WB2 - Praca domowa

Łukasz Brzozowski

16 kwietnia 2020

1 Opis zbioru danych

Let us start with the dataset analysis. Thanks to `mlr::summarizeColumns()` we may notice some pecularities in the dataset *sick*. Firstly, the columns **TBG** and **TBG_measured** contain no information about the samples, as they are consisted of a single value and no value respectively. It gives us a natural reason to remove them from the dataset. Moreover, trying to generate a simple model on the dataset returns an error signalizing a factor level of **hypopituitary** in the testing set with 0 occurrences in the training dataset. Whille more complex predictive models could have easily handled such difference, as we are limited to simple models, we therefore decide to remove **hypopituitary** from the dataset.

	name	type	na	mean	disp	median	mad	min	max	nlevs
1	age	numeric	0	51.5185676	18.905797370	54.00	22.23900	1.000	94.00	0
2	sex	factor	119	NA	NA	NA	NA	906.000	1991.00	2
3	on_thyroxine	factor	0	NA	0.122015915	NA	NA	368.000	2648.00	2
4	query_on_thyroxine	factor	0	NA	0.012599469	NA	NA	38.000	2978.00	2
5	on_antithyroid_medication	factor	0	NA	0.012267905	NA	NA	37.000	2979.00	2
6	sick	factor	0	NA	0.037466844	NA	NA	113.000	2903.00	2
7	pregnant	factor	0	NA	0.014588859	NA	NA	44.000	2972.00	2
8	thyroid_surgery	factor	0	NA	0.013594164	NA	NA	41.000	2975.00	2
9	I131_treatment	factor	0	NA	0.016246684	NA	NA	49.000	2967.00	2
10	query_hypothyroid	factor	0	NA	0.063992042	NA	NA	193.000	2823.00	2
11	query_hyperthyroid	factor	0	NA	0.062665782	NA	NA	189.000	2827.00	2
12	lithium	factor	0	NA	0.004641910	NA	NA	14.000	3002.00	2
13	goitre	factor	0	NA	0.009615385	NA	NA	29.000	2987.00	2
14	tumor	factor	0	NA	0.023872679	NA	NA	72.000	2944.00	2
15	hypopituitary	factor	0	NA	0.000000000	NA	NA	0.000	3016.00	1
16	psych	factor	0	NA	0.049734748	NA	NA	150.000	2866.00	2
17	TSH_measured	factor	0	NA	0.099801061	NA	NA	301.000	2715.00	2
18	TSH	numeric	301	5.2005672	26.407995825	1.30	1.48260	0.005	530.00	0
19	T3_measured	factor	0	NA	0.205901857	NA	NA	621.000	2395.00	2
20	T3	numeric	621	2.0174948	0.831273637	2.00	0.59304	0.050	10.60	0
21	TT4_measured	factor	0	NA	0.063328912	NA	NA	191.000	2825.00	2
22	TT4	numeric	191	108.5881416	35.591033048	104.00	26.68680	2.000	430.00	0
23	T4U_measured	factor	0	NA	0.104774536	NA	NA	316.000	2700.00	2
24	T4U	numeric	316	0.9963719	0.195110285	0.98	0.14826	0.250	2.32	0
25	FTI_measured	factor	0	NA	0.104111406	NA	NA	314.000	2702.00	2
26	FTI	numeric	314	110.5655811	32.879337473	107.00	22.23900	2.000	395.00	0
27	TBG_measured	factor	0	NA	0.000000000	NA	NA	3016.000	3016.00	1
28	TBG	numeric	3016	NaN	NA	NA	NA	Inf	-Inf	0
29	referral_source	factor	0	NA	0.420755968	NA	NA	29.000	1747.00	5
30	Class	factor	0	NA	0.061007958	NA	NA	184.000	2832.00	2

Rysunek 1: Training dataset summary

Many columns contain true/false labels as factors. To tidy up the dataset, we convert all the factor labels to numerical labels.

2 A simple model

Before any pre-processing, it is worth to check a simple model's performance. We will use a decision tree from **rpart** package.

```
tsk <- makeClassifTask(data = datTrainF, target = "Class", positive = "1")
lrn <- makeLearner("classif.rpart", predict.type = "prob")
resample(lrn, tsk, cv5, list(mlr::auc, auprc))
```

The above model - a simplest implementation with default hyperparameters - yields surprisingly good results: 96% AUC and 87% AUPRC. Although these number are quite big, we wish to improve the model nonetheless.

