Sick dataset analysis

Bogdan Jastrzębski

5 kwietnia 2020 r.

Contents

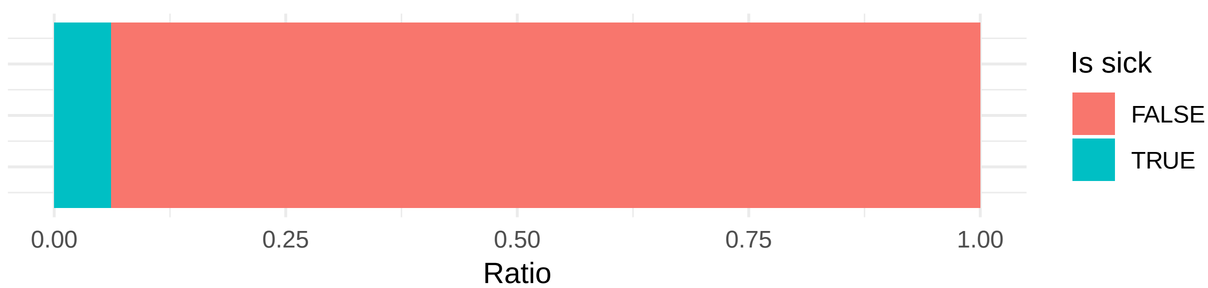
1 Introduction

In the following paper, I present an analysis of the "Sick" dataset, along with the strategy for predicting "Class" in an interpretable manner and it's results.

2 Initial Data Mining

In this section I will address all the major issues with the dataset and describe the way to face them.

2.1 Balance of the "Class" Variable



The "Class" variable is not balanced, which indicates, that we need to measure model performance in more sophisticated way than calculating accuracy of a given model. I will use the following two measures:

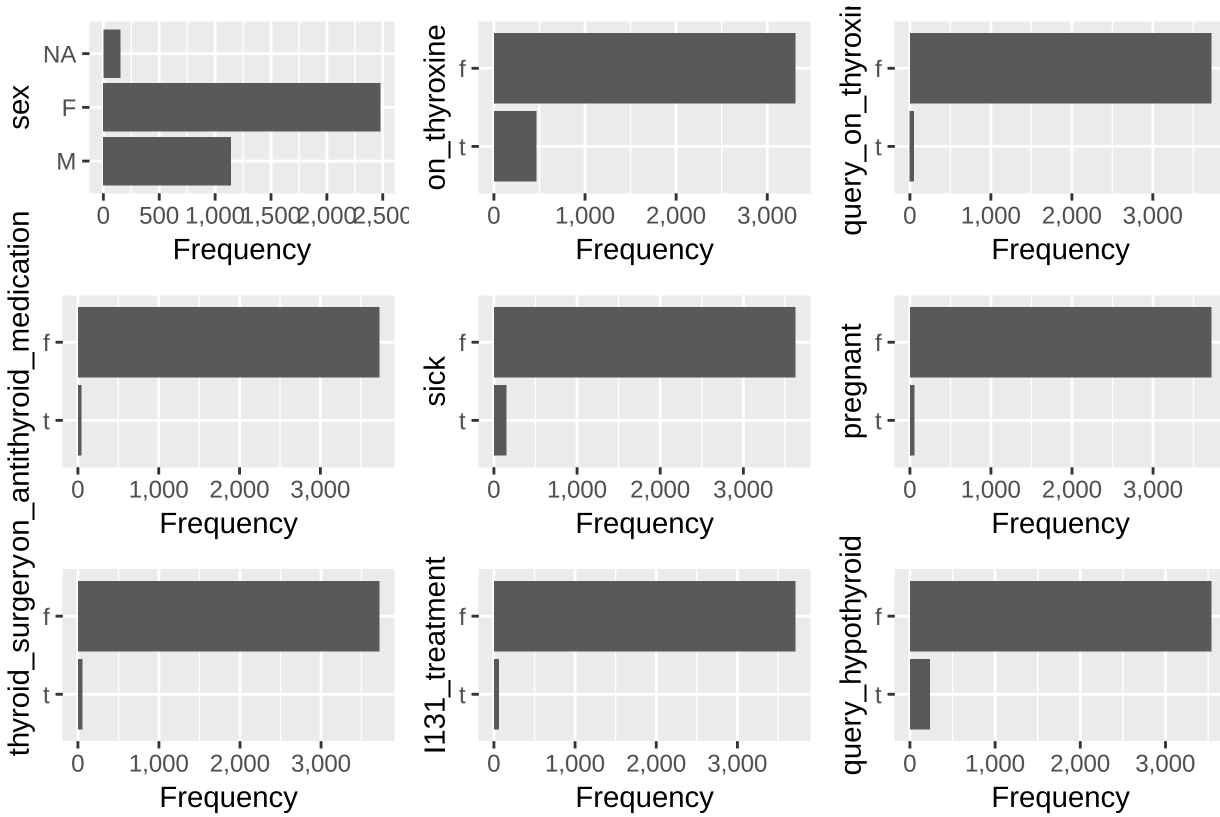
- auc
- auprc

2.2 Distributions of Categorical Variables

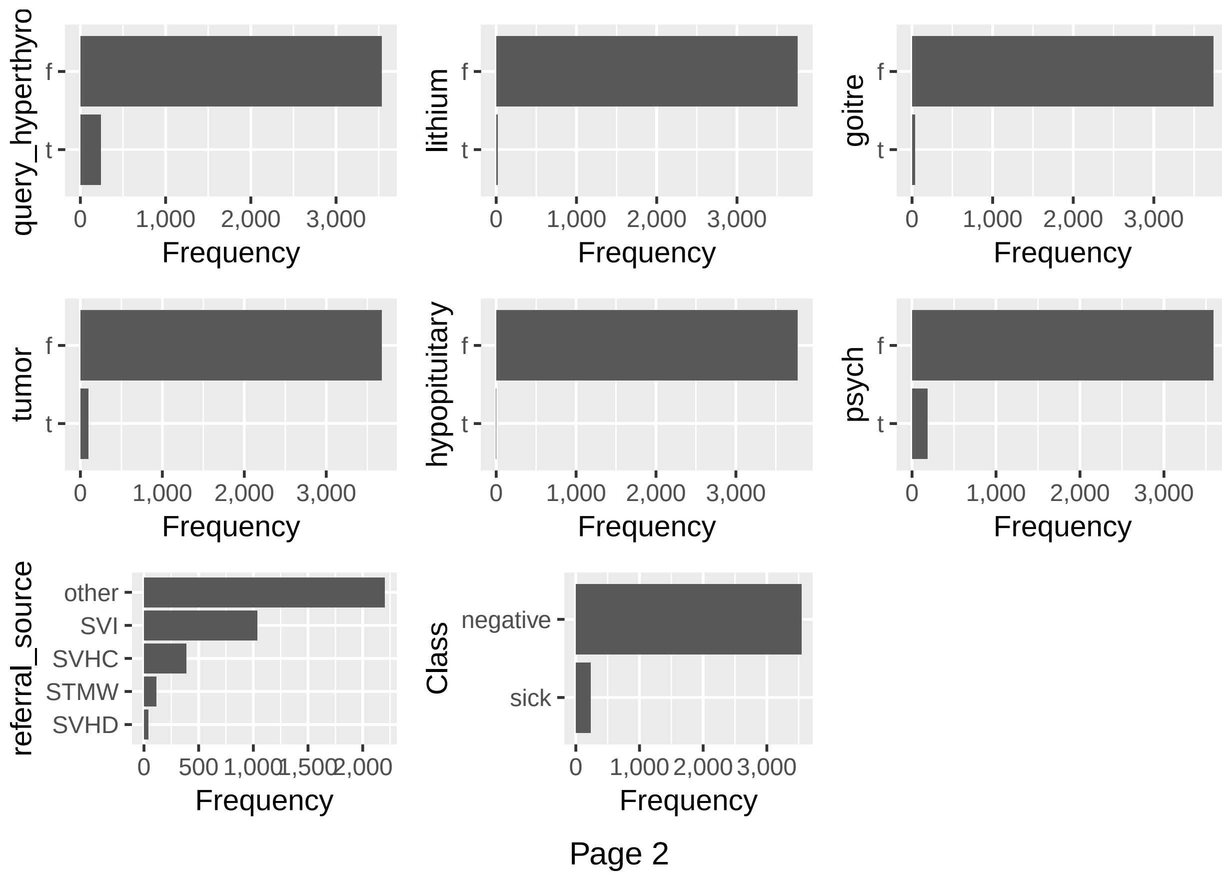
In this section, I will deliberate on some of the variables in our dataset.

