Sick dataset analysis

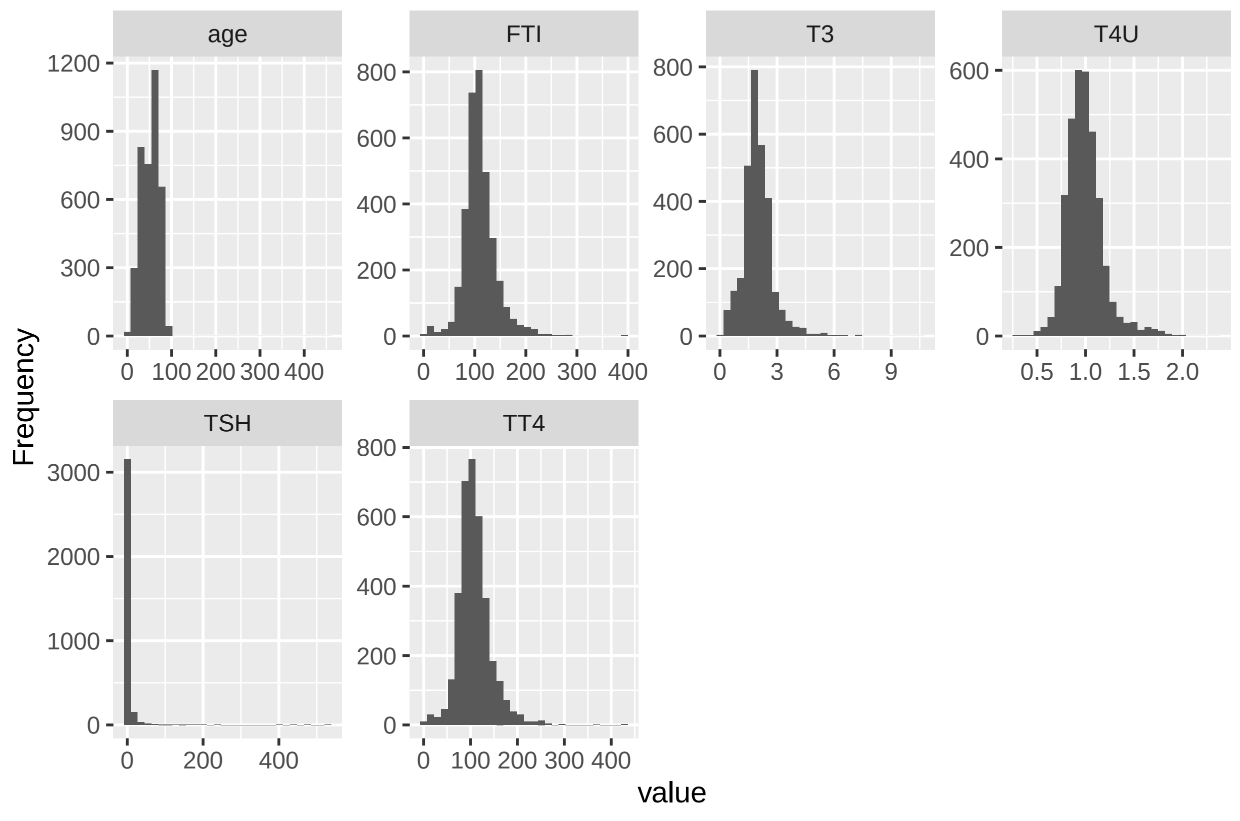
Bartłomiej Granat

29 04 2020

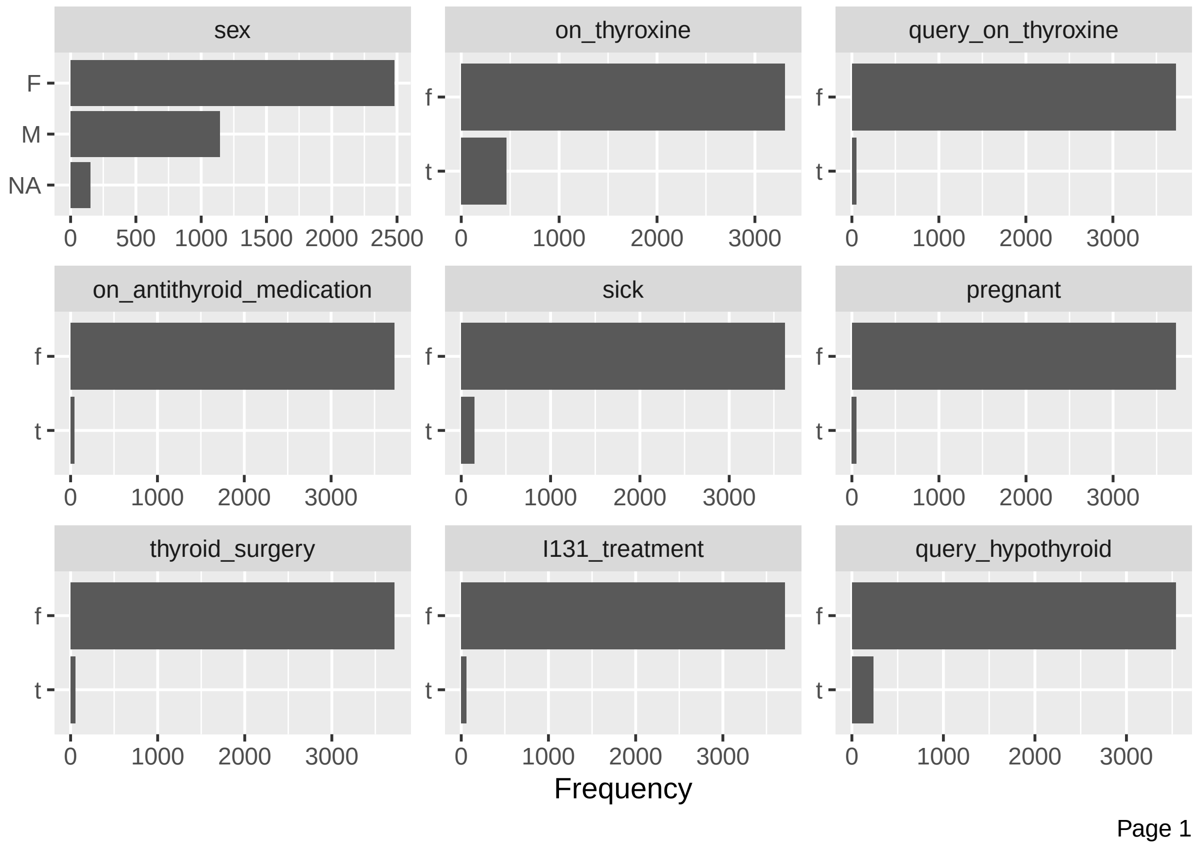
Intro

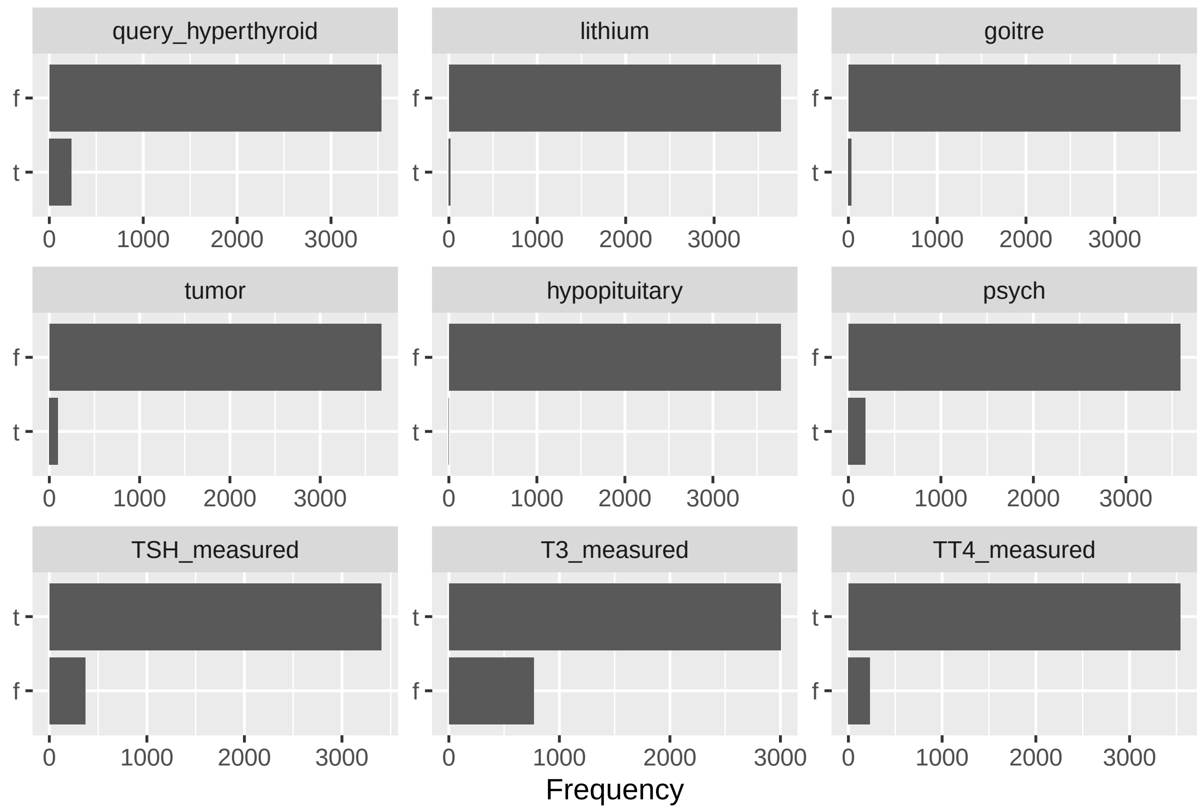
In this report I tried to keep everything simple. My data cleaning strategy was to delete all of the observations that might be