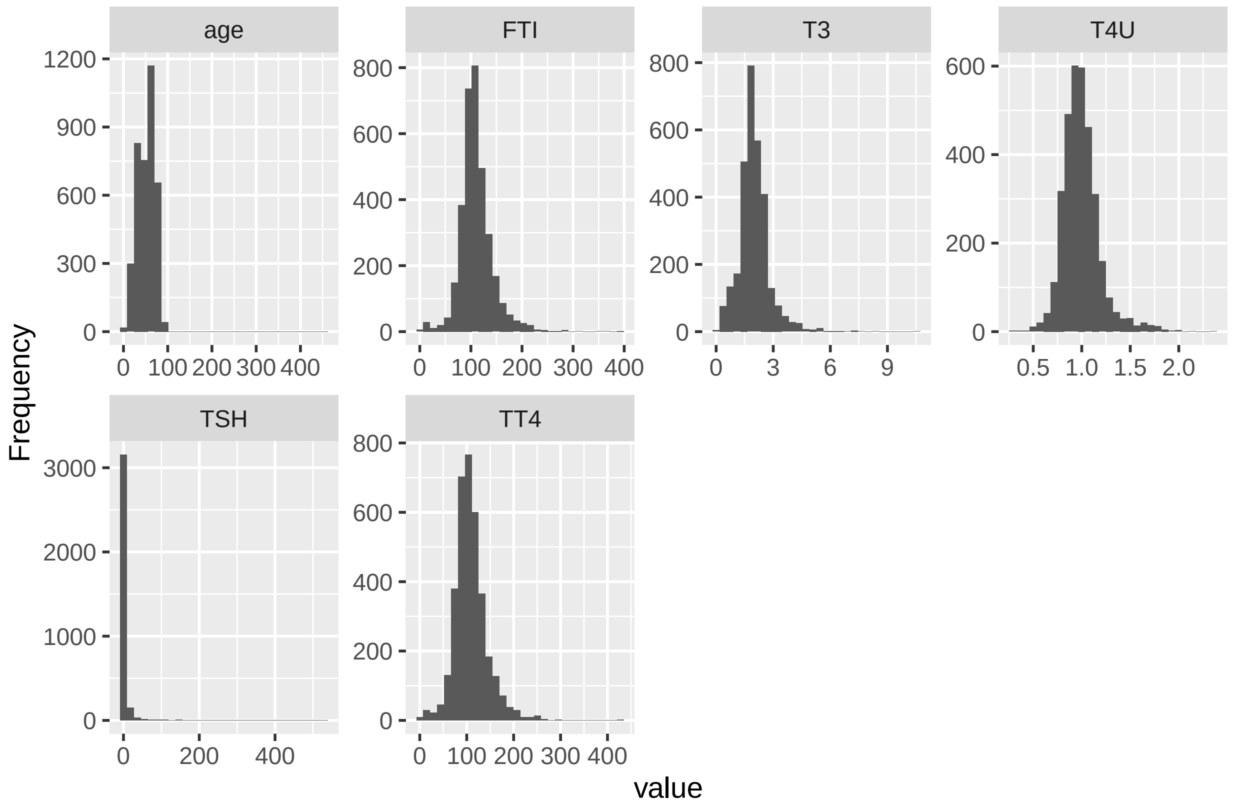
PD 2

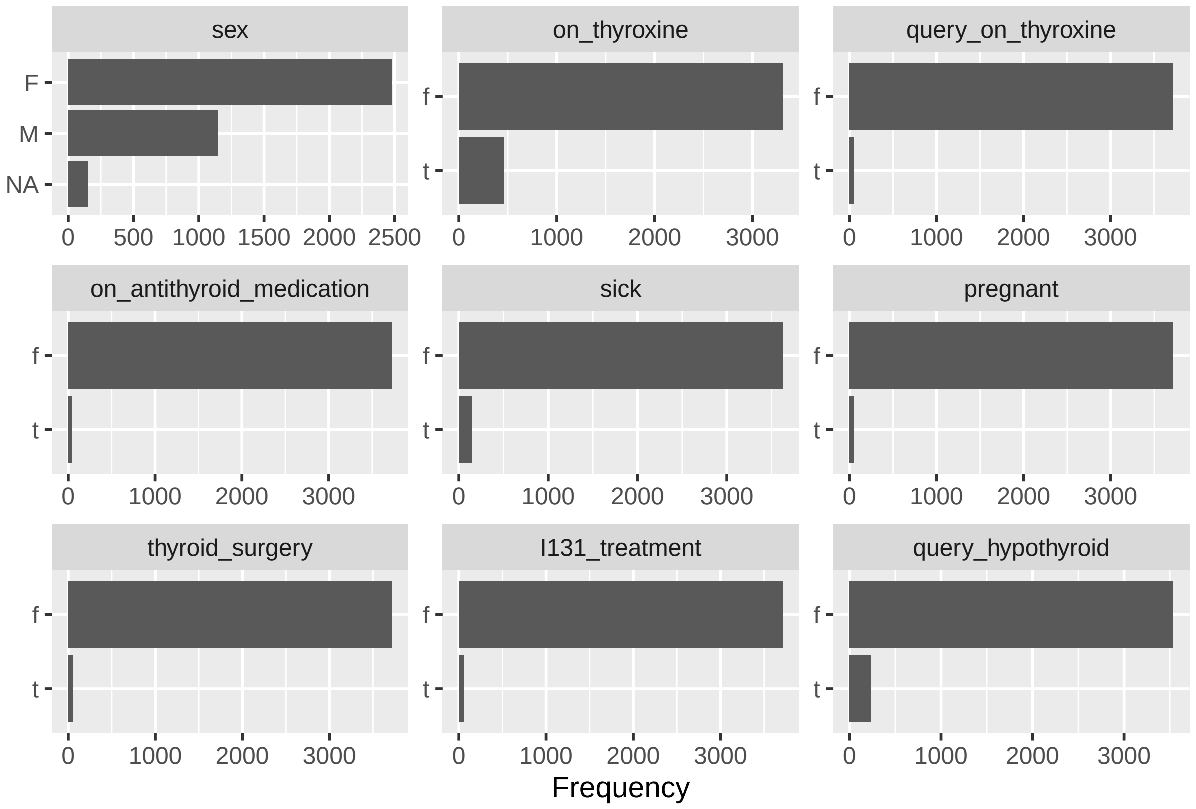
Witold Merkel

