

R Notebook

Parametros:

```
Measure = F1 measure
Columns = sampling, weight_space, ruspool, learner
Performance = holdout_measure_residual
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.
summary(ds)
```

```
##           learner      weight_space
## classif.ksvm      :17100  Mode :logical
## classif.randomForest:17100 FALSE:41040
## classif.xgboost    :17100  TRUE :10260
##                                     NA's :0
##
##
##
##           measure      sampling      ruspool
## 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.: 0.5924  1st Qu.: 0.3114  1st Qu.: 0.1648
## Median : 0.9624  Median : 0.8193  Median : 0.5192
## Mean    : 0.7570  Mean    : 0.6469  Mean    : 0.5099
## 3rd Qu.: 0.9965  3rd Qu.: 0.9879  3rd Qu.: 0.8636
## Max.    : 1.0000  Max.    : 1.0000  Max.    : 1.0000
## NA's    :1761    NA's    :1761    NA's    :1761
## 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      cardiotocography-10clases: 900  3rd Qu.:0.0500
## Max.    :3      cardiotocography-3clases : 900  Max.    :0.0500
```

```
## NA's :1761 (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.xgboost    :3420  TRUE :2052  
##                   NA's :0  
##  
##  
##  
##           measure      sampling      ruspool  
## 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. :0.0000  
## 1st Qu.:0.1667  1st Qu.:0.0000  1st Qu.:0.0187  
## Median :0.7363  Median :0.3704  Median :0.2328  
## Mean :0.5997  Mean :0.4199  Mean :0.3418  
## 3rd Qu.:0.9922  3rd Qu.:0.8000  3rd Qu.:0.6582  
## Max. :1.0000  Max. :1.0000  Max. :1.0000  
## NA's :354  NA's :354  NA's :354  
## 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  cardiotocography-10clases: 180  3rd Qu.:0.0500  
## Max. :3  cardiotocography-3clases : 180  Max. :0.0500  
## NA's :354  (Other) :9180
```

Computando as médias das iteracoes

```
ds = group_by(ds, learner , weight_space , measure , sampling , ruspool , 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. :0.000000
## 1st Qu.:0.002186
## Median :0.083784
## Mean :0.200389
## 3rd Qu.:0.314711
## Max. :0.989520
## NA's :7
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.00000
## 1st Qu.:0.09524
## Median :0.35745
## Mean :0.38910
## 3rd Qu.:0.59000
## Max. :0.99225
## NA's :35
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.0000 Min. :0.0000
## 1st Qu.:0.1114 1st Qu.:0.0000
## Median :0.3893 Median :0.1033
## Mean :0.4246 Mean :0.2038
## 3rd Qu.:0.7304 3rd Qu.:0.2686
## Max. :0.9975 Max. :0.9949
##
## FALSE, FALSE, FALSE, classif.randomForest

