

R Notebook

Parametros:

Measure = Matthews correlation coefficient
Columns = sampling, weight_space, underbagging, learner
Performance = tuning_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.  
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          :    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.   :-0.46576
## 1st Qu.: 0.3307  1st Qu.: 0.0000  1st Qu.: 0.03886
## Median : 0.8174  Median : 0.4907  Median : 0.21377
## Mean   : 0.6548  Mean   : 0.4657  Mean   : 0.30966
## 3rd Qu.: 0.9890  3rd Qu.: 0.8152  3rd Qu.: 0.53139
## Max.   : 1.0000  Max.   : 1.0000  Max.   : 1.00000
## NA's   :225     NA's   :225     NA's   :225
## 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    :225      (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. :0.5483
## 1st Qu.:0.9557
## Median :0.9899
## Mean :0.9620
## 3rd Qu.:0.9975
## Max. :1.0000
## NA's :7
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.4102
## 1st Qu.:0.9716
## Median :0.9934
## Mean :0.9723
## 3rd Qu.:0.9984
## Max. :1.0000
## NA's :25
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.4273 Min. : -0.003311
## 1st Qu.:0.9683 1st Qu.: 0.000000
## Median :0.9883 Median : 0.210174
## Mean :0.9665 Mean : 0.310733
## 3rd Qu.:0.9974 3rd Qu.: 0.565760
## Max. :1.0000 Max. : 1.000000
##
```

```

## FALSE, FALSE, FALSE, classif.randomForest
## Min.      :0.0000
## 1st Qu.:0.1832
## Median :0.5472
## Mean    :0.5307
## 3rd Qu.:0.8672
## Max.    :1.0000
## NA's    :6
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :-0.004855      Min.      :0.01683
## 1st Qu.: 0.242927      1st Qu.:0.15656
## Median : 0.643131      Median :0.39562
## Mean    : 0.570498      Mean    :0.41145
## 3rd Qu.: 0.855521      3rd Qu.:0.64835
## Max.    : 1.000000      Max.    :1.00000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.006784
## 1st Qu.:0.221089
## Median :0.395617
## Mean    :0.440705
## 3rd Qu.:0.637303
## Max.    :1.000000
## NA's    :6
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.004119      Min.      :-0.002739
## 1st Qu.:0.198170      1st Qu.: 0.000000
## Median :0.366367      Median : 0.199159
## Mean    :0.405539      Mean    : 0.304707
## 3rd Qu.:0.579288      3rd Qu.: 0.536926
## Max.    :1.000000      Max.    : 1.000000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :-0.002022
## 1st Qu.: 0.157423
## Median : 0.527450
## Mean    : 0.527645
## 3rd Qu.: 0.873868
## Max.    : 1.000000
## NA's    :11
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :-0.001833      Min.      :0.5423
## 1st Qu.: 0.245642      1st Qu.:0.9570
## Median : 0.633868      Median :0.9905
## Mean    : 0.570643      Mean    :0.9616
## 3rd Qu.: 0.859893      3rd Qu.:0.9988
## Max.    : 1.000000      Max.    :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.4552
## 1st Qu.:0.9708
## Median :0.9926
## Mean    :0.9729
## 3rd Qu.:0.9989

```

```
## Max.      :1.0000
## NA's      :20
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.3909
## 1st Qu.:0.9702
## Median :0.9899
## Mean      :0.9715
## 3rd Qu.:0.9974
## Max.      :1.0000
##
```

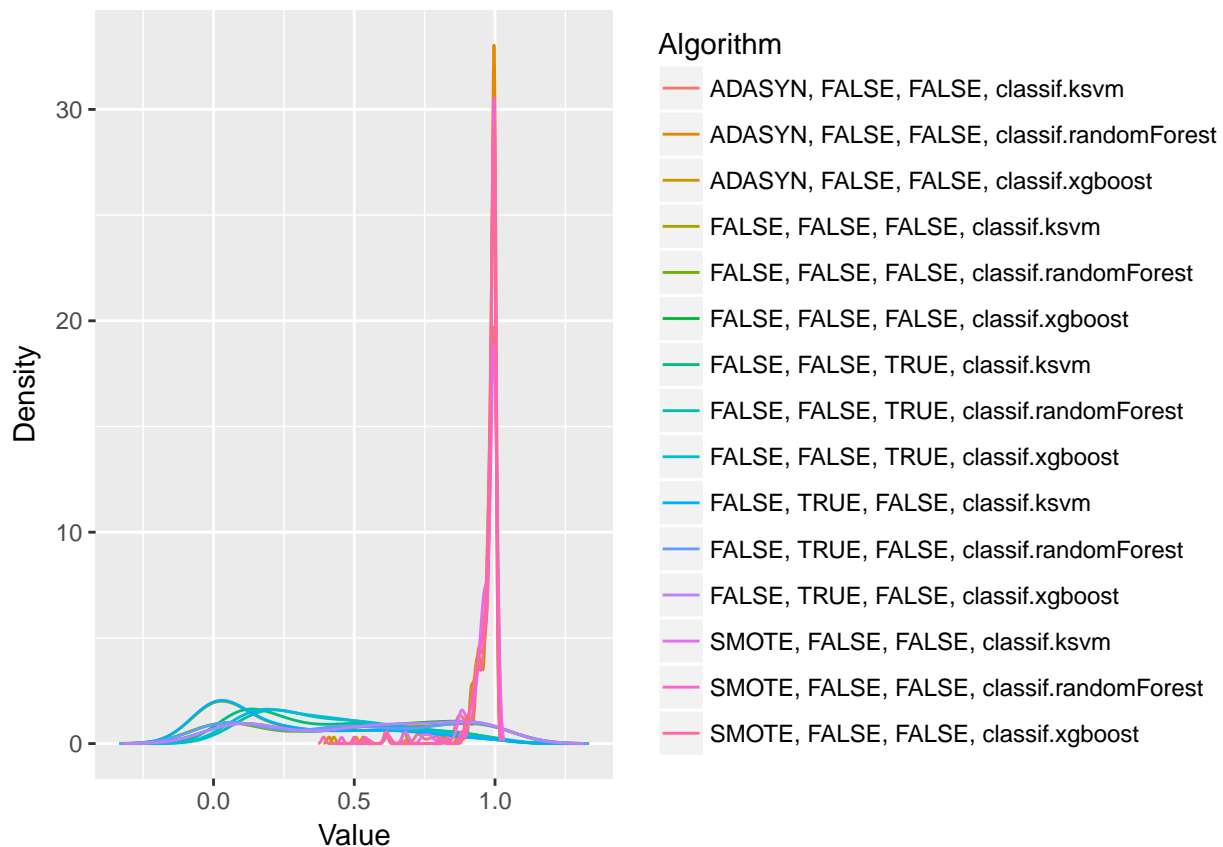
Verificando a média de cada coluna selecionada

```
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 = 0.961994214551935"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.972321766303781"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.966487678836997"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.310732475445047"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.530722387890025"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.570498273934176"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.411452317406682"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.440704888393628"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.405539345310321"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.304706805868436"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.5276454220385"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.570643543424285"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.961561133969507"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.972927544506485"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.971525555426833"
```

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 = 1939.3, 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,]                                FALSE
## [3,]                                FALSE
## [4,]                                TRUE
## [5,]                                TRUE
## [6,]                                TRUE
## [7,]                                TRUE
```

```

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

```

```

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

```



```

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

```

```

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

```

Plotando grafico de Critical Difference

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

