

R Notebook

Parametros:

```
Measure = Accuracy
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           :10260  ADASYN:2052  Mode :logical  
## Area under the curve : 0  FALSE :6156  FALSE:8208  
## F1 measure           : 0  SMOTE :2052  TRUE :2052  
## G-mean              : 0                   NA's :0  
## Matthews correlation coefficient: 0  
##  
##  
## tuning_measure  holdout_measure  holdout_measure_residual  
## Min. :0.0904  Min. :0.0152  Min. :0.0346  
## 1st Qu.:0.9591  1st Qu.:0.9535  1st Qu.:0.3643  
## Median :0.9861  Median :0.9800  Median :0.7162  
## Mean :0.9546  Mean :0.9493  Mean :0.6531  
## 3rd Qu.:0.9959  3rd Qu.:0.9925  3rd Qu.:0.9406  
## Max. :1.0000  Max. :1.0000  Max. :1.0000  
## NA's :348  NA's :348  NA's :348  
## 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 :348  (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
```

```

# Removendo linhas com NA's
df_tec_wide_residual = na.omit(df_tec_wide_residual)

# Renomeando a variavel
df = df_tec_wide_residual

summary(df)

```

```

## ADASYN, FALSE, FALSE, classif.ksvm
## Min. :0.03682
## 1st Qu.:0.32184
## Median :0.54310
## Mean :0.59393
## 3rd Qu.:0.92114
## Max. :0.99992
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.03934
## 1st Qu.:0.38759
## Median :0.72223
## Mean :0.65436
## 3rd Qu.:0.93150
## Max. :0.99987
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.04525 Min. :0.0367
## 1st Qu.:0.38110 1st Qu.:0.3088
## Median :0.75832 Median :0.5163
## Mean :0.66322 Mean :0.5840
## 3rd Qu.:0.93255 3rd Qu.:0.9141
## Max. :0.99986 Max. :0.9999
## FALSE, FALSE, FALSE, classif.randomForest