```

```

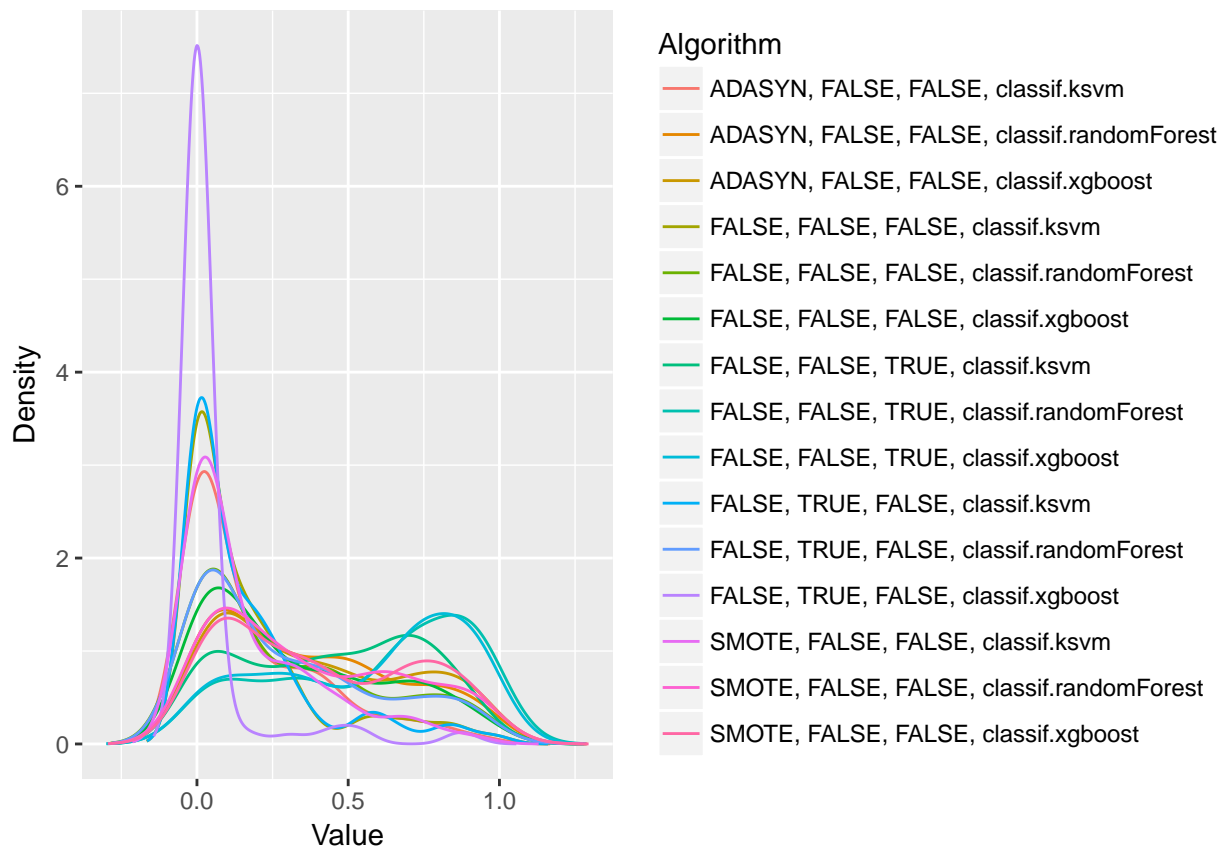
## Min.      :0.00000
## 1st Qu.:0.04393
## Median :0.25231
## Mean    :0.33493
## 3rd Qu.:0.52544
## Max.    :1.00000
## NA's    :10
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :0.00000      Min.      :0.0001813
## 1st Qu.:0.06354      1st Qu.:0.1869292
## Median :0.28825      Median :0.4789137
## Mean    :0.36556      Mean    :0.4627374
## 3rd Qu.:0.64321      3rd Qu.:0.7212287
## Max.    :0.99746      Max.    :0.9895300
##                      NA's      :3
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.003676
## 1st Qu.:0.298544
## Median :0.676779
## Mean    :0.584167
## 3rd Qu.:0.874417
## Max.    :0.983805
## NA's    :10
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.001115      Min.      :0.0000
## 1st Qu.:0.293284      1st Qu.:0.0000
## Median :0.689818      Median :0.0955
## Mean    :0.579065      Mean    :0.1956
## 3rd Qu.:0.852981      3rd Qu.:0.2581
## Max.    :0.982204      Max.    :0.9949
## NA's    :3
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.00000
## 1st Qu.:0.04256
## Median :0.22830
## Mean    :0.31667
## 3rd Qu.:0.51394
## Max.    :1.00000
## NA's    :18
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.00000      Min.      :0.000000
## 1st Qu.:0.00000      1st Qu.:0.006577
## Median :0.00000      Median :0.091957
## Mean    :0.07437      Mean    :0.193288
## 3rd Qu.:0.00000      3rd Qu.:0.307575
## Max.    :0.93236      Max.    :0.980870
##                      NA's      :2
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.0000
## 1st Qu.:0.1049
## Median :0.3198
## Mean    :0.3913
## 3rd Qu.:0.6552
## Max.    :0.9975

```

```
## NA's :30
## SMOTE, FALSE, FALSE, classif.xgboost
## Min. :0.0000
## 1st Qu.:0.1122
## Median :0.3939
## Mean :0.4329
## 3rd Qu.:0.7301
## Max. :1.0000
##
```

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 = 1116.5, 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,] 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,] FALSE
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] FALSE
## [12,] TRUE
## [13,] TRUE
## [14,] FALSE
## [15,] TRUE
##      ADASYN, FALSE, FALSE, classif.xgboost
## [1,] TRUE
## [2,] TRUE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] FALSE
## [8,] TRUE
## [9,] FALSE
## [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,]     TRUE
## [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,]     TRUE
## [13,]     TRUE
## [14,]     FALSE
## [15,]     TRUE
##      FALSE, FALSE, TRUE, classif.ksvm
## [1,]      TRUE
## [2,]     FALSE
## [3,]     FALSE
## [4,]      TRUE
## [5,]      TRUE

```

```

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

```

```
## [2,] TRUE
## [3,] FALSE
## [4,] TRUE
## [5,] TRUE
## [6,] TRUE
## [7,] FALSE
## [8,] TRUE
## [9,] FALSE
## [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) {})
```