problematic for my model as long as I am not deleting too many records of sick people. We are dealing with very unbalanced data and we can't afford to lose positive cases.

```
DataExplorer::plot_histogram(dataset_raw)
```

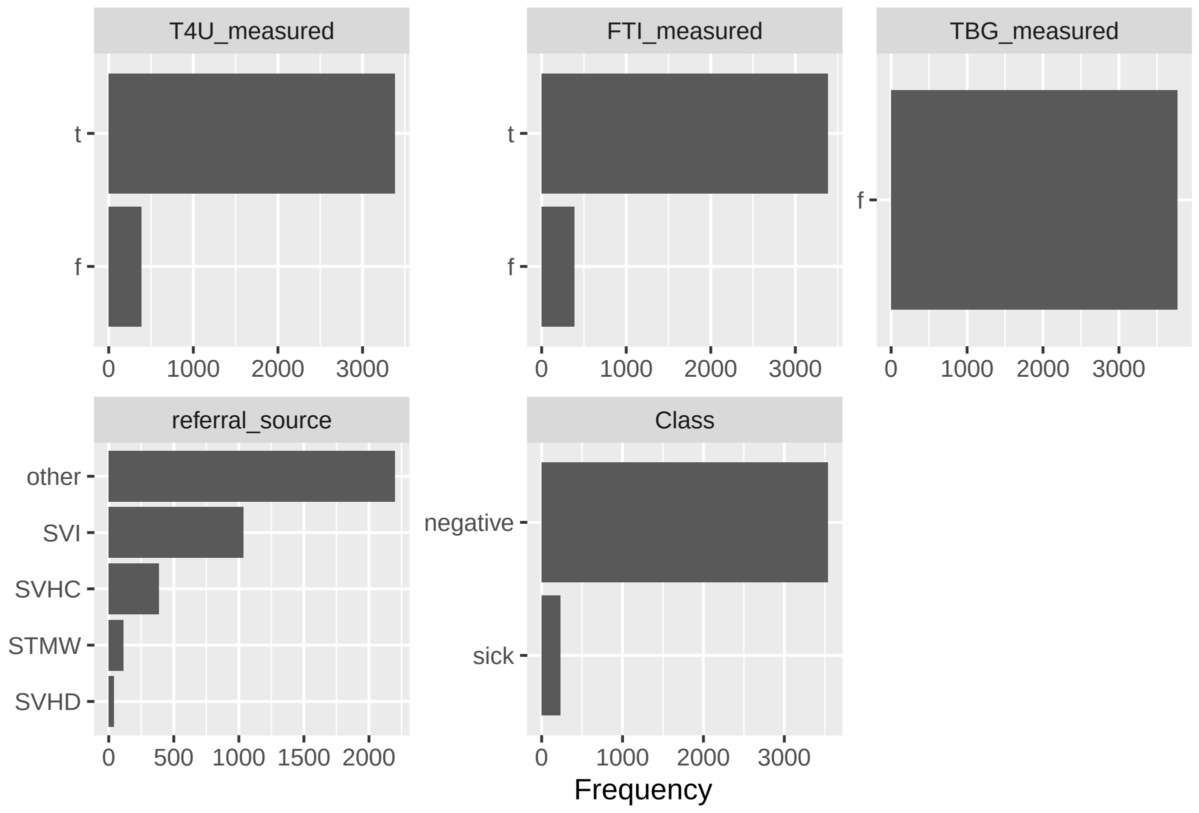


```r
DataExplorer::plot_bar(dataset_raw)
```