2.2.1 Exploration

Distributions of random variables are as follow:



Page 1



There is a number of variables, that are nearly constant, namely:

- query_on_thyroxine
- on_antithyroid_medication
- pregnant
- thyroid_surgery
- I131_treatment
- lithium
- goitre
- hypopituitary

Do these variables have an impact on "Class"?

χ^2 test results:

Names	p.values
query_on_thyroxine	0.7716142
on_antithyroid_medication	0.1229385
pregnant	0.0734633
thyroid_surgery	0.0809595
I131_treatment	0.1864068
lithium	1.0000000
goitre	1.0000000
hypopituitary	0.0659670

As we can see, some of those variables may in fact be connected with "Class". I will exclude the "hypopituitary" variable due to a very little information it provides (distribution of "hypopituitary" is 1:3771). Even if this variable is important, we have no statistical certainty to say so, given that only one observation is positive.

2.2.2 Solution

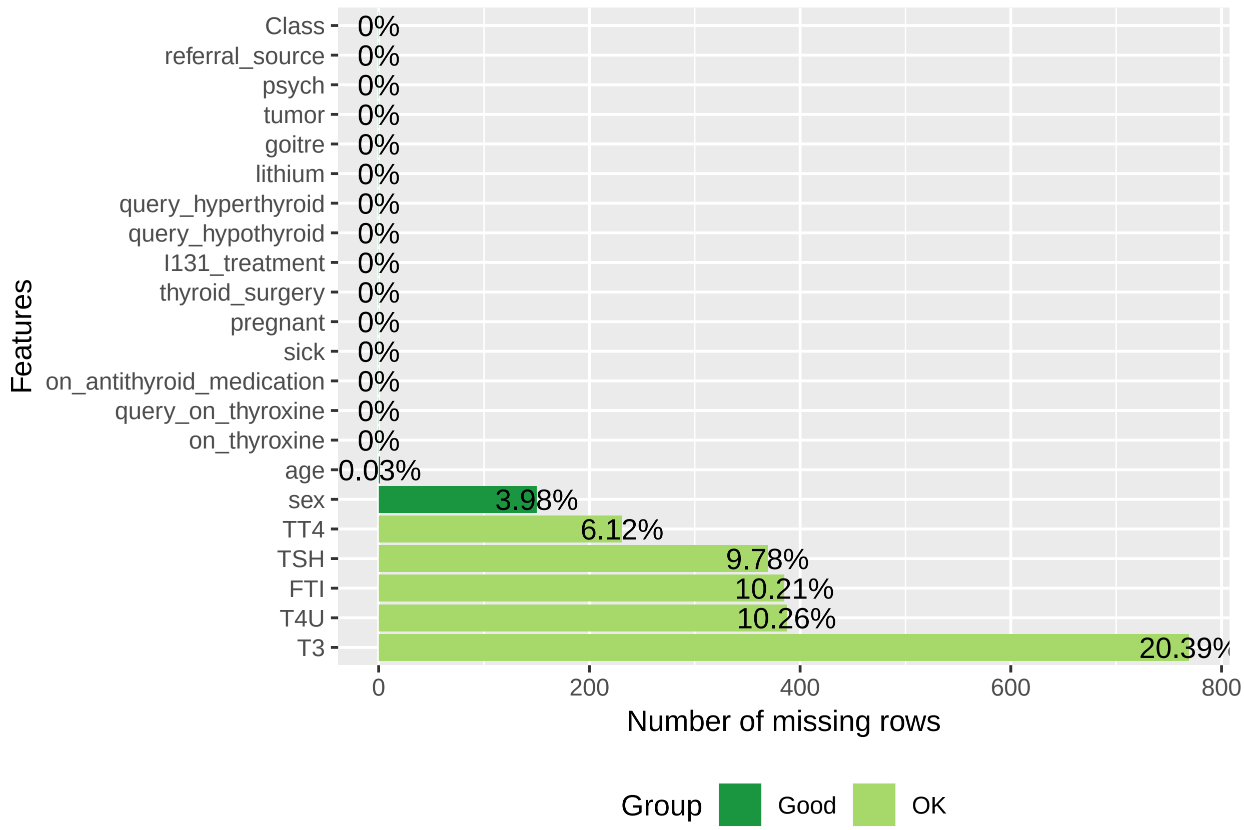
```
sick_tidy <- sick_tidy %>%
  dplyr::select(-hypopituitary)
```

