

R Notebook

Parametros:

```
Measure = Accuracy
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          :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. :0.09041  Min. :0.01517  Min. :0.0346
## 1st Qu.:0.96185 1st Qu.:0.95349 1st Qu.:0.3809
## Median :0.98796 Median :0.98113  Median :0.7239
## Mean :0.95509  Mean :0.94933  Mean :0.6600
## 3rd Qu.:0.99669 3rd Qu.:0.99347 3rd Qu.:0.9428
## Max. :1.00000  Max. :1.00000  Max. :1.0000
## NA's :204      NA's :204      NA's :204
## 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 :204      (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))

# Seleccionando 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. :0.5434
## 1st Qu.:0.9577
## Median :0.9801
## Mean :0.9665
## 3rd Qu.:0.9908
## Max. :1.0000
## NA's :7
## ADASYN, FALSE, FALSE, classif.randomForest
## Min. :0.5030
## 1st Qu.:0.9675
## Median :0.9866
## Mean :0.9707
## 3rd Qu.:0.9959
## Max. :1.0000
## NA's :26
## ADASYN, FALSE, FALSE, classif.xgboost FALSE, FALSE, FALSE, classif.ksvm
## Min. :0.5212 Min. :0.9228
## 1st Qu.:0.9687 1st Qu.:0.9655
## Median :0.9873 Median :0.9808
## Mean :0.9708 Mean :0.9772
## 3rd Qu.:0.9960 3rd Qu.:0.9916
## Max. :1.0000 Max. :1.0000
##
```

```

## FALSE, FALSE, FALSE, classif.randomForest
## Min.      :0.9333
## 1st Qu.:0.9735
## Median :0.9897
## Mean    :0.9836
## 3rd Qu.:0.9954
## Max.     :1.0000
## NA's     :6
## FALSE, FALSE, FALSE, classif.xgboost FALSE, FALSE, TRUE, classif.ksvm
## Min.      :0.9333      Min.      :0.05952
## 1st Qu.:0.9744      1st Qu.:0.67881
## Median :0.9900      Median :0.94300
## Mean     :0.9837      Mean    :0.81710
## 3rd Qu.:0.9954      3rd Qu.:0.98517
## Max.     :1.0000      Max.     :1.00000
##
## FALSE, FALSE, TRUE, classif.randomForest
## Min.      :0.3923
## 1st Qu.:0.7907
## Median :0.9070
## Mean     :0.8653
## 3rd Qu.:0.9680
## Max.     :1.0000
## NA's     :5
## FALSE, FALSE, TRUE, classif.xgboost FALSE, TRUE, FALSE, classif.ksvm
## Min.      :0.3274      Min.      :0.9333
## 1st Qu.:0.7912      1st Qu.:0.9646
## Median :0.9033      Median :0.9804
## Mean     :0.8551      Mean    :0.9773
## 3rd Qu.:0.9553      3rd Qu.:0.9916
## Max.     :1.0000      Max.     :1.0000
##
## FALSE, TRUE, FALSE, classif.randomForest
## Min.      :0.9333
## 1st Qu.:0.9734
## Median :0.9900
## Mean     :0.9835
## 3rd Qu.:0.9961
## Max.     :1.0000
## NA's     :6
## FALSE, TRUE, FALSE, classif.xgboost SMOTE, FALSE, FALSE, classif.ksvm
## Min.      :0.9380      Min.      :0.5447
## 1st Qu.:0.9744      1st Qu.:0.9579
## Median :0.9898      Median :0.9791
## Mean     :0.9836      Mean    :0.9672
## 3rd Qu.:0.9958      3rd Qu.:0.9907
## Max.     :1.0000      Max.     :1.0000
##
## SMOTE, FALSE, FALSE, classif.randomForest
## Min.      :0.5333
## 1st Qu.:0.9668
## Median :0.9878
## Mean     :0.9714
## 3rd Qu.:0.9970

```

```
## Max.      :1.0000
## NA's      :18
## SMOTE, FALSE, FALSE, classif.xgboost
## Min.      :0.5152
## 1st Qu.:0.9667
## Median :0.9868
## Mean      :0.9717
## 3rd Qu.:0.9971
## Max.      :1.0000
##
```