query_hyperthyroid
lithium
goitre
tumor
hypopituitary
psych
TSH_measured
T3_measured
TT4_measured
f
t
0
1000
2000
3000
Frequency

Page 2

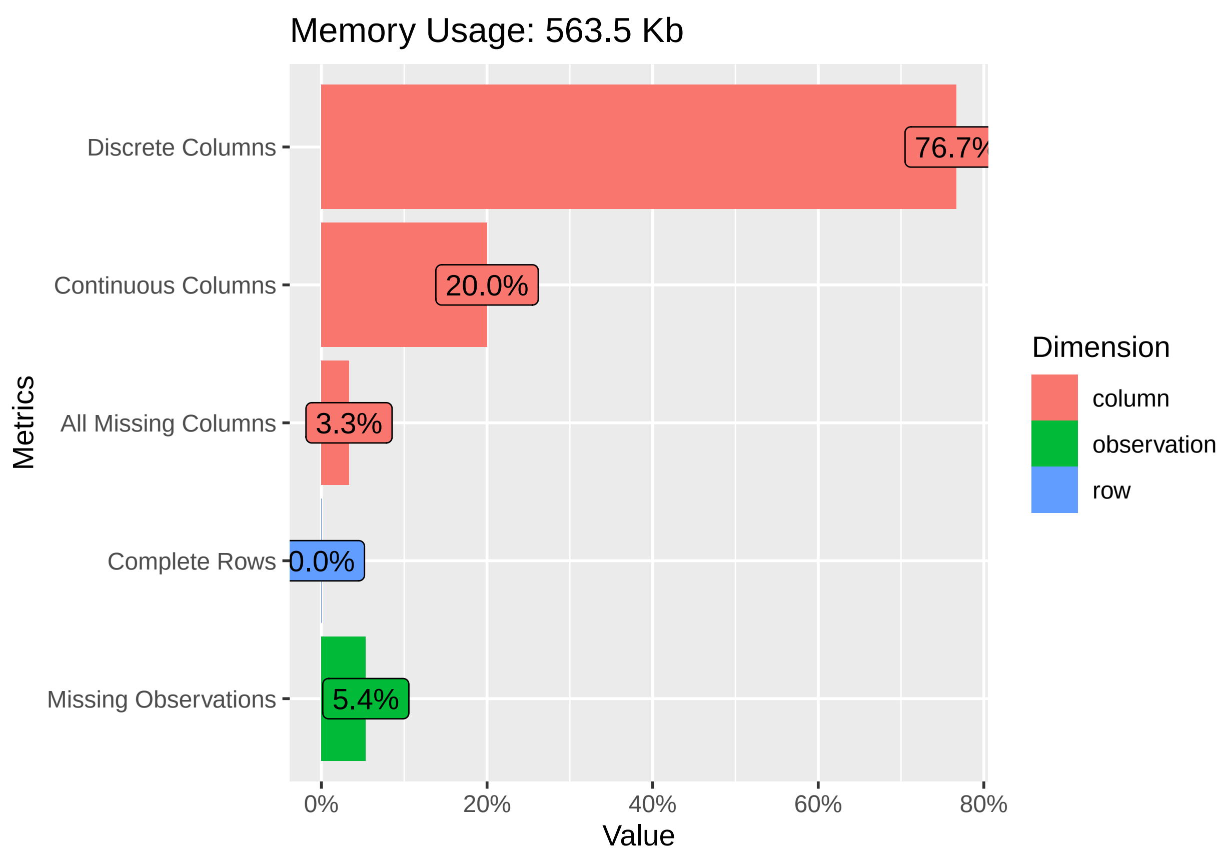


Page 3

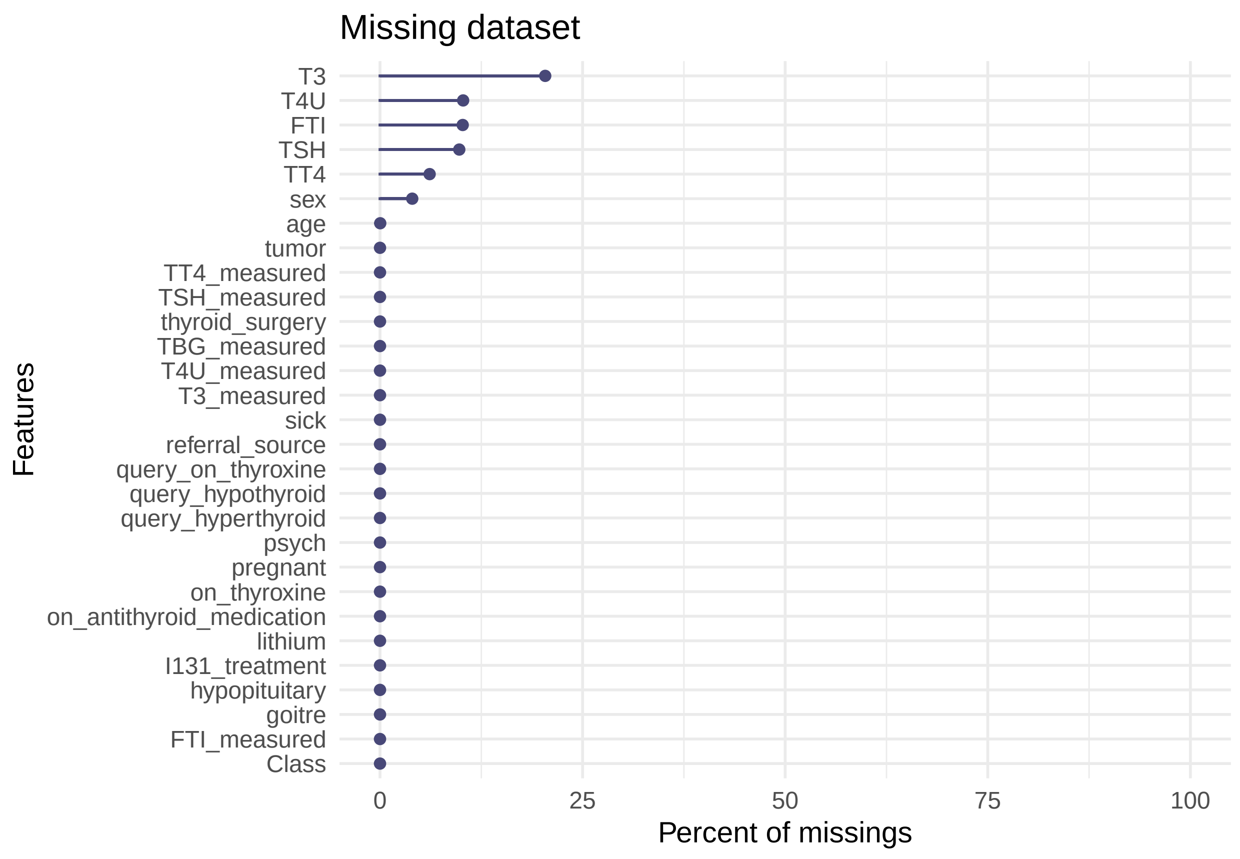
```
DataExplorer::introduce(dataset_raw)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 3772      30               23                  6                   1
##   total_missing_values complete_rows total_observations memory_usage
## 1                 6064             0             113160       577048
```

```
