420 FALSE:8208
## classif.rusboost   : 0    TRUE :2052
## classif.xgboost    :3420  NA's :0
##
##
##
##          measure      sampling  underbagging
## Accuracy              : 0    ADASYN:2052  Mode :logical
## Area under the curve  : 0    FALSE :6156  FALSE:8208
## F1 measure            :10260 SMOTE :2052  TRUE :2052
## 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.:0.2739 1st Qu.:0.0000  1st Qu.:0.04287
## Median :0.8197 Median :0.4500  Median :0.28466
## Mean :0.6468  Mean :0.4554  Mean :0.36600
## 3rd Qu.:0.9944 3rd Qu.:0.8075  3rd Qu.:0.68235
## Max. :1.0000  Max. :1.0000  Max. :1.00000
## NA's :216    NA's :216    NA's :216
## iteration_count      dataset      imba.rate
## Min. :1      abalone      : 180  Min. :0.0010
## 1st Qu.:1      adult      : 180  1st Qu.:0.0100
## Median :2      bank      : 180  Median :0.0300
## Mean :2      car      : 180  Mean :0.0286
## 3rd Qu.:3      cardiocography-10clases: 180 3rd Qu.:0.0500
## Max. :3      cardiocography-3clases : 180 Max. :0.0500
## NA's :216    (Other) :9180
```

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] 228 15
```

```
# Renomeando a variavel
df = df_tec_wide_residual

summary(df)
```

```
## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.0000
## 1st Qu.:0.0000
## Median :0.1667
## Mean :0.2736
## 3rd Qu.:0.4333
## Max. :1.0000
## NA's :7
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.0000
## 1st Qu.:0.2222
## Median :0.6389
## Mean :0.5660
## 3rd Qu.:0.9048
## Max. :1.0000
## NA's :27
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.0000
## 1st Qu.:0.3175 1st Qu.:0.0000
## Median :0.6986 Median :0.2056
## Mean :0.6061 Mean :0.3220
## 3rd Qu.:0.9229 3rd Qu.:0.5938
## Max. :1.0000 Max. :1.0000
##
```

```

## FALSE, FALSE, FALSE, classif.randomForest
## Min.      :0.0000
## 1st Qu.:0.1141
## Median :0.6084
## Mean    :0.5317
## 3rd Qu.:0.8876
## Max.    :1.0000
## NA's    :5
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :0.0000      Min.      :0.0000
## 1st Qu.:0.1667      1st Qu.:0.1030
## Median :0.6556      Median :0.2919
## Mean    :0.5488      Mean    :0.3695
## 3rd Qu.:0.8889      3rd Qu.:0.5942
## Max.    :1.0000      Max.    :1.0000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.02136
## 1st Qu.:0.14917
## Median :0.34326
## Mean    :0.40804
## 3rd Qu.:0.65516
## Max.    :1.00000
## NA's    :6
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.01227      Min.      :0.0000
## 1st Qu.:0.13304      1st Qu.:0.0000
## Median :0.30011      Median :0.1667
## Mean    :0.36805      Mean    :0.3133
## 3rd Qu.:0.54753      3rd Qu.:0.5853
## Max.    :1.00000      Max.    :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.0000
## 1st Qu.:0.1111
## Median :0.6333
## Mean    :0.5317
## 3rd Qu.:0.8807
## Max.    :1.0000
## NA's    :7
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.0000      Min.      :0.0000
## 1st Qu.:0.1667      1st Qu.:0.0000
## Median :0.6413      Median :0.1667
## Mean    :0.5505      Mean    :0.2817
## 3rd Qu.:0.8865      3rd Qu.:0.5273
## Max.    :1.0000      Max.    :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.0000
## 1st Qu.:0.1795
## Median :0.6374
## Mean    :0.5681
## 3rd Qu.:0.9278

```

```
## Max.      :1.0000
## NA's      :20
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.0000
## 1st Qu.:0.3200
## Median :0.6956
## Mean      :0.6126
## 3rd Qu.:0.9324
## Max.      :1.0000
##
```