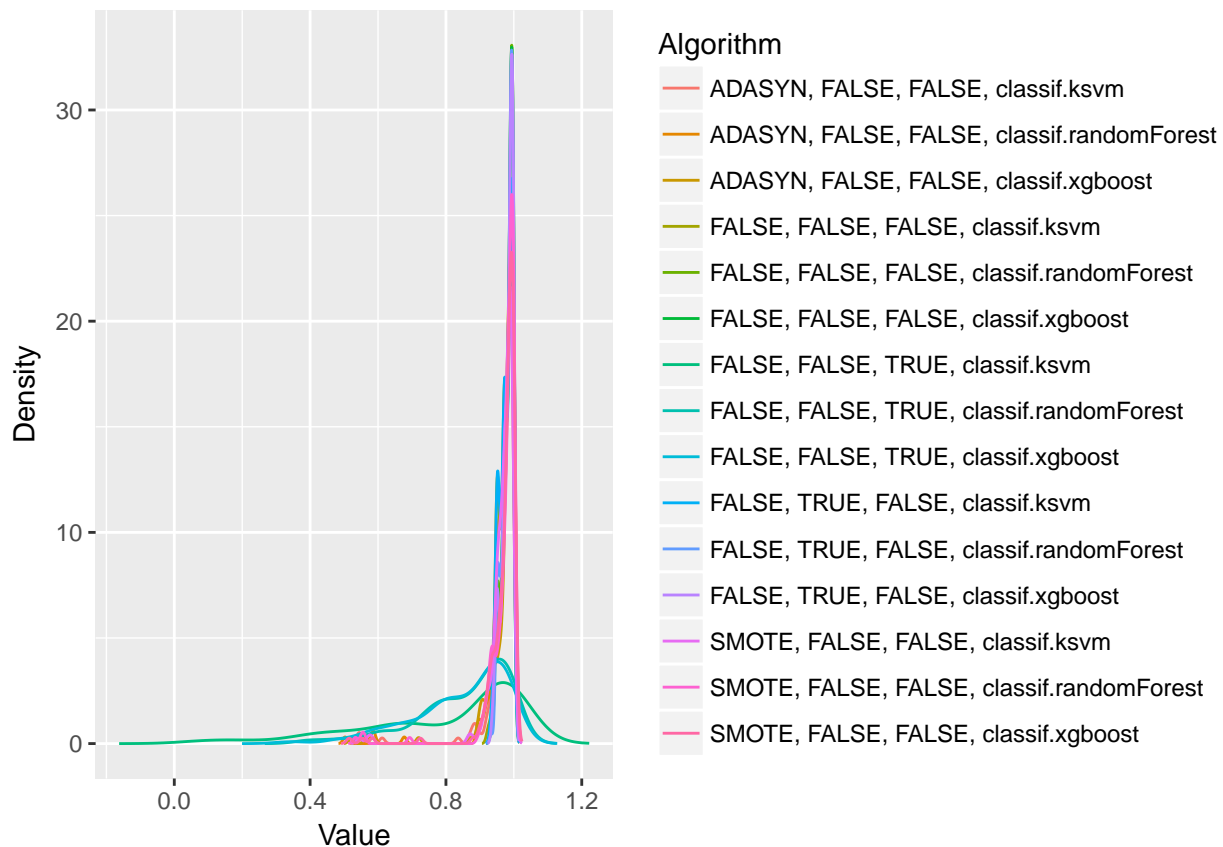
Verificando a média de cada coluna selecionada

```
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 = 0.966484094215529"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.randomForest = 0.970664035566509"
## [1] "Media da coluna ADASYN, FALSE, FALSE, classif.xgboost = 0.970789412950372"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.ksvm = 0.977243499205512"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.randomForest = 0.983614207207356"
## [1] "Media da coluna FALSE, FALSE, FALSE, classif.xgboost = 0.983670715189815"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.ksvm = 0.817099154557508"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.randomForest = 0.865321100923035"
## [1] "Media da coluna FALSE, FALSE, TRUE, classif.xgboost = 0.855110064686301"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.ksvm = 0.9772910564921"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.randomForest = 0.983491584295252"
## [1] "Media da coluna FALSE, TRUE, FALSE, classif.xgboost = 0.983551326917539"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.ksvm = 0.967214856359944"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.randomForest = 0.971349974972477"
## [1] "Media da coluna SMOTE, FALSE, FALSE, classif.xgboost = 0.971721812591086"
```

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 = 1282.7, 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,]                                TRUE
## [5,]                                TRUE
## [6,]                                TRUE
## [7,]                                TRUE
```

```

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

```

```

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

```

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