2.3 Missing values reduction

In this section, I will discuss the missing values in the dataset.

2.3.1 Exploration

There are a lot of missing values in the dataset.



However, there is no problem with removing observations with missing values, for the dataset size is very large. Given that we focus on interpretable models, which are generally not complex, there's no need to impute missing values. Such techniques may cause bias in parameter estimation, i.e. lead to the assignment of an inaccurate level of importance to some features.

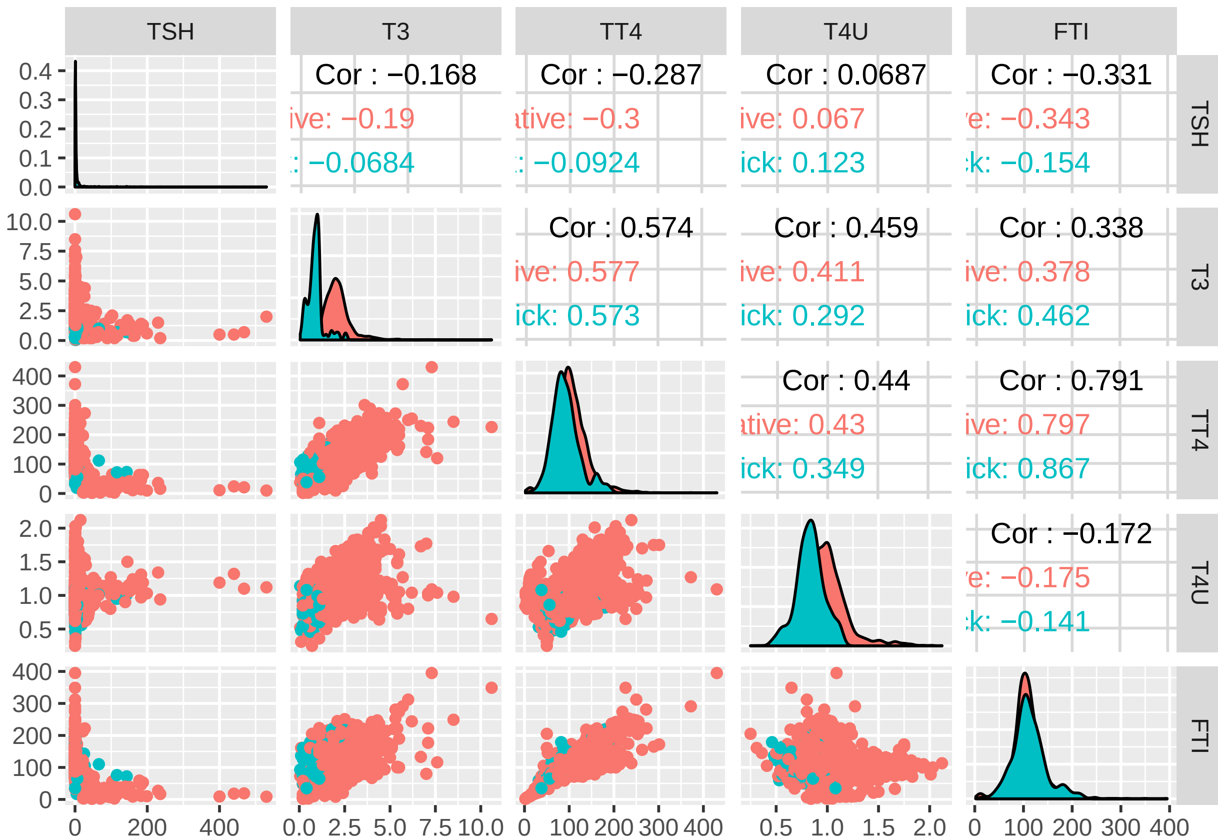
2.4 Solution

```
sick_tidy <- sick_tidy %>% na.exclude()
```

2.5 Skewness reduction

Some of the numeric variables are skewed. I will try to fix that transforming those variables.