DataExplorer::plot_intro(dataset_raw)
```


Memory Usage: 563.5 Kb
Metrics
Value
Discrete Columns
Continuous Columns
All Missing Columns
Complete Rows
Missing Observations
76.7%
20.0%
3.3%
0.0%
5.4%
0%
20%
40%
60%
80%
Dimension
column
observation
row

Preprocessing



I want to delete all rows with missing data but also to control the number of positive cases in my data.

I also want to delete rows with some unrealistic values like 'age' above 400

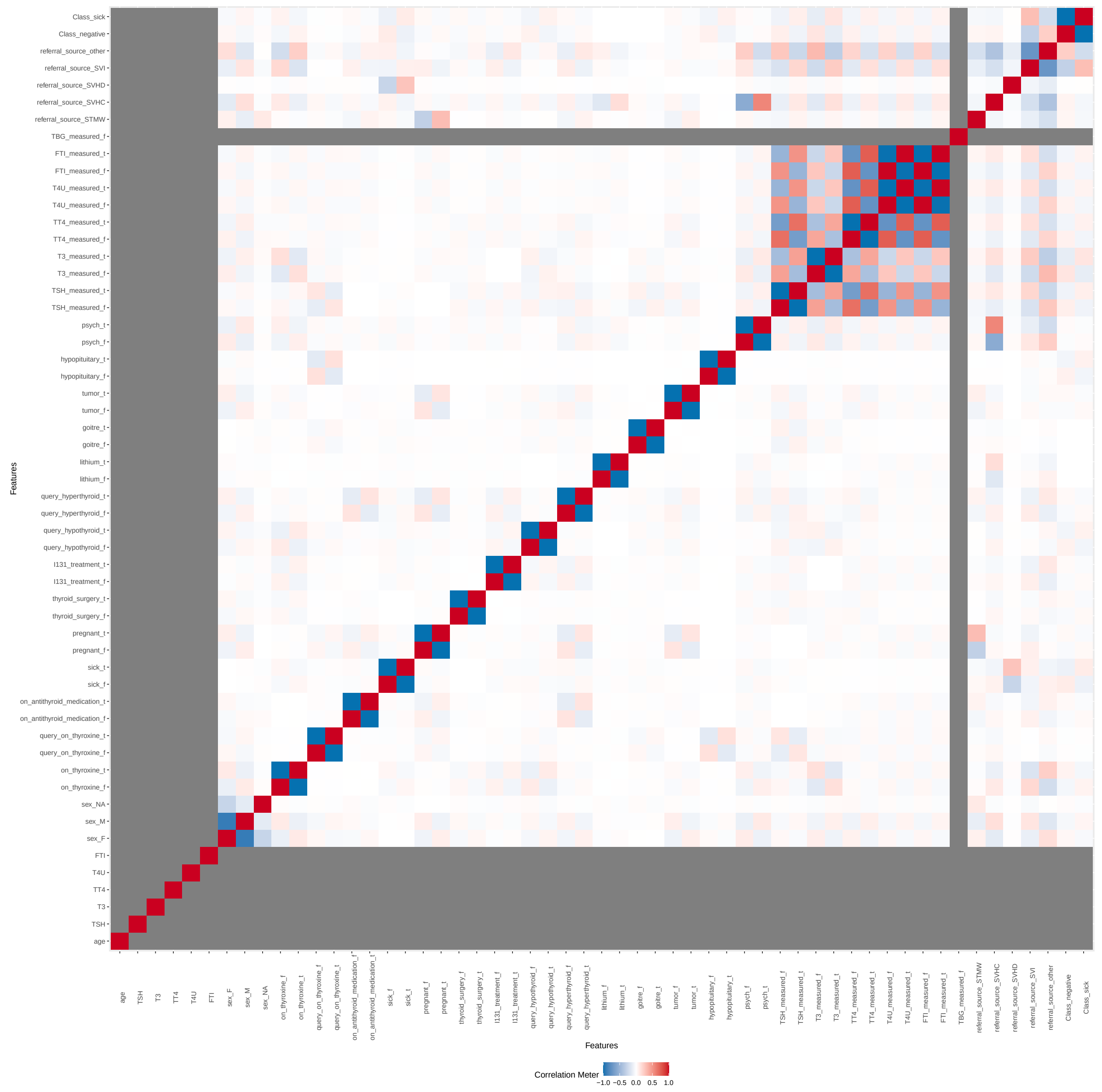
```
dataset <- dataset %>% mutate(age=replace(age, age>110 | age<0, NA))
```

