

R Notebook

Parametros:

Measure = Matthews correlation coefficient
Columns = sampling, weight_space, ruspool, 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.  
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           : 0 SMOTE :2052  TRUE :2052  
## G-mean              : 0           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.: 0.2088  1st Qu.: 0.0000  1st Qu.: 0.0067  
## Median : 0.7306  Median : 0.4258  Median : 0.1812  
## Mean   : 0.6065  Mean   : 0.4333  Mean   : 0.2883  
## 3rd Qu.: 0.9851  3rd Qu.: 0.8098  3rd Qu.: 0.5031  
## Max.   : 1.0000  Max.   : 1.0000  Max.   : 1.0000  
## NA's   :357     NA's   :357     NA's   :357  
## 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   :357     (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. : -0.06657
## 1st Qu.: 0.00000
## Median : 0.16486
## Mean : 0.26331
## 3rd Qu.: 0.45554
## Max. : 1.00000
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. : -0.05198
## 1st Qu.: 0.24137
## Median : 0.62092
## Mean : 0.56061
## 3rd Qu.: 0.90215
## Max. : 1.00000
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. : -0.06053 Min. : -0.04044
## 1st Qu.: 0.22672 1st Qu.: 0.00000
## Median : 0.65888 Median : 0.17808
## Mean : 0.56227 Mean : 0.28875
## 3rd Qu.: 0.87401 3rd Qu.: 0.58482
## Max. : 1.00000 Max. : 1.00000
## FALSE, FALSE, FALSE, classif.randomForest

```

```

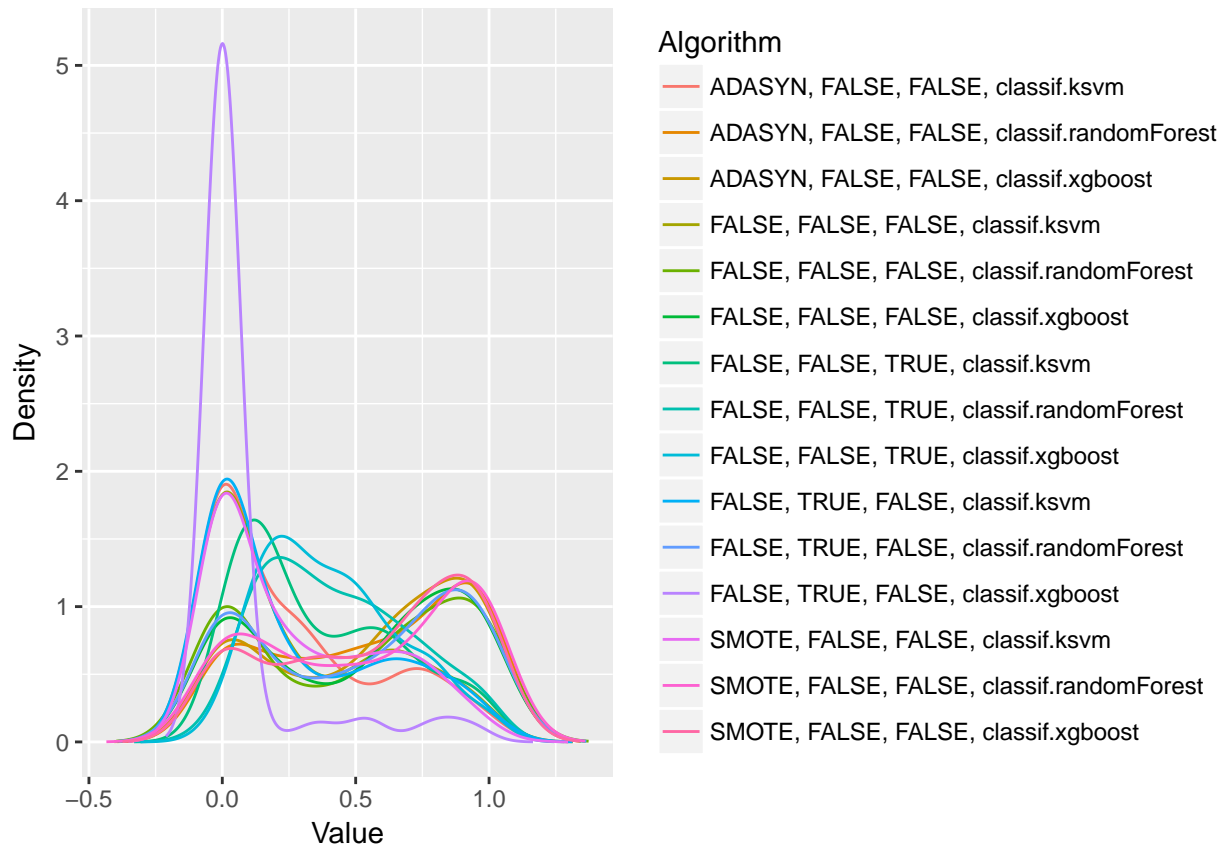
## Min.      :-0.02449
## 1st Qu.: 0.00000
## Median : 0.59999
## Mean    : 0.50730
## 3rd Qu.: 0.87157
## Max.    : 1.00000
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :-0.03192      Min.      :-0.0266
## 1st Qu.: 0.14750      1st Qu.: 0.1054
## Median : 0.63561      Median : 0.2688
## Mean    : 0.52013      Mean    : 0.3635
## 3rd Qu.: 0.85208      3rd Qu.: 0.5929
## Max.    : 1.00000      Max.    : 1.0000
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :-0.06927
## 1st Qu.: 0.20115
## Median : 0.39150
## Mean    : 0.43410
## 3rd Qu.: 0.64319
## Max.    : 1.00000
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :-0.06218      Min.      :-0.03836
## 1st Qu.: 0.21061      1st Qu.: 0.00000
## Median : 0.37085      Median : 0.14854
## Mean    : 0.41147      Mean    : 0.27612
## 3rd Qu.: 0.58944      3rd Qu.: 0.57554
## Max.    : 1.00000      Max.    : 1.00000
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :-0.02178
## 1st Qu.: 0.07913
## Median : 0.58418
## Mean    : 0.51379
## 3rd Qu.: 0.85784
## Max.    : 1.00000
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :-0.007648      Min.      :-0.05605
## 1st Qu.: 0.000000      1st Qu.: 0.00000
## Median : 0.000000      Median : 0.17796
## Mean    : 0.071685      Mean    : 0.28043
## 3rd Qu.: 0.000000      3rd Qu.: 0.53988
## Max.    : 0.960829      Max.    : 1.00000
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :-0.0748
## 1st Qu.: 0.1844
## Median : 0.6150
## Mean    : 0.5485
## 3rd Qu.: 0.9205
## Max.    : 1.0000
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :-0.03402
## 1st Qu.: 0.27250
## Median : 0.66493
## Mean    : 0.56621
## 3rd Qu.: 0.88195

```