28 04 2020

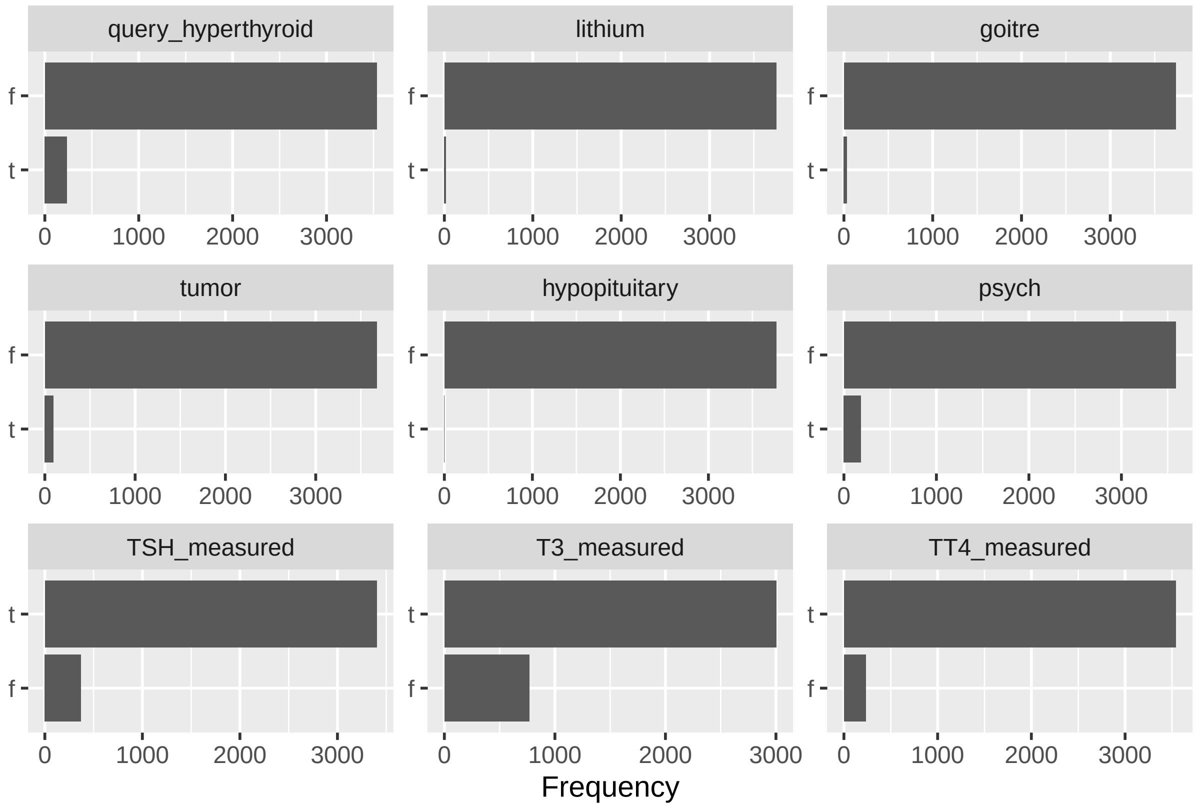
```
DataExplorer::plot_histogram(dataset_raw)
```