In the future I am dropping all of the rows with NA so they won't be included in our model.

Next I want to see if there are any variables highly correlated that would cause our model to overfit.

```
DataExplorer::plot_correlation(dataset)
```

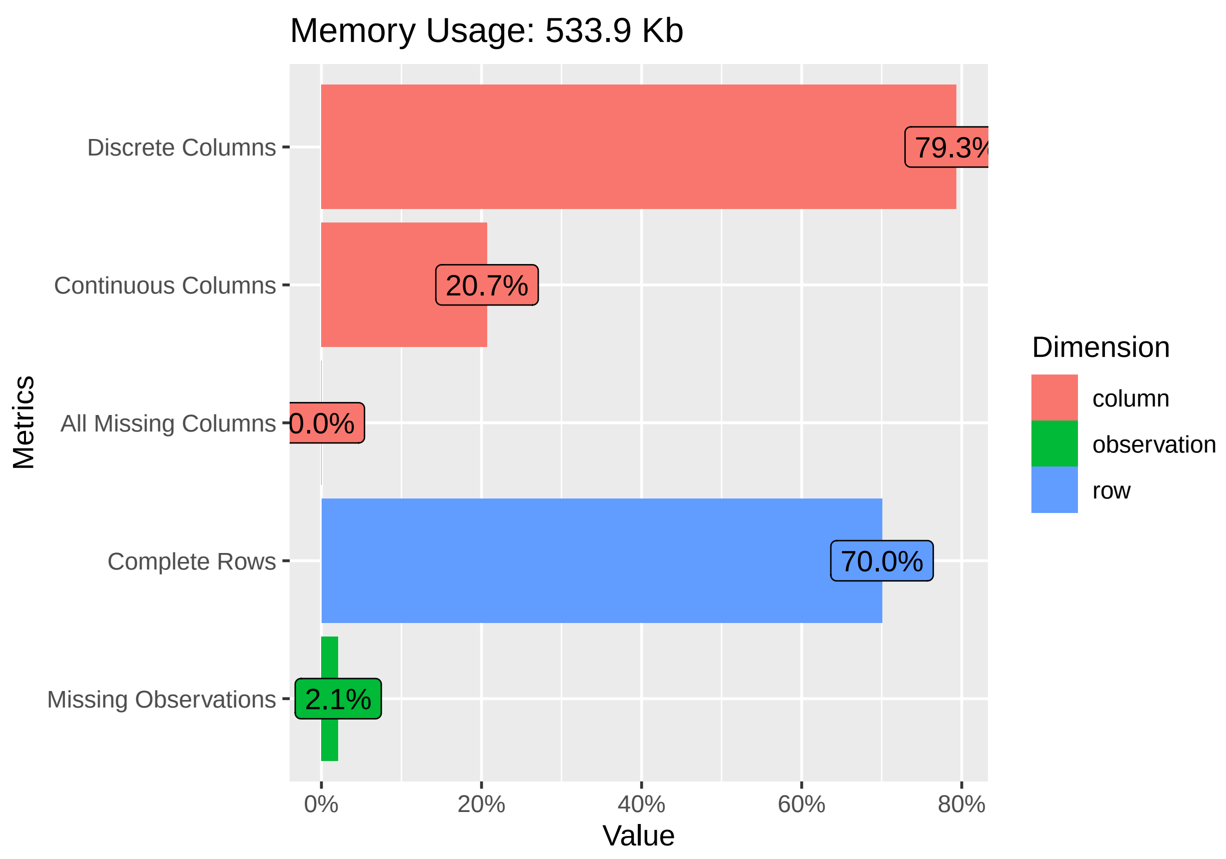
```
## Warning in cor(x = structure(list(age = c(41, 23, 46, 70, 70, 18, 59, 80, :
## odchylenie standardowe wynosi zero
```



```r
DataExplorer::introduce(dataset)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 3772      29               23                  6                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                 2293          2642             109388       546688
```

```r
DataExplorer::plot_intro(dataset)
```



```
# keeping track of positive cases before deleting rows
table(dataset$Class)
```

```
##
## negative     sick
##     3541      231
```

What I find out that there for sure are some columns with 0 variance. I also want to delete them. Other variables seem to be good.