2.5.1 Exploration



Skewness of these variables:

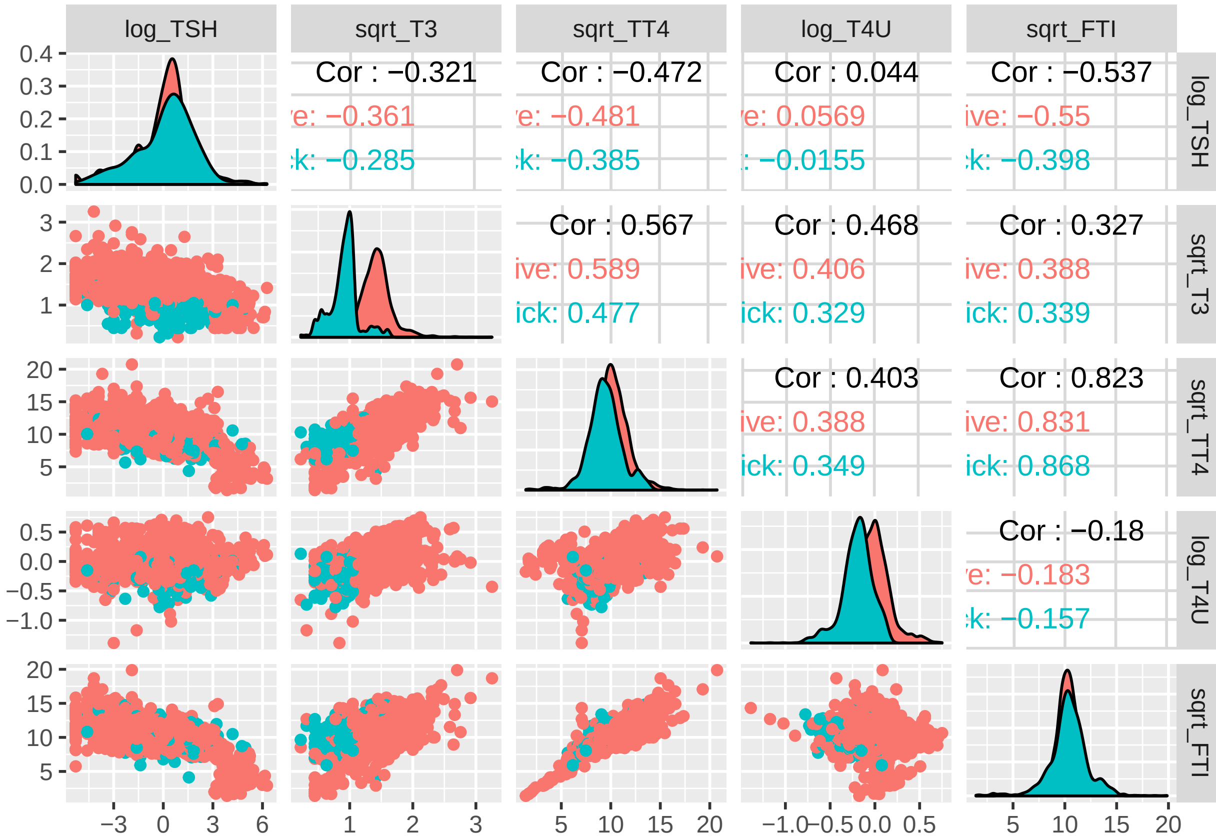
column	skewness	skewness_log	skewness_sqrt
TSH	13.231931	-0.4938557	5.4889020
T3	1.768918	-1.7744449	0.0254847
TT4	1.186118	-2.8400853	-0.2653533
T4U	1.234600	0.0157377	0.6624722
FTI	1.102262	-3.5446807	-0.5991264

As we can see, we can fix the skewness pretty easily, by taking logarithm or square root of these variables.

