

R Notebook

Parametros:

Measure = Area under the curve
Columns = sampling, weight_space, underbagging
Performance = holdout_measure_residual
Filter keys = imba.rate
Filter values = 0.05

```
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)){
  dots = paste0(params$filter_keys, " == '",params$filter_values, "'")
  ds = filter_(ds, .dots = dots)
}
```

```
summary(ds)
```

```
##          learner      weight_space
## classif.ksvm      :1230  Mode :logical
## classif.randomForest:1230 FALSE:2952
## classif.rusboost   :   0  TRUE :738
## classif.xgboost    :1230  NA's :0
##
##
##
##          measure      sampling  underbagging
## Accuracy          :   0  ADASYN: 738  Mode :logical
## Area under the curve :3690  FALSE :2214  FALSE:2952
## F1 measure          :   0  SMOTE : 738  TRUE :738
## G-mean              :   0              NA's :0
## Matthews correlation coefficient: 0
##
##
## tuning_measure  holdout_measure  holdout_measure_residual
## Min. :0.3977  Min. :0.0000  Min. :0.0000
## 1st Qu.:0.9145  1st Qu.:0.8175  1st Qu.:0.6976
## Median :0.9932  Median :0.9755  Median :0.8806
## Mean :0.9282  Mean :0.8846  Mean :0.8211
## 3rd Qu.:0.9997  3rd Qu.:0.9992  3rd Qu.:0.9784
## Max. :1.0000  Max. :1.0000  Max. :1.0000
## NA's :84      NA's :84      NA's :84
## iteration_count      dataset      imba.rate
## Min. :1      abalone      : 45  Min. :0.05
## 1st Qu.:1      adult        : 45  1st Qu.:0.05
## Median :2      annealing    : 45  Median :0.05
## Mean :2      arrhythmia   : 45  Mean :0.05
## 3rd Qu.:3      balance-scale: 45  3rd Qu.:0.05
## Max. :3      bank          : 45  Max. :0.05
## NA's :84      (Other)      :3420
```

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] 246 5
```

```
# Renomeando a variavel
df = df_tec_wide_residual

head(df)
```

```
## ADASYN, FALSE, FALSE FALSE, FALSE, FALSE FALSE, FALSE, TRUE
## 1 0.5948389 0.6479328 0.6628553
## 2 NA NA 0.8251103
## 3 0.8072758 0.8307921 0.7774846
## 4 0.5739972 0.5000000 0.6625173
## 5 0.8808254 0.8701104 0.7139014
## 6 0.7789270 0.7816640 0.7655071
## FALSE, TRUE, FALSE SMOTE, FALSE, FALSE
## 1 NA 0.5862406
## 2 NA NA
## 3 0.8307921 0.8195129
## 4 0.5000000 0.5276625
## 5 0.8701104 0.8969212
## 6 0.7816640 0.7748179
```

```
summary(df)
```

```
## ADASYN, FALSE, FALSE FALSE, FALSE, FALSE FALSE, FALSE, TRUE
## Min. :0.3506 Min. :0.3885 Min. :0.3653
## 1st Qu.:0.7146 1st Qu.:0.7080 1st Qu.:0.6728
## Median :0.8738 Median :0.8796 Median :0.8371
## Mean :0.8250 Mean :0.8266 Mean :0.8013
## 3rd Qu.:0.9773 3rd Qu.:0.9750 3rd Qu.:0.9566
## Max. :1.0000 Max. :1.0000 Max. :1.0000
```

