Final Homework Report

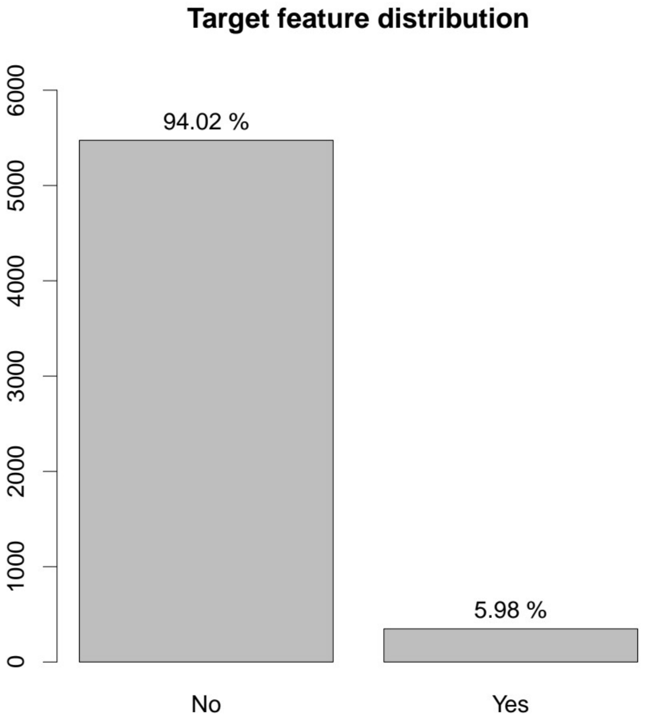
NPFL054 Introduction to Machine Learning

Milan Wikarski

Table of Contents

Task 1 – Data analysis

Dataset consists of 5822 examples. Each example has 86 attributes. The 86-th attribute *Purchase* with values *Yes (1)* and *No (0)* is the target attribute.



The distribution of the target attribute *Purchase* is heavily skewed towards *No*. The frequency of *Yes* is just 5.98%. This means that the expected precision when randomly selecting 100 examples whould be 5.98%.

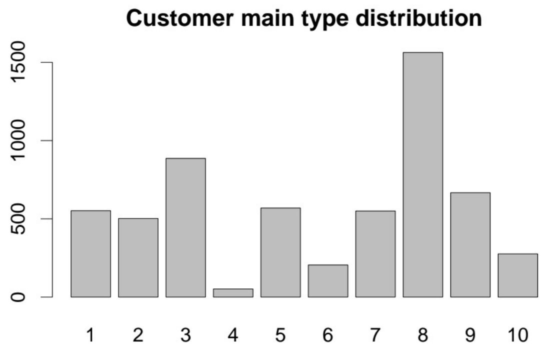
Task 1a

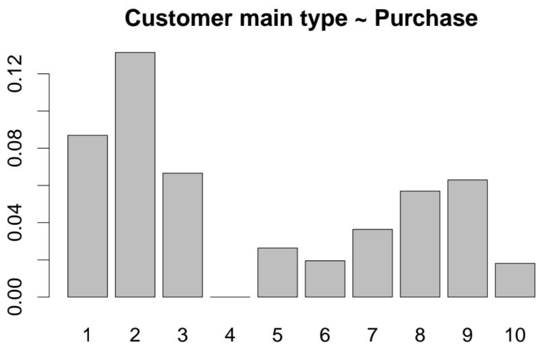
MOSHOODF

Attribute *MOSHOODF* represents the main customer type. This attribute divides customers into 10 groups described in *L2*. The table below lists the number of customers per group, as well as the percentage of examples in this group with target attribute *Purchase* value *Yes (1)* (ie. the percentage of customers who have purchased the caravan insurance policy):

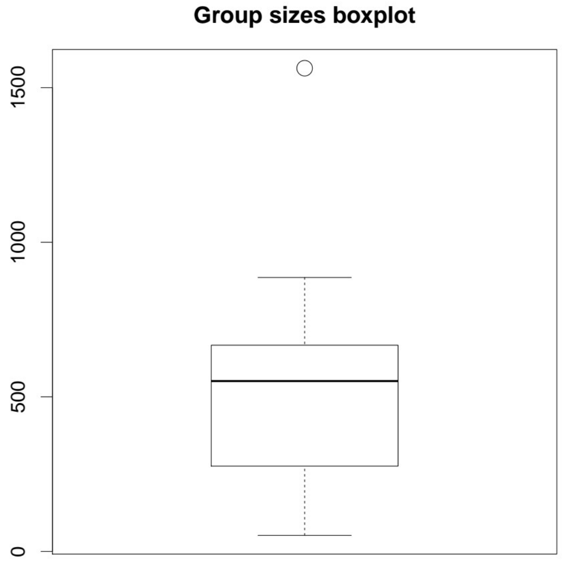
ID	Group	Size	Purchase Frequency
1	Successfull Hedonists	552	8.7%
2	Driven Growers	502	13.15%
3	Average Family	886	6.66%
4	Career Loners	52	0%
5	Living Well	569	2.64%
6	Cruising Seniors	205	1.95%
7	Retired and Religious	550	3.64%
8	Family with Grown ups	1563	5.69%
9	Convervative Families	667	6.3%
10	Farmers	276	1.81%

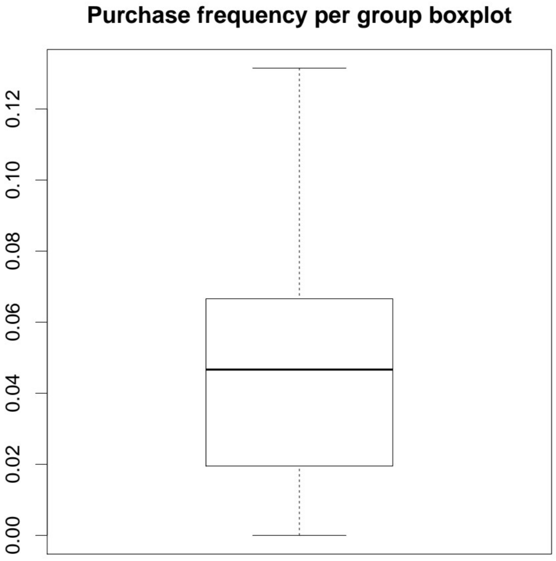
This data can also be plotted into two barcharts:





We can further analyze irregularities in groups by plotting the data into two boxplots and looking for outliers:





Group *4 (Career Loners)* is really small (only 52 members) and contains no positive examples but cannot be classified as an outlier. Group *8 Family with grown ups* is the largest (1563 members), thus can be classified as an outlier base on size, but the *Purchase* frequency of this group is slightly below average (5.69%). When it comes to *Purchase* frequncy in groups, there are no outliers.