2.5.2 Solution

```
sick_t <- sick_tidy %>%
  mutate(log_TSH = log(TSH),
         sqrt_T3 = sqrt(T3),
         sqrt_TT4 = sqrt(TT4),
         log_T4U = log(T4U),
         sqrt_FTI = sqrt(FTI)) %>%
  dplyr::select(-TSH, -T3, -TT4, -T4U, -FTI)
after <- ggpairs(sick_t[, 18:22], aes(colour=sick_t$Class))
```

After the transformation:



3 Prediction models

In this section I will compare five different interpretable models:

- naive biases
- logistic regression
- basic tree
- knn

and the winning model.

3.1 Naive Bayes

Let's perform CV resample with naive Bayes.

iter	auc	auprc
1	0.9119293	0.7191105
2	0.8969400	0.5430382
3	0.9576632	0.5556489
4	0.9031330	0.4942390
5	0.9571429	0.6425330

As we can see, naive bayes model performs quite well on auc, but auprc shows a room for improvement.

3.2 Logistic Regression

Results of logistic regression:

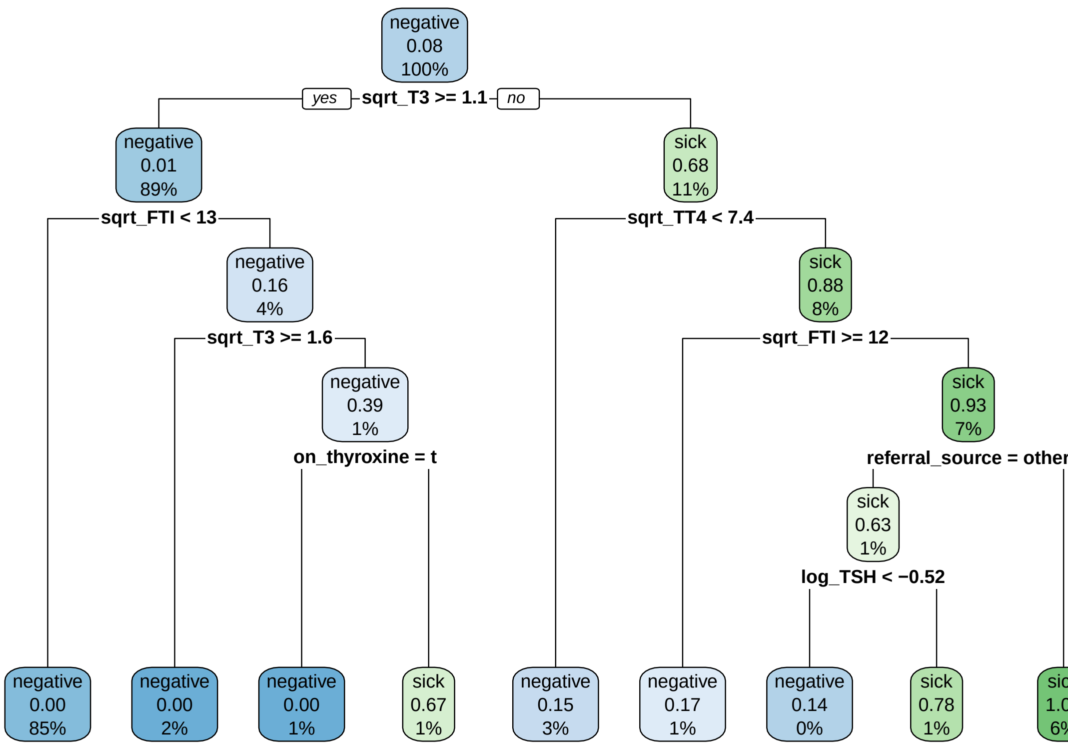
iter	auc	auprc
1	0.9357405	0.7099241
2	0.9397865	0.6735862
3	0.9613065	0.6021301
4	0.9302885	0.5852034
5	0.9680109	0.7453673