```

```

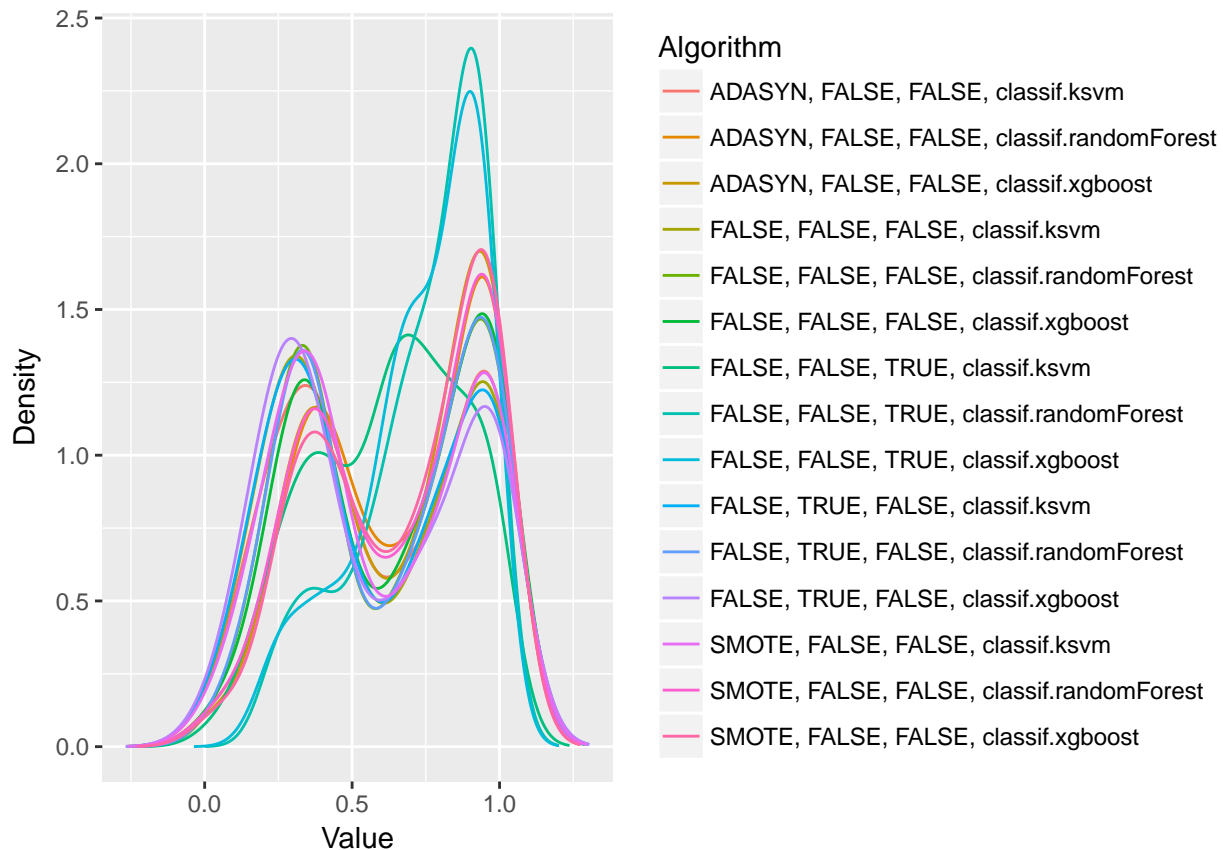
## Min.      :0.06542
## 1st Qu.:0.32815
## Median :0.69115
## Mean    :0.62419
## 3rd Qu.:0.92297
## Max.    :0.99987
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :0.03977      Min.      :0.04134
## 1st Qu.:0.34966      1st Qu.:0.42085
## Median :0.68244      Median :0.64928
## Mean    :0.63564      Mean    :0.62509
## 3rd Qu.:0.93225      3rd Qu.:0.82864
## Max.    :0.99986      Max.    :0.99744
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.2038
## 1st Qu.:0.6251
## Median :0.7916
## Mean    :0.7446
## 3rd Qu.:0.9124
## Max.    :0.9998
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.1649      Min.      :0.0367
## 1st Qu.:0.6230      1st Qu.:0.3080
## Median :0.7768      Median :0.5163
## Mean    :0.7330      Mean    :0.5817
## 3rd Qu.:0.9078      3rd Qu.:0.9037
## Max.    :0.9998      Max.    :0.9999
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.06468
## 1st Qu.:0.32789
## Median :0.67564
## Mean    :0.62449
## 3rd Qu.:0.92412
## Max.    :0.99987
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.0367      Min.      :0.03682
## 1st Qu.:0.2964      1st Qu.:0.32059
## Median :0.4486      Median :0.49899
## Mean    :0.5661      Mean    :0.59129
## 3rd Qu.:0.9090      3rd Qu.:0.92157
## Max.    :1.0000      Max.    :0.99992
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.04019
## 1st Qu.:0.38423
## Median :0.72092
## Mean    :0.65525
## 3rd Qu.:0.93081
## Max.    :0.99992
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.04523
## 1st Qu.:0.40893
## Median :0.75410
## Mean    :0.67061
## 3rd Qu.:0.93504

```

```
## Max. :0.99986
```

Fazendo teste de normalidade

```
plotDensities(data = df)
```



Testando as diferencas

```
friedmanTest(df)
```

```
##  
## Friedman's rank sum test  
##  
## data: df  
## Friedman's chi-squared = 344.4, 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,]      FALSE
## [6,]      TRUE
## [7,]      FALSE
## [8,]      TRUE
## [9,]      TRUE
## [10,]     FALSE
## [11,]     FALSE
## [12,]     TRUE
## [13,]     FALSE
## [14,]     TRUE
## [15,]     TRUE
##      ADASYN, FALSE, FALSE, classif.randomForest
## [1,]      TRUE
## [2,]     FALSE
## [3,]     FALSE
## [4,]      TRUE
## [5,]     FALSE
## [6,]     FALSE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]     TRUE
## [11,]     FALSE
## [12,]     TRUE
## [13,]     TRUE
## [14,]     FALSE
## [15,]     FALSE
##      ADASYN, FALSE, FALSE, classif.xgboost
## [1,]      TRUE
## [2,]     FALSE
## [3,]     FALSE
## [4,]      TRUE
## [5,]      TRUE
## [6,]     FALSE
## [7,]     FALSE
## [8,]     FALSE
## [9,]     FALSE
## [10,]     TRUE
## [11,]     TRUE
## [12,]     TRUE
## [13,]     TRUE
## [14,]     FALSE
## [15,]     FALSE
##      FALSE, FALSE, FALSE, classif.ksvm
## [1,]     FALSE
## [2,]      TRUE
## [3,]      TRUE
## [4,]     FALSE
## [5,]     FALSE

```

```

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

```

```

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

```

```
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] TRUE
## [12,] TRUE
## [13,] TRUE
## [14,] FALSE
## [15,] FALSE
```

Plotando grafico de Critical Difference

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