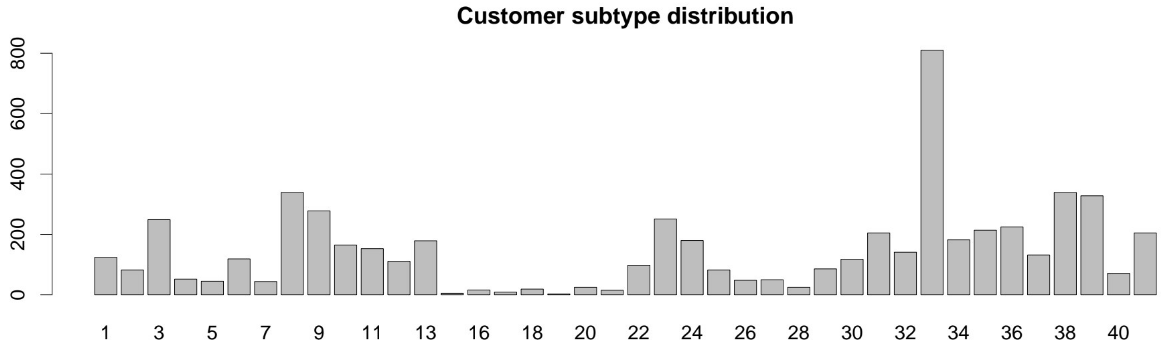
MOSTYPE

Attribute *MOSTYPE* represents the customer subtype. This attribute divides customers into 41 subgroups described in *L0*. The table below lists the number of customers per subgroup, as well as the percentage of examples in this subgroup with the target attribute *Purchase* value *Yes (1)*:

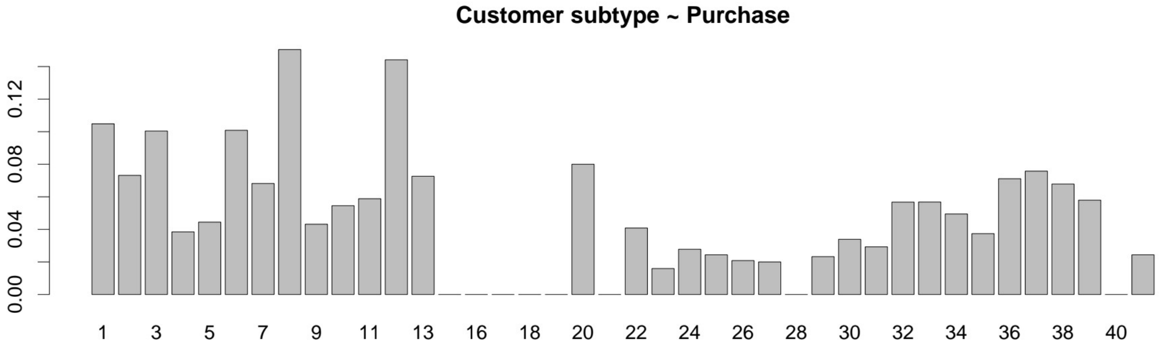
ID	Subgroup	Size	Purchase Frequency
1	High Income	124	10.48%
2	Very Important Provincials	82	7.32%
3	High status seniors	249	10.04%
4	Affluent senior apartments	52	3.85%
5	Mixed seniors	45	4.44%
6	Career and childcare	119	10.08%
7	Dinki's (double income no kids)	44	6.82%
8	Middle class families	339	15.04%
9	Modern	278	4.32%
10	Stable family	165	5.45%
11	Family starters	153	5.88%
12	Affluent young families	111	14.41%
13	Young all american family	179	7.26%
15	Senior cosmopolitans	5	0%
16	Students in apartments	16	0%
17	Fresh masters in the city	9	0%
18	Single youth	19	0%

ID	Subgroup	Size	Purchase Frequency
19	Suburban youth	3	0%
20	Etnically diverse	25	8%
21	Young urban have-nots	15	0%
22	Mixed apartment dwellers	98	4.08%
23	Young and rising	251	1.59%
24	Young	180	2.78%
25	Young seniors in the city	82	2.44%
26	Own home elderly	48	2.08%
27	Seniors in apartments	50	2%
28	Residential elderly	25	0%
29	Porchless seniors: no front yard	86	2.33%
30	Religious elderly singles	118	3.39%
31	Low income catholics	205	2.93%
32	Mixed seniors	141	5.67%
33	Lower class large families	810	5.68%
34	Large family	182	4.95%
35	Village families	214	3.74%
36	Couples with teens 'Married with children'	225	7.11%
37	Mixed small town dwellers	132	7.58%
38	Traditional families	339	6.78%
39	Large religous families	328	5.79%
40	Large family farms	71	0%
41	Mixed rurals	205	2.44%

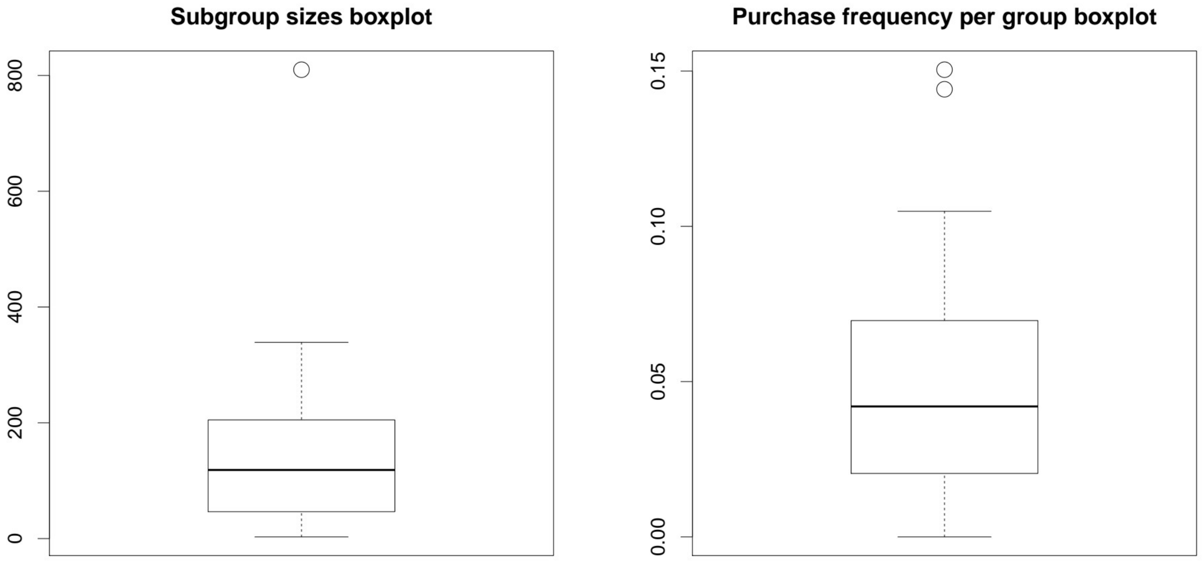
The subgroup sizes can also be visualized using a barchart:



The percentage of people in each subgroup who have purchased the caravan insurance policy can also be plotted to a barchart:



It should be noted that among 5822 examples, there is no representative of subgroup *14 (Junior cosmopolitan)*. To further see, which subgroups might be interessting, we can plot boxplots for size and *Purchase* frequency:



Subgroups *15 – 19* are very small and contain no positive examples but cannot be classified as outliers (the reason for this will be explained in the next chapter). Other subgroups with no positive examples are subgroup *21 Young urban have*-nots, subgroup *28 Residential elderly* and subgroup *40 Large family farms*. Subgroups *21* and *28* are too small to draw any conclusions but there seems to be a correlation between being a member of subgroup *40* and not purchasing the caravan insurance policy.

The largest subgroup *33 Lower class large families* is composed of almost 14% of all examples but the *Purchase* frequency in this subgroup is slightly below the average (5.68%). This subgroup is an outlier based on size.

The most promising subgroups are subgroup *12 Affluent young families* with 111 members and 14.41% Purchase frequency, and subgroup *8 Middle class families* with 339 members and *Purchase* frequency 15.04%, which is the greatest among all subgroups. These two subgroups can be classified as outliers based on *Purchase* frequency.

Task 1b

After some analysis, it can be seen that *MOSHOODF* divides the customers into 10 groups and *MOSTYPE* further divides these customers into subgroups. This information can be gained by calling:

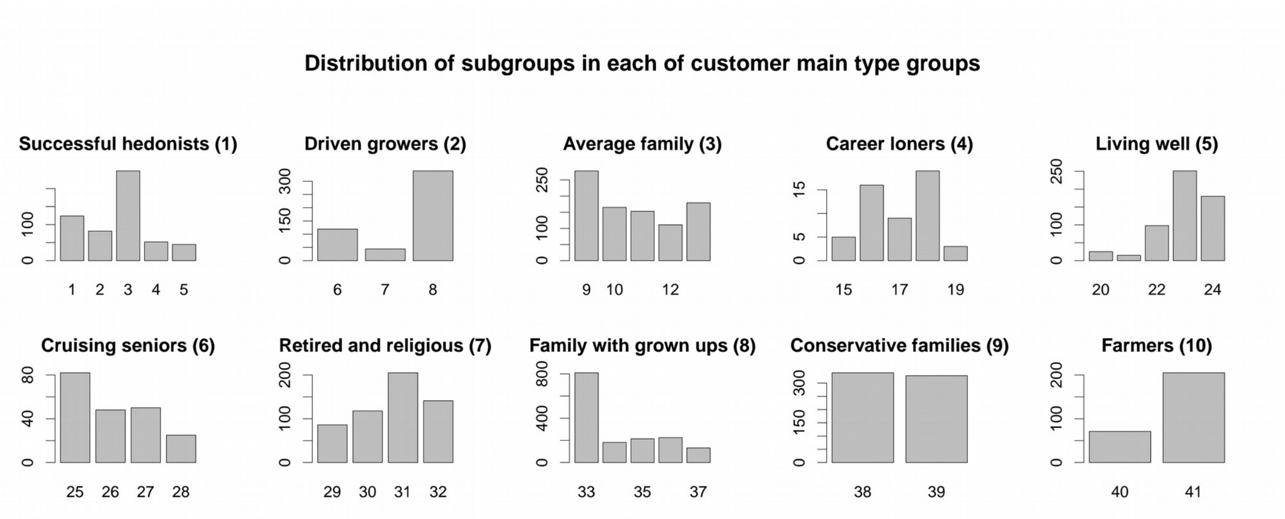
```
table(MOSTYPE, MOSHOOFD)
```

and analyzing the output:

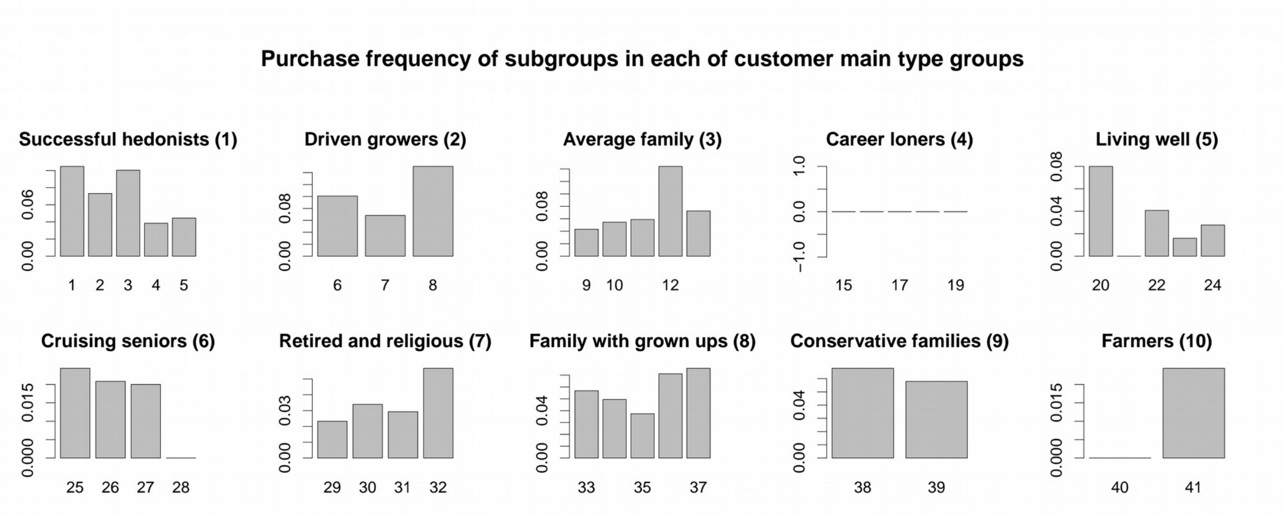
```
       MOSHOOFD
MOSTYPE   1   2   3   4   5   6   7   8   9  10
     1  124   0   0   0   0   0   0   0   0   0
     2   82   0   0   0   0   0   0   0   0   0
     3  249   0   0   0   0   0   0   0   0   0
     4   52   0   0   0   0   0   0   0   0   0
     5   45   0   0   0   0   0   0   0   0   0
     6    0 119   0   0   0   0   0   0   0   0
     7    0  44   0   0   0   0   0   0   0   0
     8    0 339   0   0   0   0   0   0   0   0
     9    0   0 278   0   0   0   0   0   0   0
     10   0   0 165   0   0   0   0   0   0   0
       ...
```

It is clear that all customers of certain subgroup *(MOSTYPE)* belong to just one group (*MOSHOOFD)* and that every customer from each group (*MOSHOOFD)* is assigned one subgroup (*MOSTYPE)*.