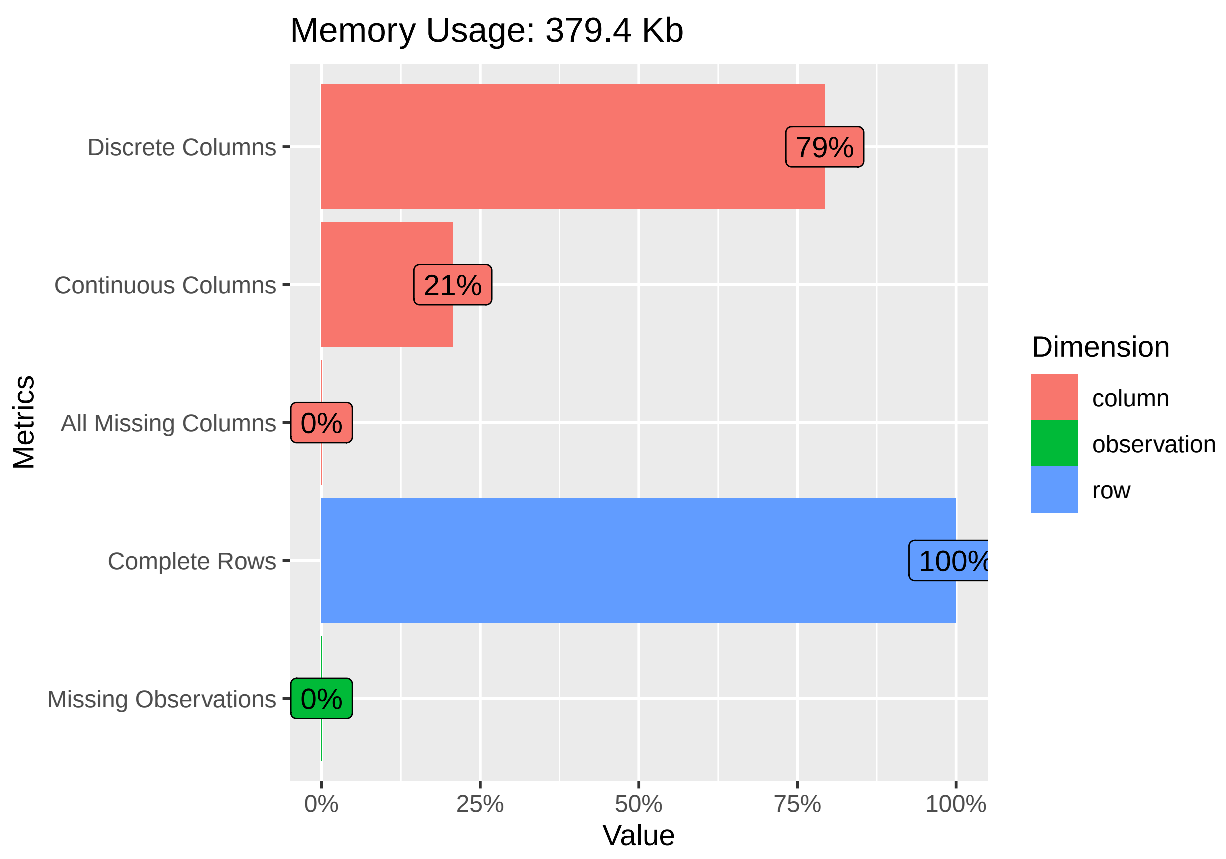
First I will drop all the rows that currently have some missing values and see if we lost a lot of positive cases.

```
dataset <- dataset %>% drop_na()

DataExplorer::introduce(dataset)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 2642      29               23                  6                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0          2642              76618       388488
```

```
DataExplorer::plot_intro(dataset)
```



```
table(dataset$Class)
```

```
##
## negative     sick
##     2430      212
```

We didn't lose a lot of positive cases yet a lot of negatives ones which is good in case of inbalanced data.

Next I will delete all the columns with 0 variance. I will recognize them thanks to function `describe`.

```
#describe(dataset)

dataset <- dataset %>%
  select(-c("TBG_measured", "FTI_measured", "T4U_measured", "TT4_measured", "T3_measured", "TSH_measure
```

Modelling

First let's split data into train and test based on the file we got.

```
train_idx <- read.table("indeksy_treningowe.txt", sep=" ", header = TRUE)$x
test_idx <- setdiff(1:3772, train_idx)
train <- dataset[train_idx,] %>% drop_na()
test <- dataset[test_idx,] %>% drop_na()

table(train$Class)
```

```
##
```

```
## negative     sick
##     1929      178
```

```
table(test$Class)
```

```
##
## negative     sick
##      501       34
```

Classes ratios seem to be pretty similiar in both train and test set.

I want to use mlr3 for modelling and a simple decision tree that is easy to explain. I also used Dominik Rafacz's great package auprc that allows us to use this measure inside of mlr3.

To use mlr3 I had to make sure it is using stratification when resampling our training set.

```
measure_auc = msr('classif.auc')
measure_auprc = msr("classif.auprc")


task2 = TaskClassif$new(id = "classif_task",
                       backend = train,
                       target = "Class",
                       positive = 'sick')

resampling = rsmp("cv", folds = 5)

r= resampling$instantiate(task2)

prop.table(table(task2$truth()))
```

```
##
##      sick  negative
## 0.0844803 0.9155197
```

```
prop.table(table(task2$truth(r$train_set(1))))
```

```
##
##       sick   negative
## 0.08605341 0.91394659
```

```
prop.table(table(task2$truth(r$train_set(2))))
```

```
##
##       sick   negative
## 0.08249258 0.91750742
```

```
prop.table(table(task2$truth(r$train_set(3))))
```

```
##
##       sick   negative
## 0.08422301 0.91577699
```

```
prop.table(table(task2$truth(r$train_set(4))))
```

```
##
##       sick   negative
## 0.08659549 0.91340451
```

```r
prop.table(table(task2$truth(r$train_set(5))))
```

```
##
##       sick   negative
## 0.08303677 0.91696323
```