3 Feature selection

Let us check, whether there are features in the dataset having a negative impact on the prediction. We will use a natural Feature Importance measure for the decision tree based on the impurity measure.

```
validIX <- datTrainF$Class %>% createDataPartition(p = 0.8, list = FALSE)

train <- datTrainF[validIX,]
test <- datTrainF[-validIX,]

rp <- rpart(Class~., data = train, model = TRUE)
pred <- predict(rp, test)

s <- summary(rp)
```

The **summary** object returns the Feature Importance values for every variable. Since only 9 variables achieves the FI measure higher than 1, we may compare the prediction on a dataset without the rest of the columns with our previous prediction.

```
s$variable.importance
nm <- names(s$variable.importance[1:9])
nm[10] <- "Class"
datTrainN <- datTrainF[,nm]

tsk <- makeClassifTask(data = datTrainN, target = "Class", positive = "1")
lrn <- makeLearner("classif.rpart", predict.type = "prob")
resample(lrn, tsk, cv5, list(mlr::auc, auprc))
```

the above model, despite having removed 17 (1) columns, yields very similar results to the previous simple model **AUC** equal 95.8% and **AUPRC** equal 87.6%. Therefore, it suffices to continue our feature engineering on the smaller dataset, as we likely will reduce any noise in the dataframe without loss of efficiency. The variables left are: "T3", "TT4", "TSH", "FTI", "referral_source", "on_thyroxine", "T3_measured", "age", "T4U", "Class" (ordered by measured Feature Importance).

4 Data imputation

Next, we shall look into missing data, which is quite often on the dataset. Although the decision tree works fine on such datasets, imputing the data may improve models' accuracy. We will test two variants of the imputation - a classic, common one, and imputation with predictive models.

4.1 Imputacja klasyczna

In the classic imputation process with replace the missing values with:

- a mode - for categorical data,
- a mean - for numerical data.

```
datTrainI <- impute(datTrainN, classes = list(numeric = imputeMean(),
                                              factor = imputeMode()))$data
```

the above imputation has not yielded expected results: on the cross-validated dataset we achieved 95.5% **AUC** and 87.4% **AUPRC**. We might as well have done nothing.

4.2 Imputation with predicted values

The idea behind this variant is using predictive models to fill out the blank spots. For each column with missing data, we generate another decision tree model predicting the missing values. An important question is whether to start filling the data from the most important columns or inversely, from the least important. It is crucial due to the fact that the most important variable, namely **T3**, contains more 600 missing values. As the columns in the dataset are now ordered with the importance, we need to choose to start from either side. Both ideas have upsides and downsides; however, due to the fact that our most important variable nas so many NAs, we decide to start filling the dataset from the most important ones. This is because if we were to start from the least important ones, we would fill over 600 spots in **T3** with predictions based on predictions based on predictions .... We would therefore risk heavily biasing the data.

```
genData <- function(dat, name){
  datToImpute <- dat[is.na(dat[name]),]
  datFull <- dat[!is.na(dat[name]),]
  n <- nrow(datFull)


  if(class(unlist(c(datFull[name]))) == "factor"){
    print("Class: factor")
    tsk <- makeClassifTask(id = name, datFull, name,
                           fixup.data = "no", check.data = FALSE)
    lr <- makeLearner("classif.rpart")
  }else{
    print("Class: numeric")
```

```
    tsk <- makeRegrTask(id = name, datFull, name,
                        fixup.data = "no",check.data = FALSE)
    lr <- makeLearner("regr.rpart")
  }
  model <- mlr::train(lr, tsk)
  pred <- as.data.frame(predict(model,
                                newdata = datToImpute %>% select(-name)))
  colnames(pred) <- c(name)
  nd <- cbind(datToImpute %>% select(-name), pred)
  nd <- nd[colnames(datFull)]
  rbind(datFull, nd)
}
```

The above function generates a new dataset with imputed values of the "name" column. Iterating on the column set of the dataset, it allows us to fill all the blank spaces. Such imputation resulted in our first success, when it comes to improving the measures' values: **AUPRC** went up more than 1 percentage point, achieving 88.6%. **AUC** remains almost unchanged with 96.4%.