```
## NA's :8          NA's :6          NA's :3
## FALSE, TRUE, FALSE SMOTE, FALSE, FALSE
## Min. :0.4130    Min. :0.3945
## 1st Qu.:0.7122    1st Qu.:0.7107
## Median :0.8726    Median :0.8667
## Mean :0.8254     Mean :0.8275
## 3rd Qu.:0.9761    3rd Qu.:0.9752
## Max. :1.0000     Max. :1.0000
## NA's :4          NA's :7
```

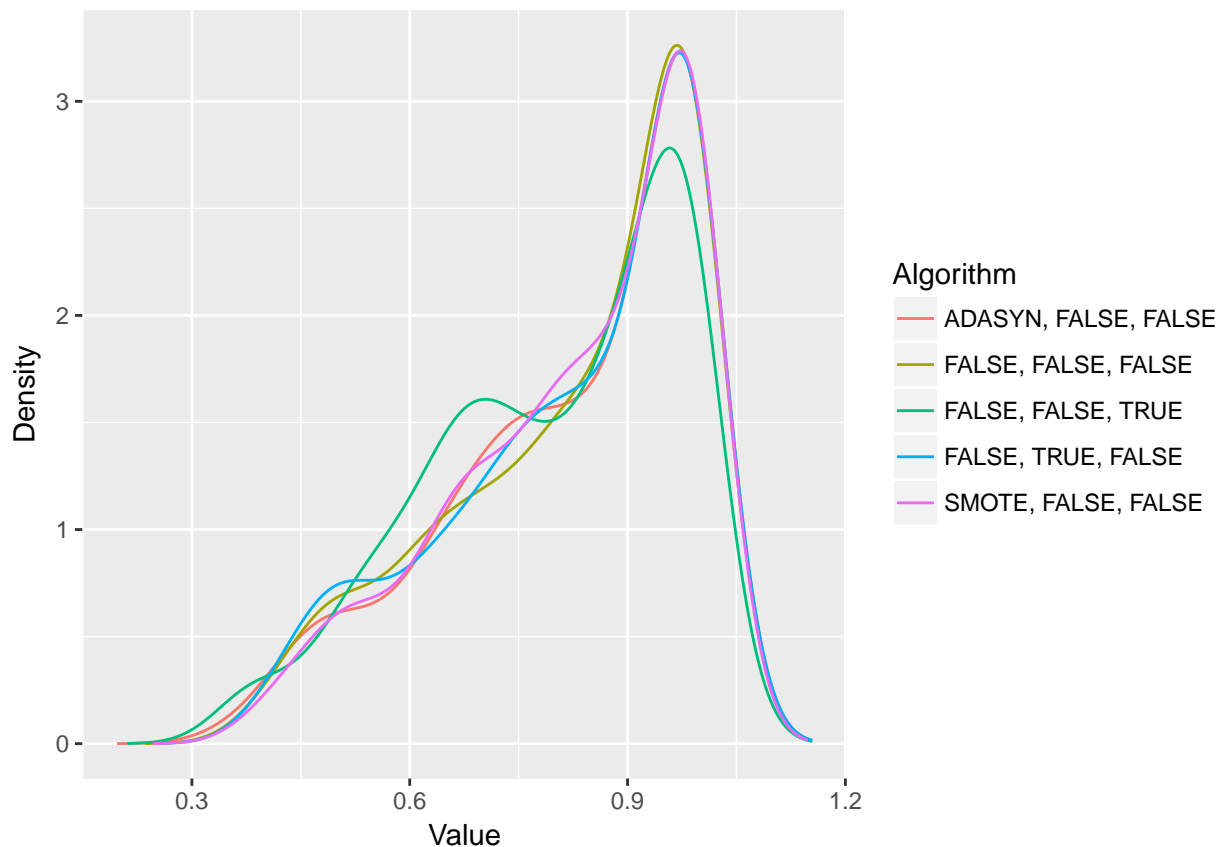
Verificando a média de cada coluna selecionada

```
for(i in (1:dim(df)[2])){
  print(paste("Media da coluna ", colnames(df)[i], " = ", mean(df[,i], na.rm = TRUE), sep=""))
}

## [1] "Media da coluna ADASYN, FALSE, FALSE = 0.825045335525325"
## [1] "Media da coluna FALSE, FALSE, FALSE = 0.826565590222471"
## [1] "Media da coluna FALSE, FALSE, TRUE = 0.801261015448162"
## [1] "Media da coluna FALSE, TRUE, FALSE = 0.825414328632424"
## [1] "Media da coluna SMOTE, FALSE, FALSE = 0.827450322953879"
```

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 = 50.181, df = 4, p-value = 3.309e-10
```

Testando as diferencas par a par

```
test <- nemenyiTest (df, alpha=0.05)
abs(test$diff.matrix) > test$statistic
```

```
##      ADASYN, FALSE, FALSE FALSE, FALSE, FALSE FALSE, FALSE, TRUE
## [1,]          FALSE          FALSE          TRUE
## [2,]          FALSE          FALSE          TRUE
## [3,]           TRUE          TRUE         FALSE
## [4,]          FALSE          FALSE          TRUE
## [5,]          FALSE          FALSE          TRUE
##      FALSE, TRUE, FALSE SMOTE, FALSE, FALSE
## [1,]          FALSE          FALSE
```

```
## [2,]          FALSE          FALSE
## [3,]           TRUE           TRUE
## [4,]          FALSE          FALSE
## [5,]          FALSE          FALSE
```

Plotando os ranks

```
print(colMeans(rankMatrix(df)))
```

```
## ADASYN, FALSE, FALSE  FALSE, FALSE, FALSE  FALSE, FALSE, TRUE
##           2.945122           2.853659           3.613821
##  FALSE, TRUE, FALSE  SMOTE, FALSE, FALSE
##           2.902439           2.684959
```

Plotando grafico de Critical Difference

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