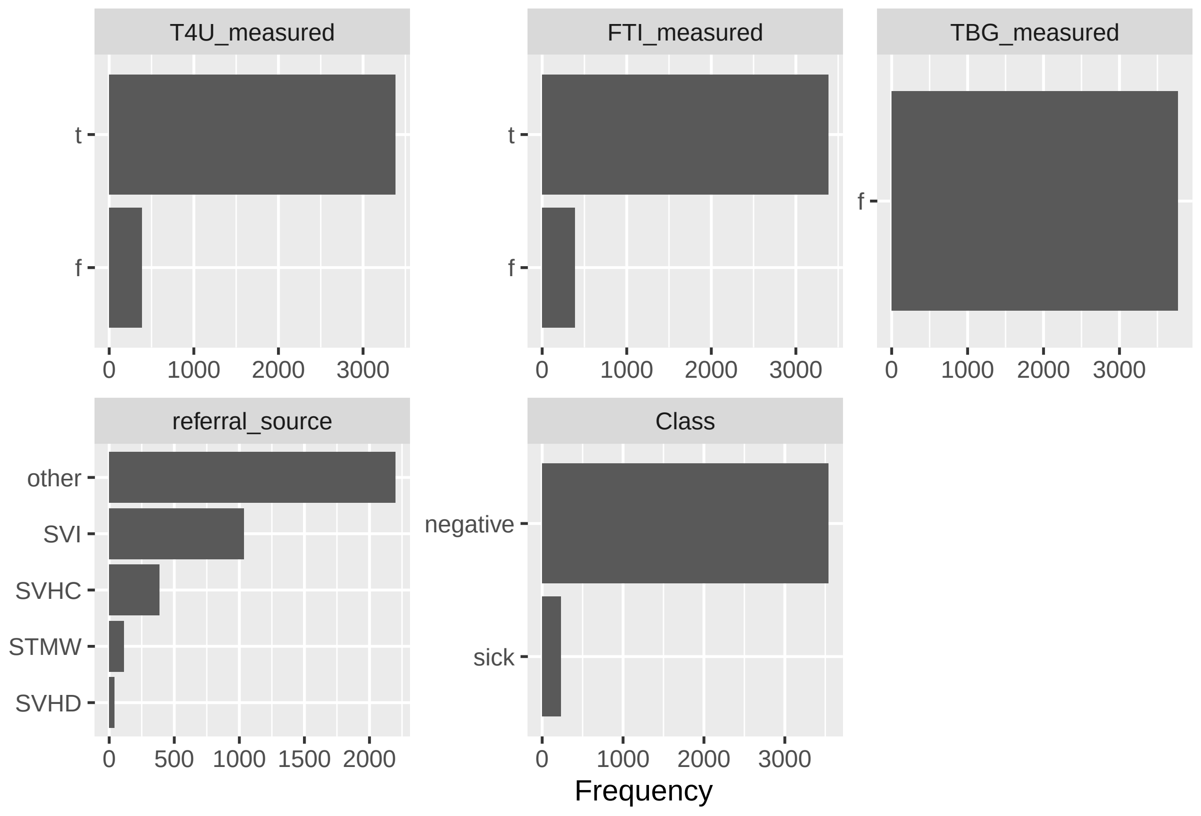
```
DataExplorer::plot_bar(dataset_raw)
```