We can see that each of the splits has similiar Class' ratios as the original train set. It means I can do the tuning using this resampling.

```r
learner_rpart2 = lrn('classif.rpart', predict_sets = c("train", "test"), predict_type = 'prob')


tune_ps2 = ParamSet$new(list(
  ParamDbl$new("cp", lower = 0.001, upper = 0.1),
  ParamInt$new("minsplit", lower = 16, upper = 64),
  ParamInt$new("maxdepth", lower = 7, upper = 30)
))

evals20 = term("evals", n_evals = 50)

instance2 = TuningInstance$new(
  task = task2,
  learner = learner_rpart2,
  resampling = resampling,
  measures = measure_auprc,
  param_set = tune_ps2,
  terminator = evals20
)

tuner2 = tnr('random_search')
tuner2$tune(instance2)

learner_rpart2$param_set$values <- instance2$result$params
learner_rpart2$train(task2)

pred_train <- learner_rpart2$predict(task2)
pred_test <- learner_rpart2$predict_newdata(task2, newdata = test)

score_train <- pred_train$score(measure_auprc)
score_test <- pred_test$score(measure_auprc)

auc_train <- pred_train$score(measure_auc)
auc_test <- pred_test$score(measure_auc)
```

Final results

```r
results <- data.frame(
  'auc train' = auc_train,
  'auc_test' = auc_test,
  'auprc_train' = score_train,
  'auprc_test'= score_test
)
```

```r
kable(results)
```

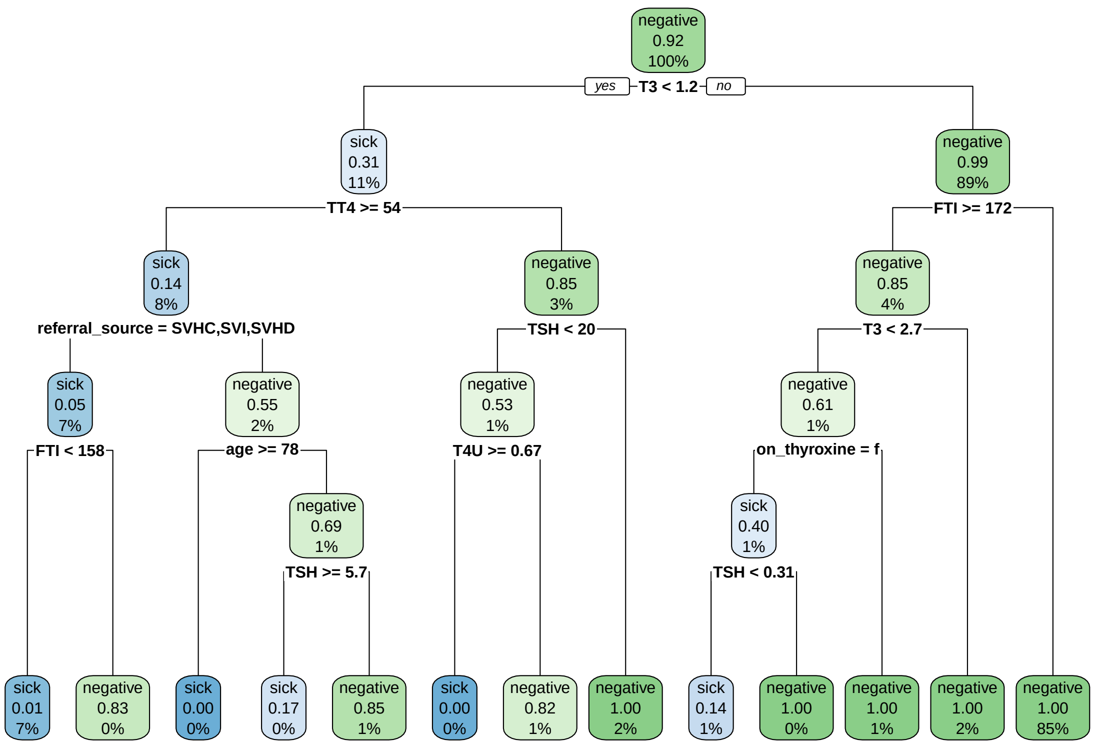
	auc.train	auc_test	auprc_train	auprc_test
classif.auc	0.9855226	0.9848538	0.9573875	0.9576183

As we deal with unbalanced data `auprc` is much better measure to look at and it seems to be really good with our very simple approach.

We can also look at how easy to interpret this model is by looking at the tree visualization and variables' importances.

What I have proven is that cleaning data without losing positive cases is the way to deal with unbalanced data.

```r
rpart.plot::rpart.plot(learner_rpart2$model,roundint=FALSE)
```



```r
kable(learner_rpart2$importance())
```

	x
T3	205.7176021
TT4	83.8303439
FTI	66.7831378
TSH	62.6804912
T4U	31.0363727
referral_source	15.2761462
age	8.5667813
on_thyroxine	7.4127397
I131_treatment	1.1210959
query_hypothyroid	1.0285714
sex	1.0285714
sick	0.7183761

Second approach

Using numerics instead of factors seems to impact our trees.

```
dataset <- data.frame(lapply(dataset, as.numeric), stringsAsFactors=FALSE)
dataset$Class <- as.factor(dataset$Class)

train <- dataset[train_idx,] %>% drop_na()
test <- dataset[test_idx,] %>% drop_na()


task2 = TaskClassif$new(id = "classif_task",
                        backend = train,
                        target = "Class",
                        positive = '2')

learner_rpart2 = lrn('classif.rpart', predict_sets = c("train", "test"), predict_type = 'prob')


tune_ps2 = ParamSet$new(list(
  ParamDbl$new("cp", lower = 0.001, upper = 0.1),
  ParamInt$new("minsplit", lower = 16, upper = 64),
  ParamInt$new("maxdepth", lower = 7, upper = 30)
))

evals20 = term("evals", n_evals = 50)

instance3 = TuningInstance$new(
  task = task2,
  learner = learner_rpart2,
  resampling = resampling,
  measures = measure_auprc,
  param_set = tune_ps2,
  terminator = evals20
)

tuner2 = tnr('random_search')
tuner2$tune(instance3)
```

```
learner_rpart2$param_set$values <- instance3$result$params
learner_rpart2$train(task2)

pred_train <- learner_rpart2$predict(task2)
pred_test <- learner_rpart2$predict_newdata(task2, newdata = test)

score_train <- pred_train$score(measure_auprc)
score_test <- pred_test$score(measure_auprc)

auc_train <- pred_train$score(measure_auc)
auc_test <- pred_test$score(measure_auc)
```