5 Variable discretization

Motivated with previous success we then proceed to data discretization - it may improve the model's efficiency, by transforming numerical data into categorical.

```
require(arules)
datTrainD <- datTrainI
datTrainD$age <- discretize(datTrainD$age, breaks = 5,
                            labels = c(0, 1, 2, 3, 4))
```

We present an example of **age** discretization. We repeated the above process for each numerical variable separately (only one variable was discretized at a time). We came to the conclusion that the idea was hopeless and yielded no positive results.

6 Hyperparameter tuning

What remains is hyperparameter tuning to maximize the **AUPRC** value. I use the bayesian optimisation from `mlrMBO` package. Our search space is following:

- `minsplit` - from 1 to 30,
- `cp` - from 0 to 1,
- `maxcompete` - from 0 to 10,
- `usesurrogate` - values: 0, 1 and 2,
- `maxdepth` - from 1 to 30

We use 100 iterations in the optimisation process

```
train <- datTrainI[validIX,]
test <- datTrainI[-validIX,]

tsk <- makeClassifTask(data = train, target = "Class", positive = "1")

rp <- makeSingleObjectiveFunction(name = "rpart.tuning",
                      fn = function(x) {
                      lrn <- makeLearner("classif.rpart",
                                      par.vals = x, predict.type = "prob")
                      m <- mlr::train(lrn, tsk)

                      pred <- predict(m, newdata = test)
                      prob <- getPredictionProbabilities(pred)

                      fg <- prob[test[,"Class"] == 1]
                      bg <- prob[test[,"Class"] == 0]
                      pr <- pr.curve(scores.class0 = fg,
                                      scores.class1 = bg, curve = F)
                      pr$auc.integral
                      },
                      par.set = par.set,
                      noisy = TRUE,
                      has.simple.signature = FALSE,
                      minimize = FALSE
)
ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = iters)
res = mbo(rp, control = ctrl, show.info = TRUE)
```

The above optimisation algorithm was not able to find a better set of parameters than the default one - the maximum of **AUPRC** found was 84%, significantly below our previous results.

7 Decision Tree Regressor

To finish our research, we propose to use a Decision Tree Regressor instead of Decision Tree Clasifier to predict the target values. As the values are convieniently mapped to 0 and 1 values and the DTR model won't produces values outside of the $[0, 1]$ interval in that dataset, we may therefore tahe the returned value for each sample as a probability of belonging the the positive class. We will produce the results for three variants of DTR optimisation:

7.0.1 Variant 1

In the first variant we minimize **AUPRC** for a prediction of separate testing subset of the training dataset, which is invariant to the optimising process iterations. While we

have achieved extremely high results (99% **AUPRC**), the model overfitted quite easily. We achieved much lower results on the testing dataset than the standard classification yielded.

7.0.2 Variant 2

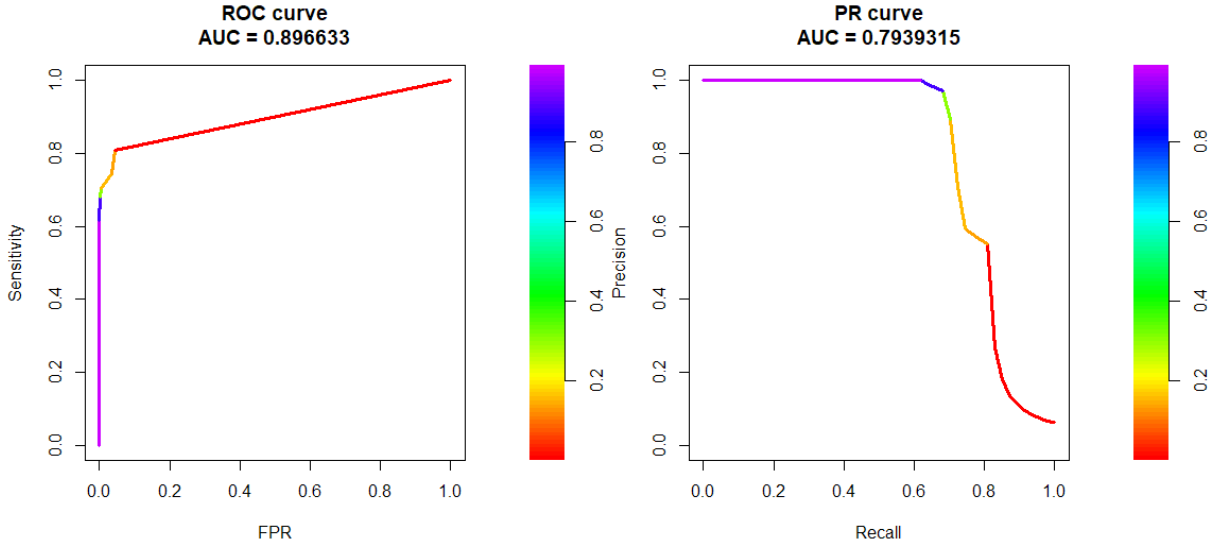
In the second variant we optimised the cross-validated measure on the training dataset. After optimisation, we have achieved almost 93% **AUPRC**; however, the model was poorly fit to the testing dataset, reaching 87.5% **AUPRC** - the overfitting happened once again.