sex
F
M
NA
0
500
1000
1500
2000
2500
on_thyroxine
query_on_thyroxine
on_antithyroid_medication
sick
pregnant
thyroid_surgery
I131_treatment
query_hypothyroid
f
t
1000
2000
3000
Frequency

Page 1


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

Page 2



Page 3

Preprocessing

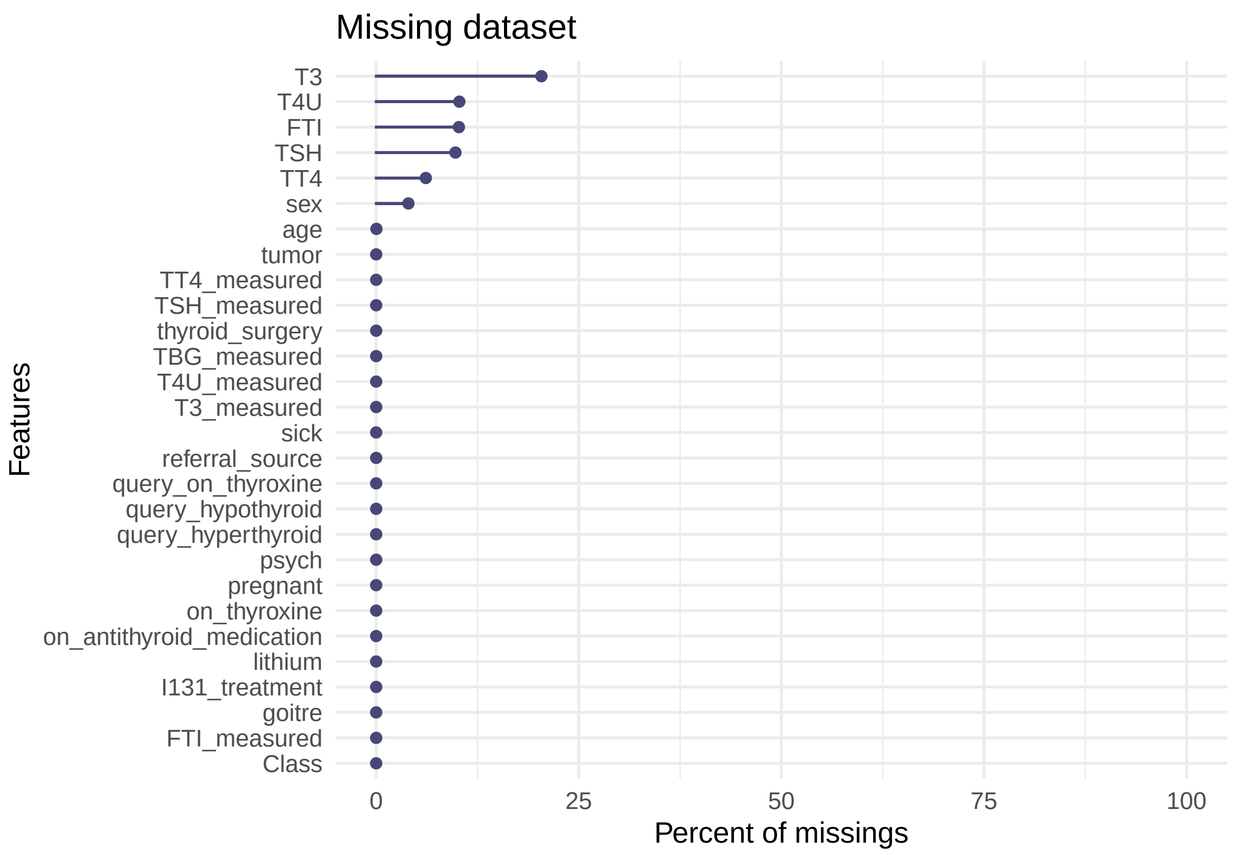
I tried different things to make my model a better one:

- logaritmic transformation of selected columns,
- getting rid of lines with missing elements,
- different tyes of imputing missing data,
- deleting columns.