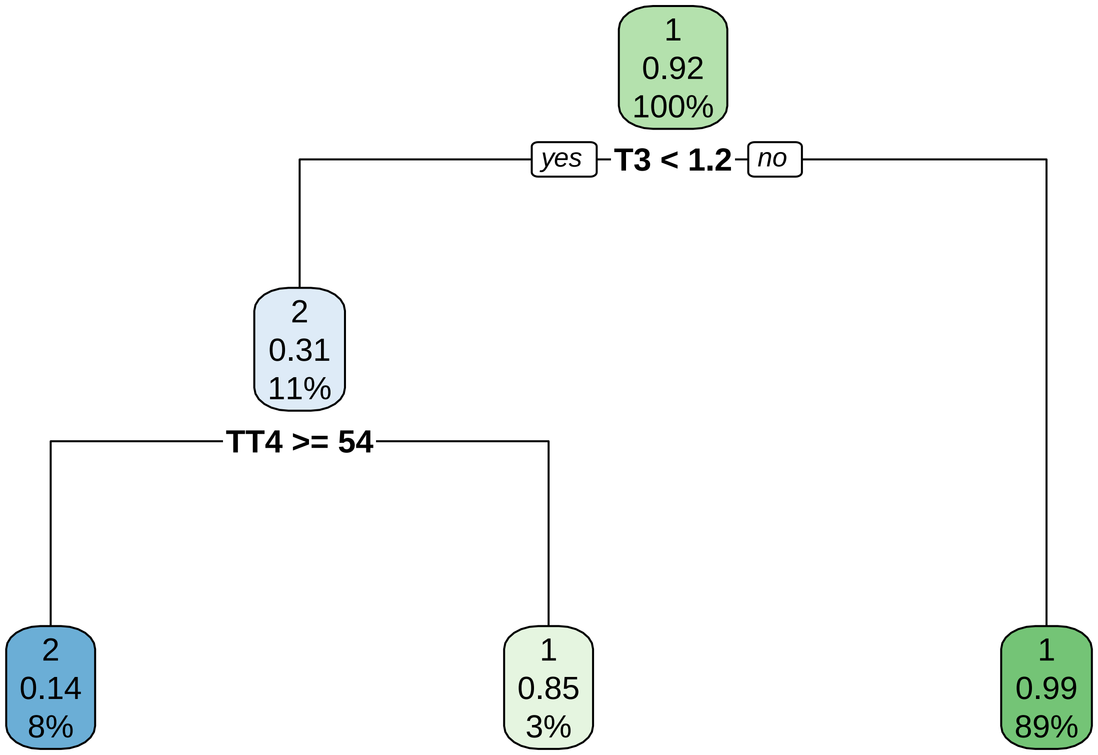
Final results

```
results <- data.frame(
  'auc train' = auc_train,
  'auc_test' = auc_test,
  'auprc_train' = score_train,
  'auprc_test'= score_test
)

kable(results)
```

	auc.train	auc_test	auprc_train	auprc_test
classif.auc	0.9437649	0.9313432	0.7779511	0.6940053

```
rpart.plot::rpart.plot(learner_rpart2$model,roundint=FALSE)
```



```r
kable(learner_rpart2$importance())
```

	x
T3	197.4060488
TT4	70.1485979
FTI	46.8812973
TSH	40.5521998
T4U	19.6119124
age	0.7791253

Blackbox

```r
learner_xgb = lrn('classif.xgboost', predict_sets = c("train", "test"), predict_type = 'prob')


tune_ps_xgb = ParamSet$new(list(
  ParamDbl$new("eta", lower = 0.01, upper = 0.3),
  ParamInt$new("nrounds", lower = 30, upper = 300),
  ParamInt$new("max_depth", lower = 7, upper = 30)
))

evals20 = term("evals", n_evals = 50)

instance_xgb = TuningInstance$new(
  task = task2,
  learner = learner_xgb,
  resampling = resampling,
  measures = measure_auprc,
  param_set = tune_ps_xgb,
  terminator = evals20
)

tuner_xgb = tnr('random_search')
tuner_xgb$tune(instance_xgb)

learner_xgb$param_set$values <- instance_xgb$result$params
learner_xgb$train(task2)

pred_train_xgb <- learner_xgb$predict(task2)
pred_test_xgb <- learner_xgb$predict_newdata(task2, newdata = test)

score_train_xgb <- pred_train_xgb$score(measure_auprc)
score_test_xgb <- pred_test_xgb$score(measure_auprc)

auc_train_xgb <- pred_train_xgb$score(measure_auc)
auc_test_xgb <- pred_test_xgb$score(measure_auc)
```

Final results

```
results <- data.frame(
  'auc train' = auc_train_xgb,
  'auc_test' = auc_test_xgb,
  'auprc_train' = score_train_xgb,
  'auprc_test'= score_test_xgb
)

kable(results)
```

	auc.train	auc_test	auprc_train	auprc_test
classif.auc	0.9999403	0.9989433	0.9941968	0.9586099

```
kable(learner_xgb$importance())
```

	x
T3	0.6574734
TT4	0.1478032
FTI	0.0633884
TSH	0.0604625
referral_source	0.0326013
age	0.0132886
on_thyroxine	0.0127648
T4U	0.0122178

We see that blackbox model outscores rpart model on train set but has similiar results to our first approach.

The most important features are the same.