With this information, we can explore the groups (*MOSHOOFD)* in more detail by looking at the size of its subgroups:

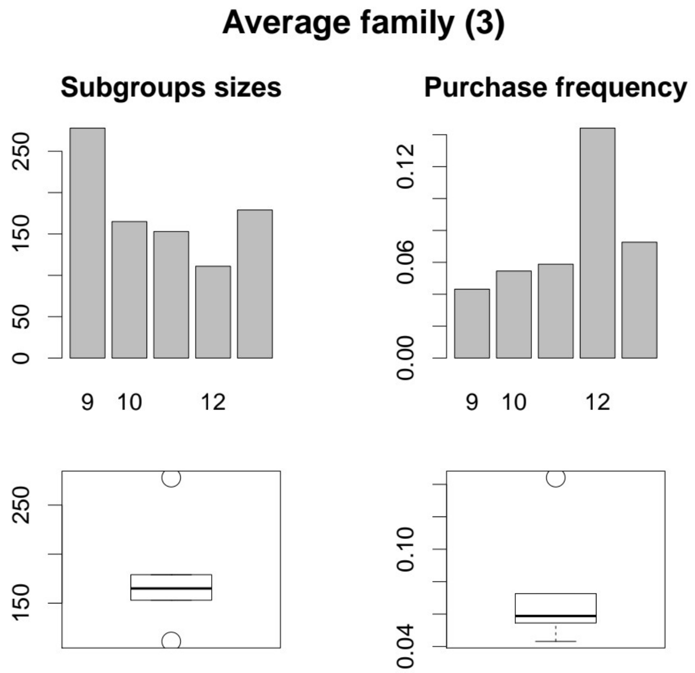


and the *Purchase* frequency of its subgroups:

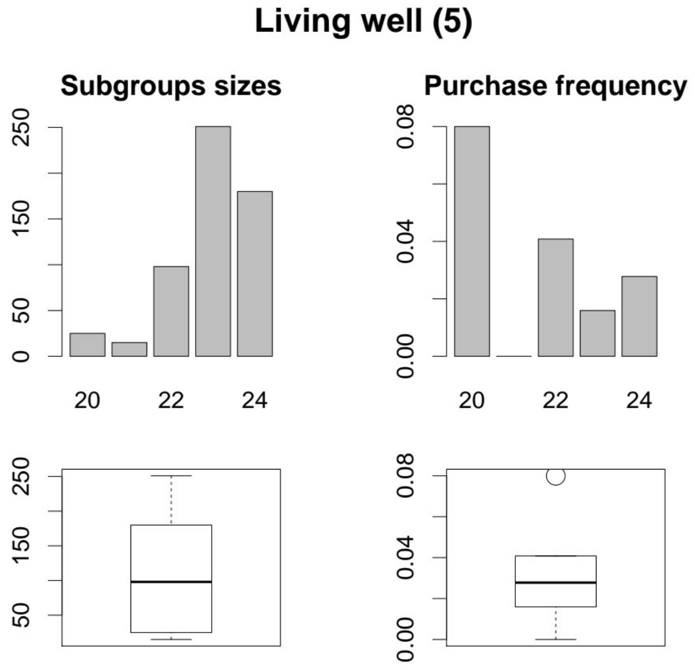


This visuallization is really useful when exlaining why subgroups *15-19* contain no examples. These subgroups are part of group *4 Career loners*, which as was shown in **Task 1a** contains no positive examples.

Groups which are interesting are the ones with irregularities in the *Purchase* distribution. Most notably, the likelihood of the members of group *3 Average family* to purchase the caravan insurance policy is average (6.66%), unless they are part of subgroup *12 Affluent young families*, in which case it is 14.41%.



The same can be said about members of group *5 Living well* and subgroup *20 Ethnically diverse*. Although this subgroup can be classified as an outlier compared to other subgrous in the parent group (based on *Purchase* frequency), it contains only 25 members which drastically decreases its importance.



Charts like these two (group details) were created for each group and can be seen in */out/group-<group number>-detail.pdf*.

Task 2 – Model fitting, optimization and selection

Three models were fitted, optimized and compared: *Decision Tree (2a)*, *Random Forest (2b)* and *Regularized Logistic Regression (2c)*.

Each model was fitted and optimized using a similar method: model performance with certain parameters was measured by computing the mean of $AUC_{0.2}$ gathered by performing a 10-fold cross-validation. Parameters which yielded the best performance were selected.

Afterwads, models with best parameters were trained using the train data set and evaluated using the test data set. The model which produced the best results (again, evaluated using $AUC_{0.2}$) was selected to be the final model.

$AUC_{0.2}$

Bla Bla Bla

Cross-validation

It is important to note that cross-validation was regularized by ensuring that same number of positive examples was in each fold. For this purpuse a custom function *cv.split.safe()* was defined.

Parameter tuning

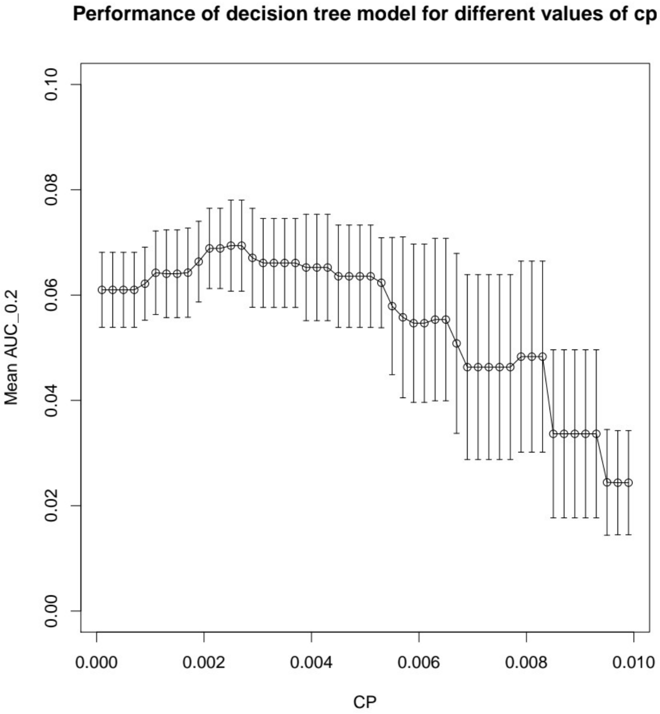
Parameters for a certain model are selected by creating a large number of models, one for each value from a predefined range and then running a 10-fold cross-validation (thus creating and testing 10 models; one per iteration). This is computationally demaning and requires a lot of time. To make sure that this process does not have to run every time the R script is executed, data from parameter optimation process is stored in *.csv* files (eg. *out/decision-tree/decision-tree-eval.csv*) and later loaded into R rather than computed from scratch. This is the way the script behaves only if the value of *params.readDataFromFiles* is set to *TRUE*.