7.0.3 Variant 3

In the third variant we do not optimise the parameters and generate a model on almost raw data achieved after the basic pre-processing. It gave absolutely best effect on the testing dataset: 92.4% **AUPRC** and 97.6% **AUC**

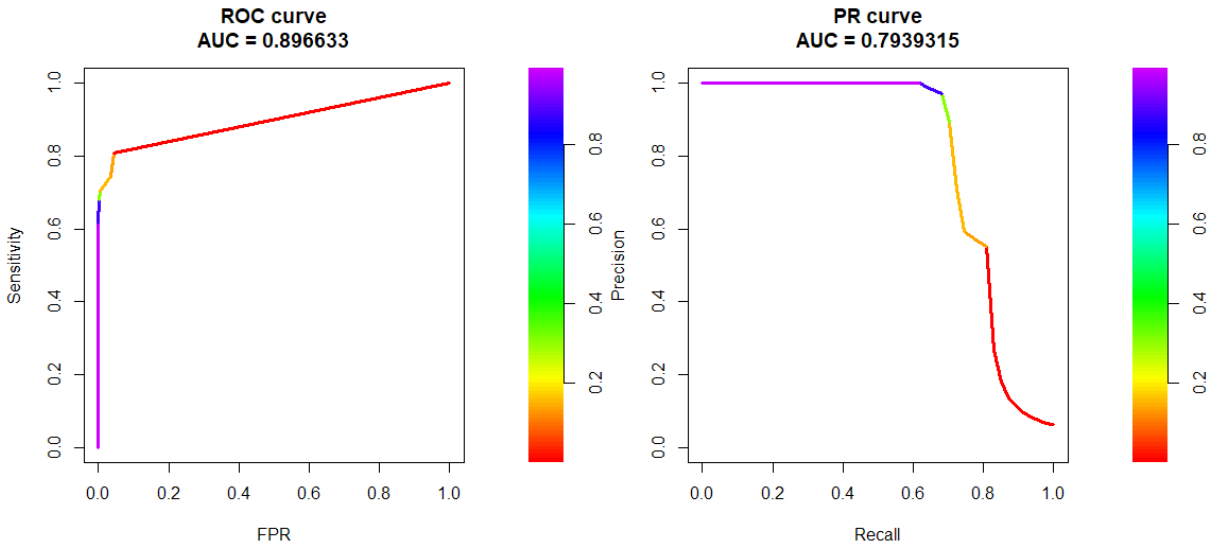
8 Combined results

For each above-presented engineering and processing operation we may now summarize achieved results on the testing dataset.