None of those things helped mi with my score, so I decided to only delete the empty column and the "hypopituitary" column.

```
dataset_raw <- dataset_raw %>% select(-c("TBG", "hypopituitary"))
dataset_raw$Class <- as.factor(ifelse(dataset_raw$Class == 'sick', 1, 0))
```

```
gg_miss_var(dataset_raw,
            show_pct = TRUE) +
  ylim(0, 100) +
  labs(title = "Missing dataset",
       x = "Features",
       y = "Percent of missings")
```



Dividing data

Division that was given to us.

```
index <- read.table("index.txt")$x
train <- dataset_raw[index,]
test <- dataset_raw[-index,]
```

Basic model and its performance

After running some test I decided to go for decision trees. I choose the mlr implementation with rpart.

AUC - 5 cv on traininng data

```
task <- mlr::makeClassifTask(id = "1", data = train, target = "Class", positive = "1")
dec_tree <- mlr::makeLearner(cl = "classif.rpart", predict.type = "prob")

rd <- makeResampleDesc("CV", iters=5)
benchmark(learners = list(dec_tree), tasks = task, resamplings = rd, measures = auc)
```

```
## Task: 1, Learner: classif.rpart
```

```
## Resampling: cross-validation
## Measures:             auc
## [Resample] iter 1:    0.9689185
## [Resample] iter 2:    0.9196038
## [Resample] iter 3:    0.9584288
## [Resample] iter 4:    0.9402802
## [Resample] iter 5:    0.9701401
##
## Aggregated Result: auc.test.mean=0.9514743
##
##   task.id    learner.id auc.test.mean
## 1       1 classif.rpart     0.9514743
```

AUPRC - on test data

```r
model1 <- train(dec_tree, task)
"AUPRC"
```

```
## [1] "AUPRC"
```

```r
pred_c(model1, test)
```

```
## [1] 0.7939315
```

I think those scores aren't bad, but I would like to do something more, so it doesn't seem that I didn't do anything.

Tunning of the hyperparameters

Here I would like to present my excellent model, which I got by tunning the parameters of a basic rpart model, with random search method, which is better than searching through grid.

```r
params <- makeParamSet(
  makeIntegerParam("minsplit", lower = 5, upper = 100),
  makeIntegerParam("minbucket", lower = 2, upper = 100),
  makeNumericParam("cp", lower = 0.001, upper = 0.8),
  makeIntegerParam("maxdepth", lower = 1, upper = 30))

ctrl <- makeTuneControlRandom(maxit = 100)
classification_best <- tuneParams("classif.rpart", task = task, resampling = rd,
                                  par.set = params, control = ctrl, show.info = FALSE,
                                  measures = list(acc))
classification_best
```

```
## Tune result:
## Op. pars: minsplit=25; minbucket=34; cp=0.0318; maxdepth=30
## acc.test.mean=0.9817639
```

```
rpart <- makeLearner("classif.rpart", predict.type = "prob",
                      par.vals = list("cp" = classification_best$x$cp,
                     "maxdepth" = classification_best$x$maxdepth,
                     "minsplit" = classification_best$x$minsplit,
                     "minbucket" = classification_best$x$minbucket))
rf_hyper <- train(rpart, task)
```

Above we can see the optimal found parameters for our problem. After learning them I made a better model.

Performance of the better model

Let's check how much better our new model is.

AUC - 5 cv on training data

```
benchmark(learners = list(rpart), tasks = task, resamplings = rd, measures = auc)
```

```
## Task: 1, Learner: classif.rpart

## Resampling: cross-validation

## Measures:             auc

## [Resample] iter 1:    0.9791741

## [Resample] iter 2:    0.9169326

## [Resample] iter 3:    0.9630611

## [Resample] iter 4:    0.9668121

## [Resample] iter 5:    0.9508167

##

## Aggregated Result: auc.test.mean=0.9553593

##

##   task.id    learner.id auc.test.mean
## 1       1 classif.rpart     0.9553593
```

AUPRC - on test data

```
"AUPRC"
```

```
## [1] "AUPRC"
```

```
pred_c(rf_hyper, test)
```

```
## [1] 0.6413837
```

We can see that the model is better when it comes to AUC, but it's performance acording to AUPRC is worst, so we have to decide which meassure is more important to us and act accordingly.