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	3.7320495	1.0518184	3.5481880	0.0003879
age	0.0044963	0.0059947	0.7500435	0.4532285
sexM	0.1383344	0.2477745	0.5583077	0.5766343
on_thyroxinet	-0.5013958	0.4754913	-1.0544794	0.2916635
query_on_thyroxinet	-0.3228639	1.1094947	-0.2910009	0.7710506
on_antithyroid_medicationt	-12.8934230	1866.9522545	-0.0069061	0.9944897
sickt	0.6597554	0.4241061	1.5556374	0.1197943
pregnantt	-8.9355534	1281.9846610	-0.0069701	0.9944387
thyroid_surgeryt	-15.6751416	1807.2890681	-0.0086733	0.9930798
I131_treatmentt	0.6623973	1.2268905	0.5398993	0.5892665
query_hypothyroidt	1.0978406	0.4066080	2.6999980	0.0069340
query_hyperthyroidt	-0.0431322	0.6503117	-0.0663255	0.9471187
lithiumt	-15.2966467	2849.8350812	-0.0053676	0.9957173
goitret	0.3530832	1.6397543	0.2153269	0.8295124
tumort	0.7265394	0.9963818	0.7291778	0.4658929
psycht	-0.7970172	0.7913147	-1.0072063	0.3138356
referral_sourceother	-1.2130664	0.5393083	-2.2493005	0.0244934
referral_sourceSVI	0.0051169	0.4912247	0.0104167	0.9916888
referral_sourceSTMW	-13.3549401	982.6208624	-0.0135911	0.9891562
referral_sourceSVHD	-0.2164818	1.0240545	-0.2113968	0.8325777
log_TSH	-0.1416668	0.0804439	-1.7610632	0.0782277
sqrt_T3	-10.5385900	0.7698978	-13.6882978	0.0000000
sqrt_TT4	-0.7191372	0.6945789	-1.0353571	0.3005022
log_T4U	5.0028491	3.3289203	1.5028444	0.1328792
sqrt_FTI	1.3323980	0.6750729	1.9737097	0.0484148

Logistic regression is already much better in both measures. What if this problem is nonlinear?

3.3 Tree

The most basic nonlinear model is regression tree.



Tree results:

iter	auc	auprc
1	0.9700330	0.8465122
2	0.9812224	0.8774844
3	0.9946157	0.8518021
4	0.9123035	0.7494416
5	0.9307222	0.7448206

This basic tree achives astonishing 20% improvement over naive bayes classifier in auprc.

3.4 The KNN

K-nearest neighbors is one of the oldest classifiers. Providing enough data, might make it very robust. Just like other "shallow", nonlinear models, like svm with gaussian kernel for instance, it has a scalibility problem. However, unlike SVM, it's highly interpretable, despite that it doesn't generalise knowledge.

KNN Performance:

iter	auc	auprc
1	0.9767288	0.8775712
2	0.9581340	0.8205113
3	0.9372239	0.7417158
4	0.9189950	0.6489036
5	0.9270808	0.7429218

Performance is good. What if knn was trained only on variables significant in logistic regression and numeric data?

iter	auc	auprc
1	0.9767939	0.8844633
2	0.9609859	0.8592776
3	0.9874918	0.8243306
4	0.9713944	0.7643902
5	0.9774151	0.8833900

Results are a bit better. The KNN model achives about 84% in auprc benchmark, beating all other models. Note that dataset size is large enough, for it work good - there are about 2600 observations.