Task 2a – Decision Tree

Decision tree is an unstable machine learning algorithm. This means that small differences in training data will result in significant differences in the produced model. Therefore high variance is expected when performing the cross-validation.

Only one parameter was tuned: *cp* – the complexity parameter. c*p* defines the minimal decrease in impurity in order for a split to be performed. 50 values in range *0.0001-0.0099* were tested, with step *0.0002*. Each model, produced by changing the complexity parameter, was evaluated by computing the value of AUC$_{0.2}$.

This means 10 values for each value of *cp*. The mean and confidence intervals can be plotted into a line chart with error bars:



As was predicted, the error bars on the mean of AUC$_{0.2}$ are very large, making it diffucult to select the best value of *cp*.

cp	**AUC$_{0.2}$ mean**	**AUC$_{0.2}$ standard deviation**	**AUC$_{0.2}$ confidence interval**
0.0025	0.0694	0.0121	(0.0607; 0.0780)

The best value of *cp* in this case is **0.0025**.

Task 2b – Random Forest

Random Forest is an ensamble machine learning algorithm which uses bagging to create multiple decision trees and then voting to get the final prediction.

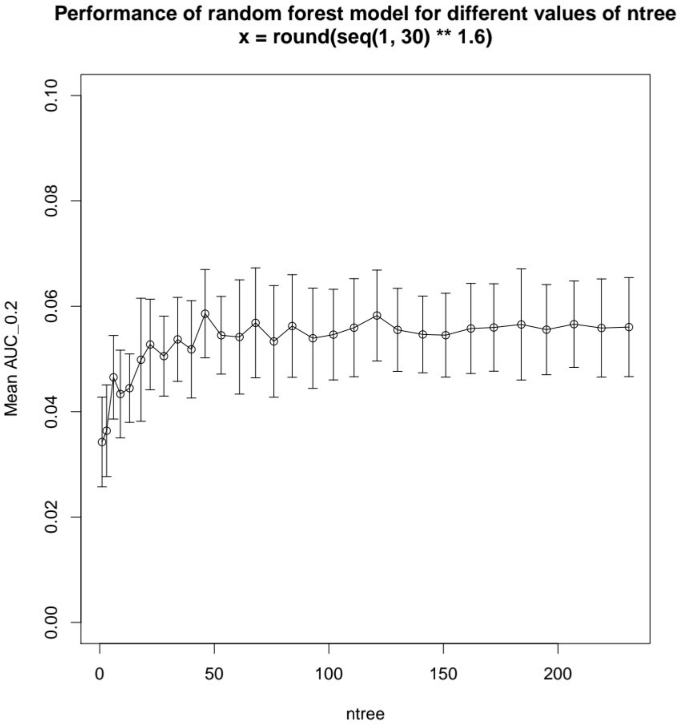
In each node only a subset of features is considered as the split candidates. This results in the more important features appearing more frequently than the less important features. The number of features is controlled by the parameter *mtry*. It is also possible to control the number of trees built using the parameter *ntree*. These two parameters will be tuned.

ntree

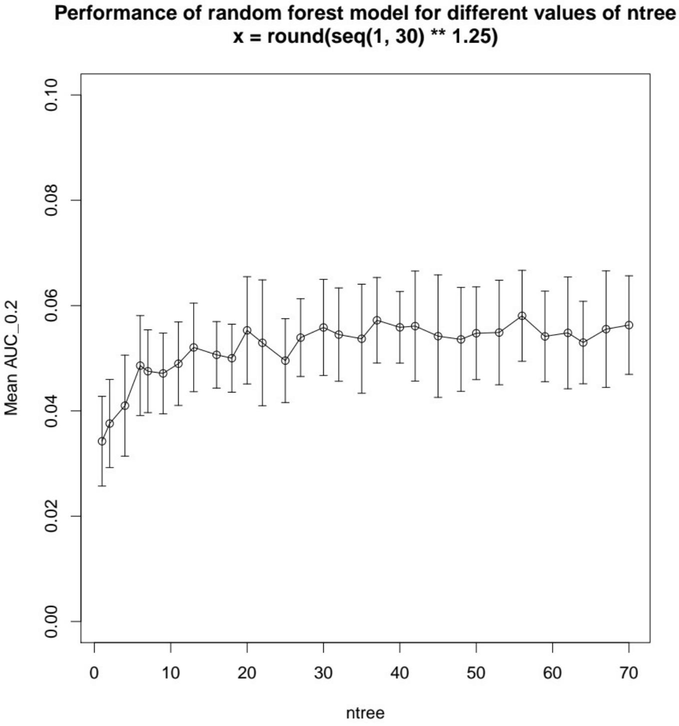
The more trees you grow, the better the model; but there is an upper limit to the performance of a Random Forest. The goal when tuning the *ntree* parameter is to find such value of *ntree* that produces a model which can no longer be improved by increasing the number of trees. We will call this value the *Critical Point*.

Random Trees can never overfit the data, thus finding the optimal value of the *ntree* parameter is more about saving computation time than about classifier performance. Also, it is important to find the minimal value of *ntree* before tuning the value of *mtry*.

The tuning began by testing the values given by *round(seq(1, 30) ** 1.6)* to see if we can find the *Critical Point*. The *mtry* parameter was set to default (9). Here is the chart of AUC$_{0.2}$ means and confidence intervals from 10-fold cross-validation:



From this chart it looks like the *Critical Point* might be somewhere between 50 and 70. To analyze this even further, values given by *round(seq(1, 30) ** 1.25)* were tested. The *mtry* parameter was again set to default (9):



From this chart it is even more clear that it is safe to assume that any value of *ntree* ***>69*** is going to yield good results. The final step was to check if, by any chance, obscurely large values of the *ntree* parameter produced a better model. For this purpuse values 500, 1000 and 2000 were tested. Here are the results:

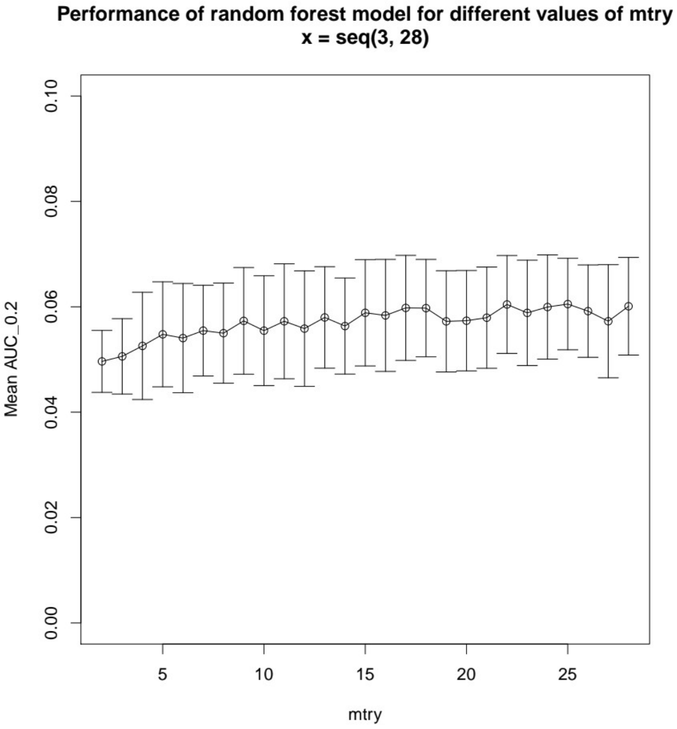
ntree	**AUC$_{0.2}$ mean**	**AUC$_{0.2}$ standard deviation**	**AUC$_{0.2}$ confidence interval**
500	0.0583	0.0123	(0.0495; 0.0670)
1000	0.0577	0.0129	(0.0485; 0.0669)
2000	0.0565	0.0130	(0.0472; 0.0658)

As can be seen, the values of AUC$_{0.2}$ have not improved.

mtry

Now, that it is clear that at least 70 trees (>69) in a Random Forest should be enough, we can start tuning the *mtry* parameter. Lower values of *mtry* require more trees in a forest to be effective, so we will work with n*tree=150*.

Values in range {2, 3, 4, 5, … , 26, 27, 28} have been tested to find the best value:



As can be see from the chart, the performance improves with higher values of *mtry*. Although values >=15 are impossible to compare decisively. Thus the optimal value of the *mtry* parameter is ***15***. Because it is the easiest to compute.

mtry	**AUC$_{0.2}$ mean**	**AUC$_{0.2}$ standard deviation**	**AUC$_{0.2}$ confidence interval**
15	0.0589	0.0141	(0.0488; 0.0690)

Task 2c – Regularized Logistic Regression

Logistic Regression uses sigmoid function to model the probability that an example is positive. The goal in creating a Logistic Regression model is to find the parameters that minimize the loss function. In Regularized Logistic Regression, a penalty is added to the loss function and the goal is to minimize the sum of loss function and penalty function.

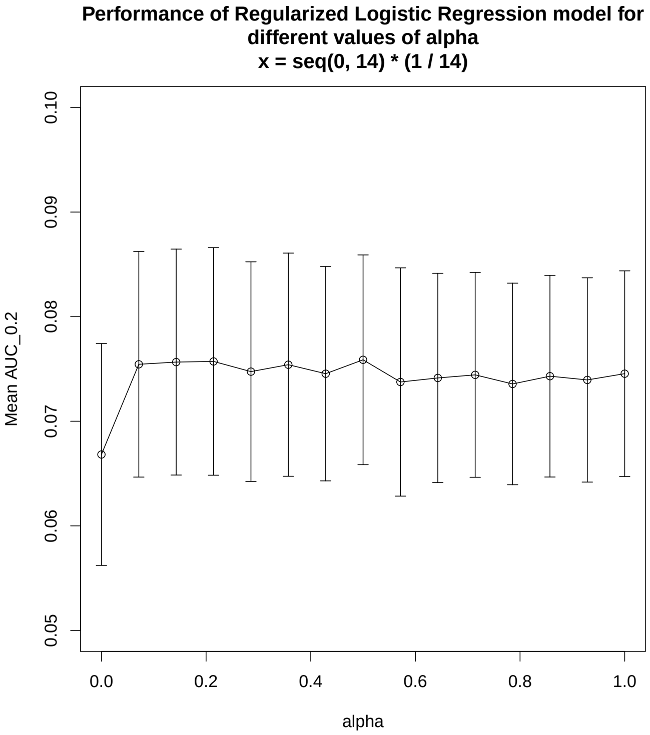
There are two methods for regularization: Lasso and Ridgre Regression. Our model will use a combination of both, which is called an Elastic Net. Elastic Net uses two parameters: *lambda* – the penalty parameter and *alpha* – the parameter which determines the proportion of the penalties of Lasso and Ridge Regression in the final penalty function.

Lambda

The best value of lambda depends on the value of *alpha* and can be easilly calculated using *cv.glmnet* function. This function runs 10-fold cross validation and determines the best value of *lambda* for a given value of *alpha*.

Alpha

Since it is easy to find to best *lambda* for a given *alpha*. Our goal will be to find the best value of *alpha*. For this we will create 15 models with different values of *alpha* given by *seq(0, 14) * (1 / 14)*:



The only value of *alpha* that produces worse model is *alpha=0*. We will go with *alpha=0.0714*

alpha	AUC$_{0.2}$ mean	AUC$_{0.2}$ standard deviation	AUC$_{0.2}$ confidence interval
0.0714	0.0754	0.0151	(0.0647; 0.0862)

Task 2d – Model comparison

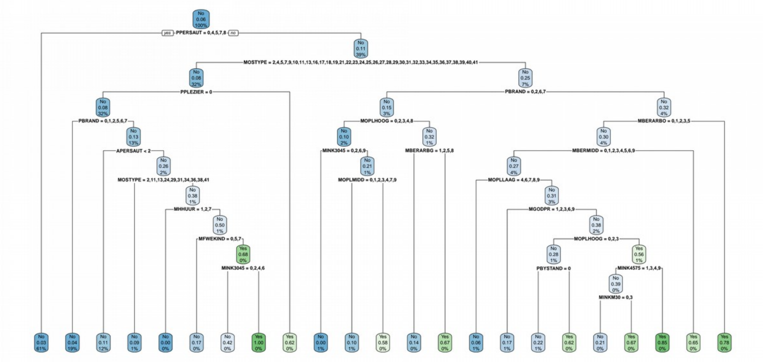
The parameters of the three models were tuned to find the best values. Then the best values were used to produce three candidate models. The best model of the three candidate models will be selected as the final model, which will then be used for final prediction.