Black box

For my black box model I decided to use XGBoost, because it is very effective and I had a presentation about it, so it is perfect.

Firstly I will chech the perforamnce of the bare model and then check if some actions can up my score.

Basic XGBoost

```r
task <- mlr::makeClassifTask(id = "1", data = train_no_factor, target = "Class",
                             positive = "1")
xgb <- mlr::makeLearner(cl = "classif.xgboost", predict.type = "prob")

rd <- makeResampleDesc("CV", iters=5)
benchmark(learners = list(xgb), tasks = task, resamplings = rd, measures = auc)
```

```
## Task: 1, Learner: classif.xgboost
## Resampling: cross-validation
## Measures:             auc
## [Resample] iter 1:    0.9567377
## [Resample] iter 2:    0.9813398
## [Resample] iter 3:    0.9346318
## [Resample] iter 4:    0.9982477
## [Resample] iter 5:    0.9270527
##
## Aggregated Result: auc.test.mean=0.9596019
##
##   task.id      learner.id auc.test.mean
## 1       1 classif.xgboost     0.9596019
```

We can see that our score is slightly better than last time, now lets calculate AUPRC.

```r
model1 <- train(xgb, task)
"AUPRC"
```

```
## [1] "AUPRC"
```

```r
pred_c(model1, test_no_factor)
```

```
## [1] 0.9330146
```

Tunning of the hyperparameters

Here I would like to present my excellent model, which I got by tunning the parameters of a basic xgboost model, with random search method, which is better than searching through grid.

```r
params <- makeParamSet(
  makeIntegerParam("nrounds", lower = 100, upper = 500),
  makeIntegerParam("max_depth", lower = 1, upper = 10),
  makeNumericParam("eta", lower = .1, upper = .5),
  makeNumericParam("lambda", lower = -1, upper = 0, trafo = function(x) 10^x))

ctrl <- makeTuneControlRandom(maxit = 100)
classification_best <- tuneParams("classif.xgboost", task = task, resampling = rd,
                                  par.set = params, control = ctrl, show.info = FALSE,
                                  measures = list(acc))
classification_best
```

```
## Tune result:
## Op. pars: nrounds=159; max_depth=7; eta=0.365; lambda=0.278
## acc.test.mean=0.9890586
```

```r
xgboost <- makeLearner("classif.xgboost", predict.type = "prob",
                     par.vals = list("nrounds" = classification_best$x$nrounds,
                    "max_depth" = classification_best$x$max_depth,
                    "eta" = classification_best$x$eta,
                    "lambda" = classification_best$x$lambda))
xg_hyper <- train(xgboost, task)
```

Performance of the better model

Let's check how much better our new model is.

AUC - 5 cv on training data

```r
benchmark(learners = list(xgboost), tasks = task, resamplings = rd, measures = auc)
```

```
## Task: 1, Learner: classif.xgboost

## Resampling: cross-validation

## Measures:             auc

## [Resample] iter 1:    0.9950058

## [Resample] iter 2:    0.9912571

## [Resample] iter 3:    0.9965096

## [Resample] iter 4:    0.9975263

## [Resample] iter 5:    0.9991320

##

## Aggregated Result: auc.test.mean=0.9958862

##

##   task.id      learner.id auc.test.mean
## 1       1 classif.xgboost     0.9958862
```

AUPRC - on test data

```r
"AUPRC"
```

```
## [1] "AUPRC"
```

```r
pred_c(xg_hyper, test_no_factor)
```

```
## [1] 0.9461535
```

Conclusion

We can see that tuning the hyperparameers helped this time, because our AUC and AUPRC went up so we should be happy. We also can see that this blackbox is much more accurate that the previous explainable model.

Formality

Potwierdzam samodzielność powyższej pracy oraz niekorzystanie przeze mnie z niedozwolonych źródeł.

Witold Merkel