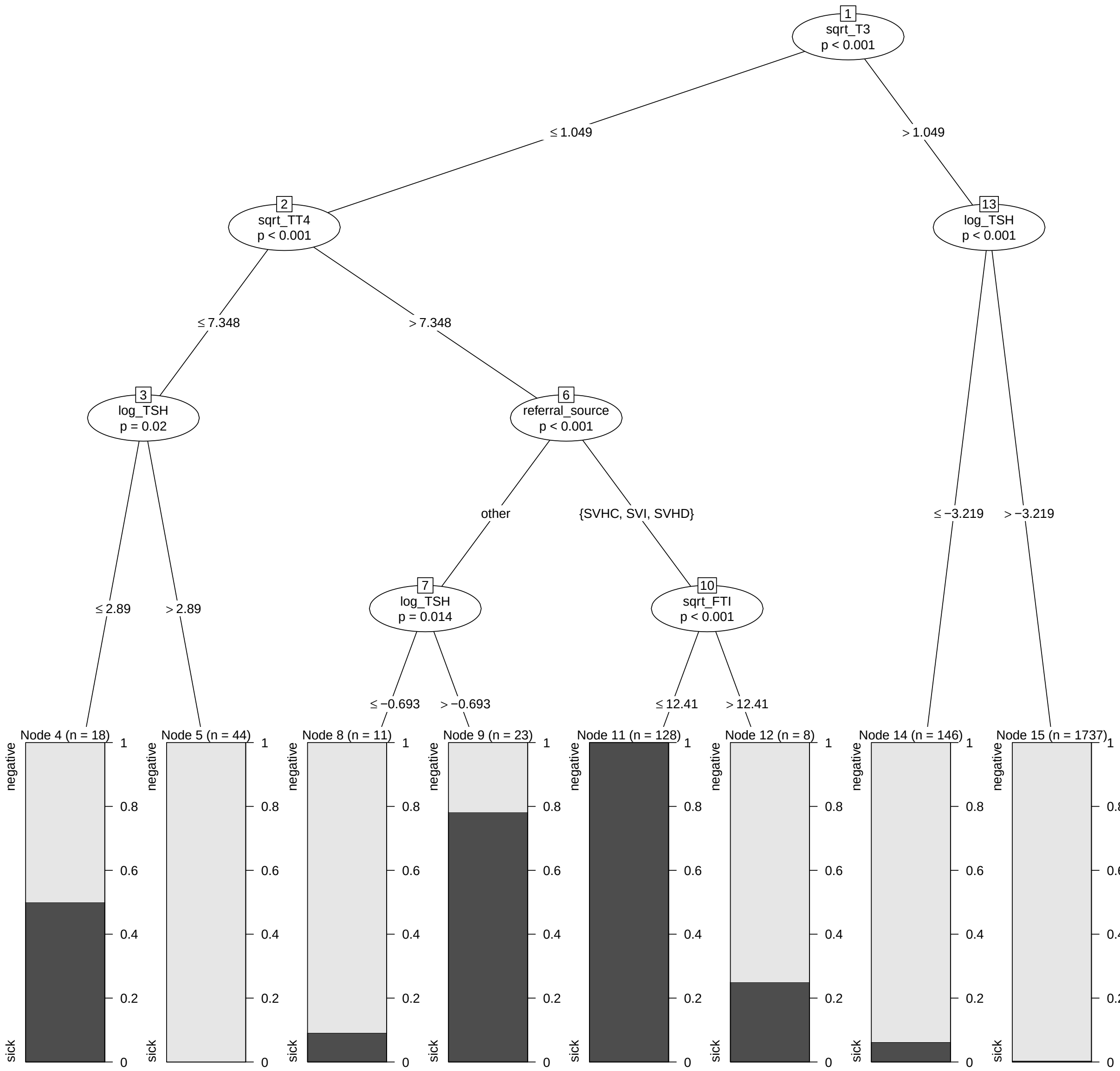
Can this result be improved? KNN relies on data, and we deleted about a third of the dataset. Let's interpolate missing values and check, if it improves performance. It's worth noticing, that test set should only include observations without missing values, since they are more trustworthy.

KNN Performance after imputation:

iter	auc.auc	auprc
1	0.9272176	0.7582913
2	0.9360902	0.7302373
3	0.8731209	0.6056126
4	0.9103994	0.7329072
5	0.9276762	0.7857786

As we can see, results are worst. The idea of interpolating data turned out to be not effective.

3.5 C-Tree



C-Tree results:

iter	auc	auprc
1	0.9738651	0.8134677
2	0.9673811	0.9036939
3	0.9697674	0.8532789
4	0.9771819	0.9125367
5	0.9218807	0.8485409

Results are a lot better.The tree is not overly complicated. It mostly uses the T3 variable and TSH.

4 Conclusions and the last benchmark

The C-tree model turned out to be the best. Perhaps the KNN could be improved by learning metric, but this goes beyond the scope of this paper.

Here's comparison of different models on a test dataset:

classif.naiveBayes	0.629457663909351
classif.binomial	0.663283802308449
classif.rpart	0.75114430570143
classif.kknn	0.70751523046402
classif.ctree	0.813156423130956
classif.kknn subset	0.800324900769413