Decision Tree

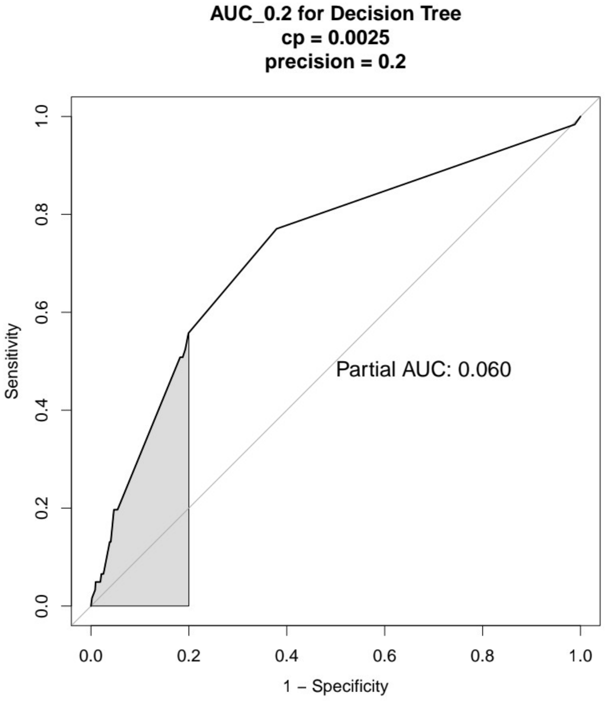
The best parameters for the Decision Tree model are as follows:

cp	0.0025

The tree that has been build using the value of *cp=0.0025* looks like this:



It has been evaluated using an ROC curve. Also, precision when predictng 100 positive examples from 1000 examples in the test data set was calculated:



The Decision Tree model has actually shown to be a pretty good choice for this classification task.

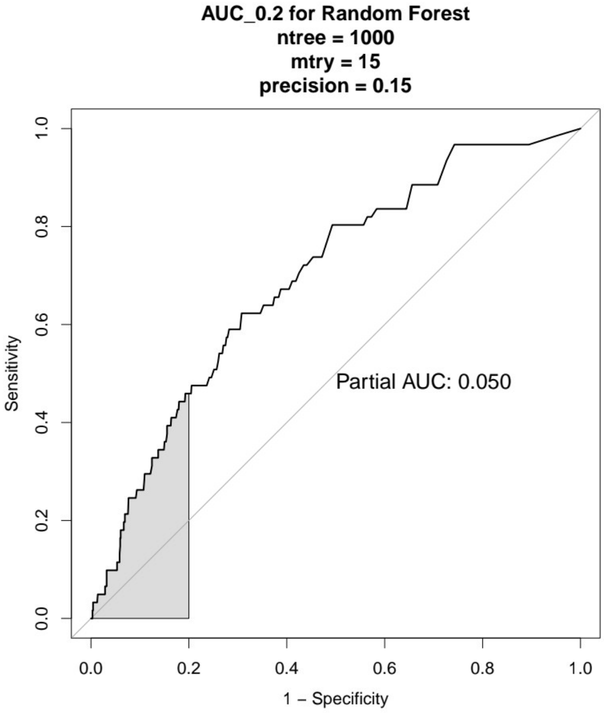
Random Forest

The best parameters for the Random Forest model are as follows:

ntree	>69
mtry	15

We will use the value of *ntree=1000*, because it cannot hurt us and it will make it easier to explore the feature importance measured by the Random Forest.

The Random Forest model with optimal parameters has been evaluated using an ROC curve. Also, precision when predictng 100 positive examples from 1000 examples in the test data set was calculated:



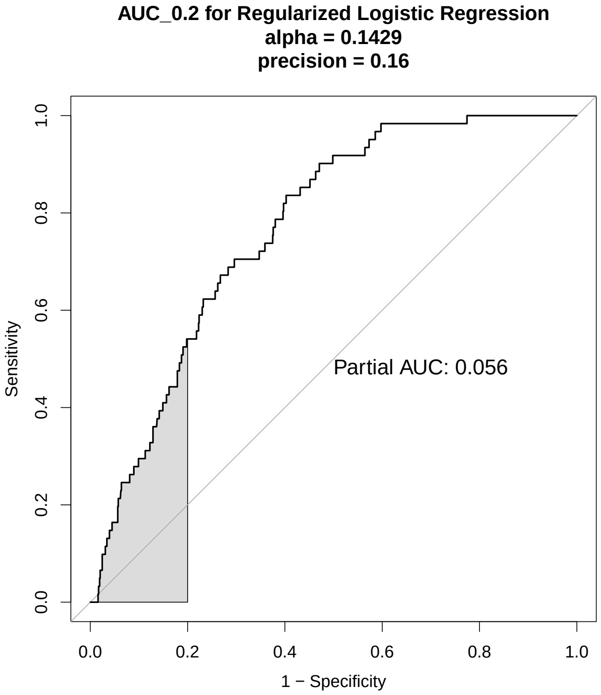
The Random Forest model has suprisingly shown worse results than a single Decision Tree. The reason for this might be that it is really difficult to identify the positive examples and creating an ensamble of very weak classifiers is not going to improve the predictor performance.

Regularized Logistic Regression

The best parameters for the Regularized Logistic Regression model are as follows:

lambda	Based on *alpha*
alpha	0.0714

The Regularized Logistic Regression model with optimal parameters has been evaluated using an ROC curve. Also, precision when predictng 100 positive examples from 1000 examples in the test data set was calculated:

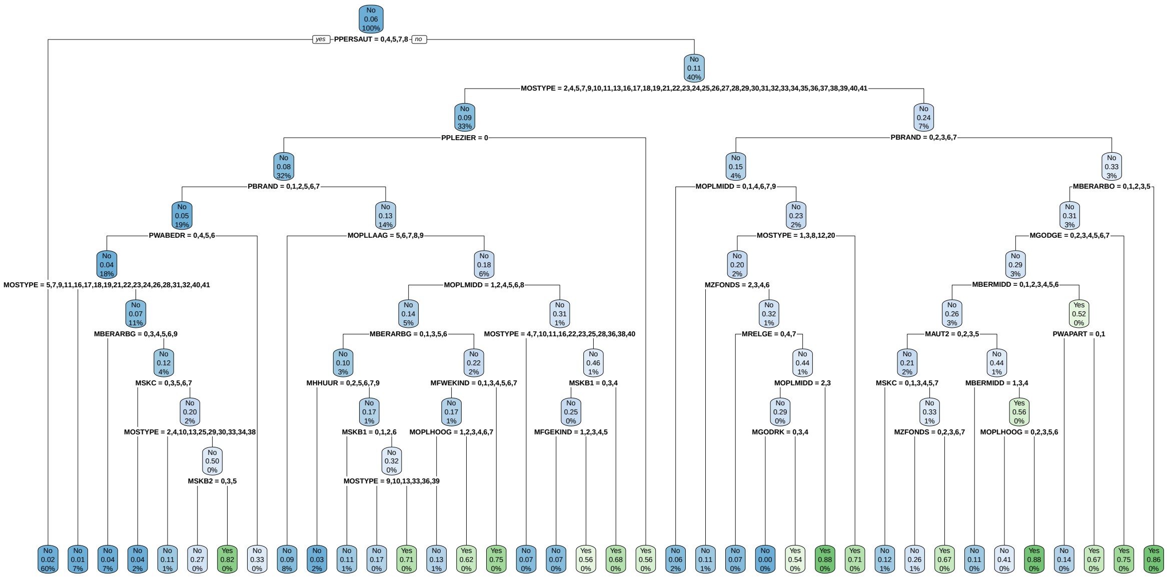


Comparison and final choice

All three models were tuned and tested. The performance is compared by comparing the $AUC_{0.2}$. The model which has performed the best is the Decision Tree (*cp=0.0025*).