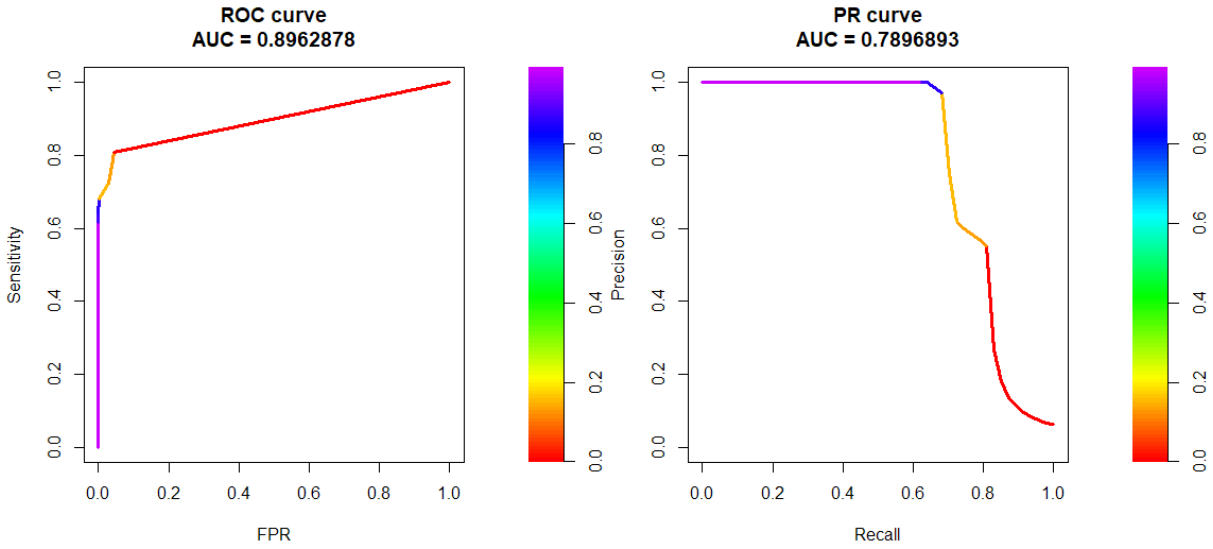
(a) ROC of the basic model

(b) PRC of the basic model



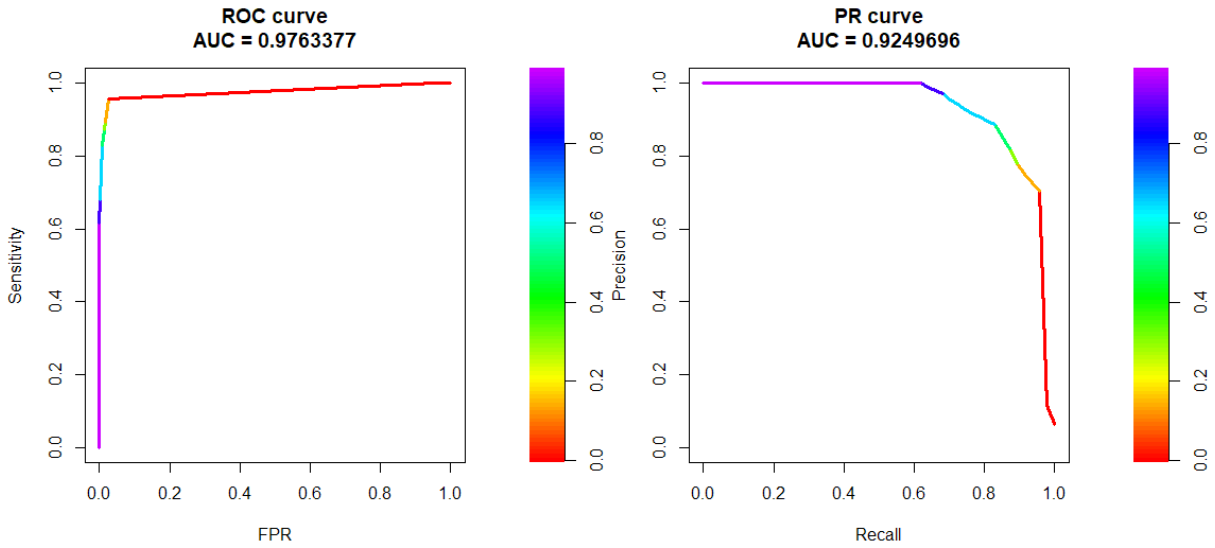
(a) ROC after feature removal

(b) PRC after feature removal



(a) ROC after imputation

(b) PRC after imputation



(a) ROC of DTR

(b) PRC of DTR

Having completed the analysis, we conclude that the decision tree classicator achieved its possible maximal efficiency, since the differences between presented models are almost insignificant. However, treating the problem as a regression problem instead of classification yields surprisingly high results. Although the model exhibited a tendency to overfit easily, it is worth to research it further, as it seems to be an unexpected ally in anomaly detection.