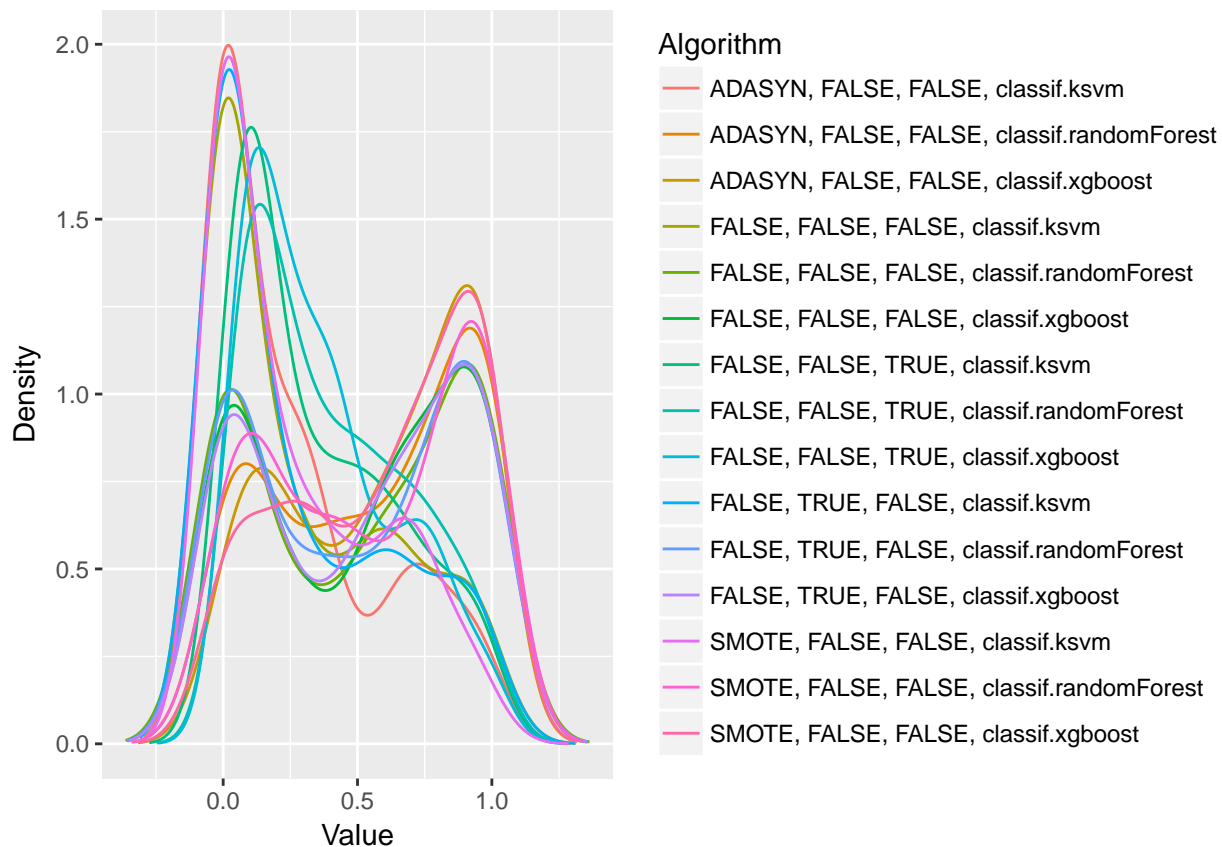
## Verificando a média de cada coluna selecionada

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

## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.ksvm = 0.273565164999428"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.566003047237143"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.606083540351536"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.322029397181557"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.531706972216012"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.548797217471997"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.369515673127528"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.408044694894222"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.368052388403102"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.313261523689142"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.531659101661382"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.550509756610902"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.281747266122444"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.568142671435832"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.61263631673472"
```

## 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 = 774.82, 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,] TRUE
## [12,] TRUE
## [13,] FALSE
## [14,] TRUE
## [15,] TRUE
## ADASYN, FALSE, FALSE, classif.randomForest
## [1,] TRUE
## [2,] FALSE
## [3,] TRUE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] FALSE
## [12,] FALSE
## [13,] TRUE
## [14,] FALSE
## [15,] TRUE
## ADASYN, FALSE, FALSE, classif.xgboost
## [1,] TRUE
## [2,] TRUE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [13,] TRUE
## [14,] TRUE
## [15,] FALSE
## FALSE, 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,] TRUE
## [12,] TRUE
## [13,] FALSE

```

```

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

```

```

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

```

## Plotando grafico de Critical Difference

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