```
## Max. : 1.00000
```

Fazendo teste de normalidade

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



Testando as diferencas

```
friedmanTest(df)
```

```
##  
## Friedman's rank sum test  
##  
## data: df  
## Friedman's chi-squared = 663.81, 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,]     FALSE
## [13,]     FALSE
## [14,]     TRUE
## [15,]     TRUE
##      ADASYN, FALSE, FALSE, classif.randomForest
## [1,]      TRUE
## [2,]     FALSE
## [3,]     FALSE
## [4,]      TRUE
## [5,]     FALSE
## [6,]     FALSE
## [7,]      TRUE
## [8,]      TRUE
## [9,]      TRUE
## [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,]     FALSE
## [6,]     FALSE
## [7,]      TRUE
## [8,]      TRUE
## [9,]      TRUE
## [10,]     TRUE
## [11,]     FALSE
## [12,]     TRUE
## [13,]     TRUE
## [14,]     FALSE
## [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,] FALSE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] FALSE
## [12,] TRUE
## [13,] TRUE
## [14,] FALSE
## [15,] FALSE
## FALSE, FALSE, FALSE, classif.xgboost
## [1,] TRUE
## [2,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] FALSE
## [12,] TRUE
## [13,] TRUE
## [14,] FALSE
## [15,] FALSE
## 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,] FALSE
## [6,] FALSE
## [7,] FALSE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] FALSE
## [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,] FALSE TRUE
## [6,] FALSE TRUE
## [7,] FALSE TRUE
## [8,] FALSE TRUE
## [9,] FALSE TRUE
## [10,] TRUE FALSE
## [11,] FALSE TRUE
## [12,] TRUE TRUE
## [13,] TRUE FALSE
## [14,] TRUE TRUE
## [15,] TRUE TRUE
## FALSE, TRUE, FALSE, classif.randomForest
## [1,] TRUE
## [2,] FALSE
## [3,] FALSE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] FALSE
## [9,] FALSE
## [10,] TRUE
## [11,] FALSE
## [12,] TRUE
## [13,] TRUE
## [14,] FALSE
## [15,] FALSE
## FALSE, TRUE, FALSE, classif.xgboost
## [1,] FALSE

```



```

## [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,] FALSE
## [4,] TRUE
## [5,] FALSE
## [6,] FALSE
## [7,] TRUE
## [8,] TRUE
## [9,] TRUE
## [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,] FALSE
## [6,] FALSE
## [7,] TRUE

```

```
## [8,] TRUE
## [9,] TRUE
## [10,] TRUE
## [11,] FALSE
## [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) {})
```