Task 2e – The best model

The best model – a Decision Tree (*cp=0.0025*) has been trained on the whole data set. The final Decision Tree is shown below:



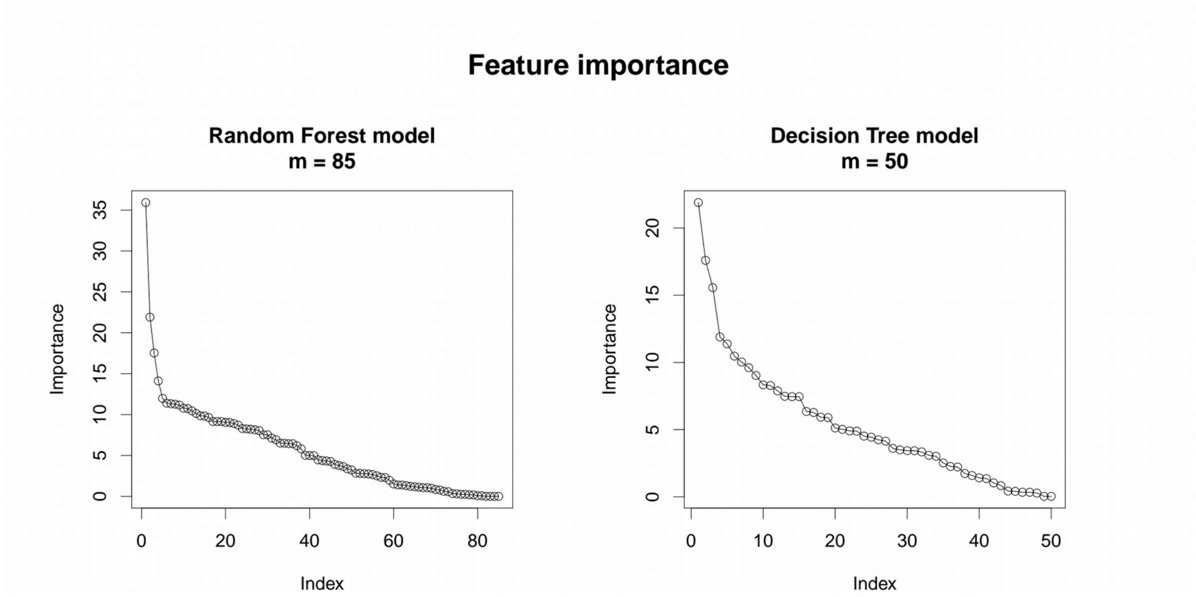
This model will be used to select 100 examples which are most likely to be positive from the blind data set.

Task 3 – Model interpretation and feature selection

There are multiple ways to do feature selection. One way is to train a model using our data and observe the variable importance produced by this model. Decision Tree and Random Forest models can be used this way. Another way is to use the Lasso model which shrinks some parameters to 0 and select a subset of parameters from the ones that were not shrinked to 0.

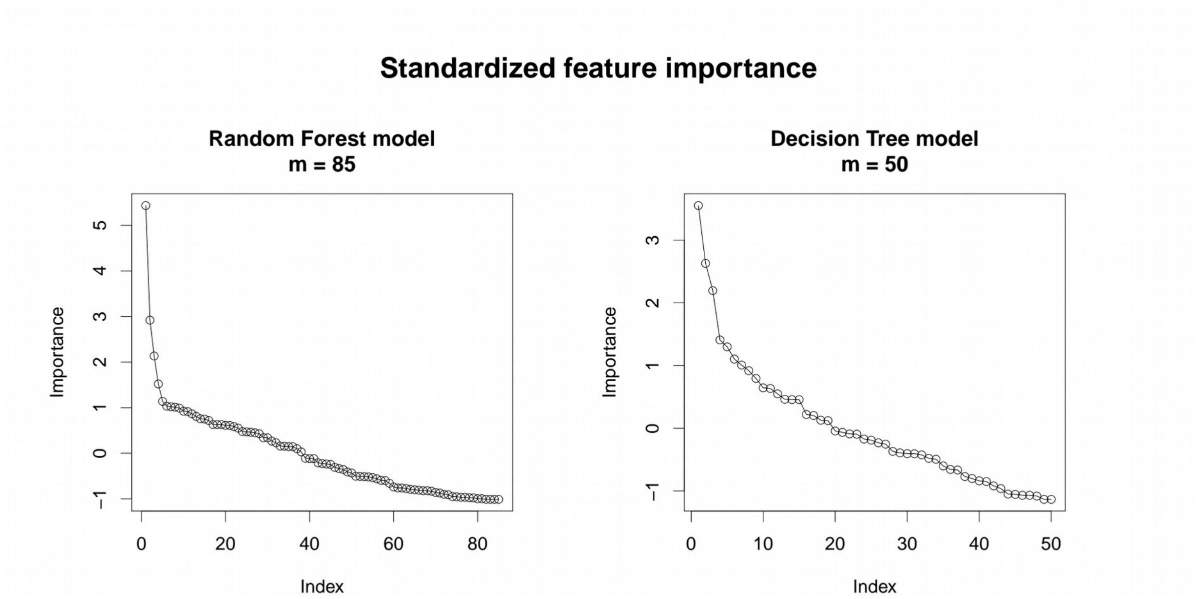
Decision Tree vs Random Forest

Decision Tree (*cp=0.0025*) and Random Forest (*ntree=1000*, *mtry=15*) models were produced in Task 2. We can observe the variable importance of all features based on these models:



where *m* is the number of features used.

According to both models, the by far most important feature is the customer subgroup *MOSTYPE*. The difference is that the Random Forest prioritized MOSTYPE a lot more than the Decision Tree. This can be easilly seen by looking at the standradized version of feature importance (Z-score standardization):



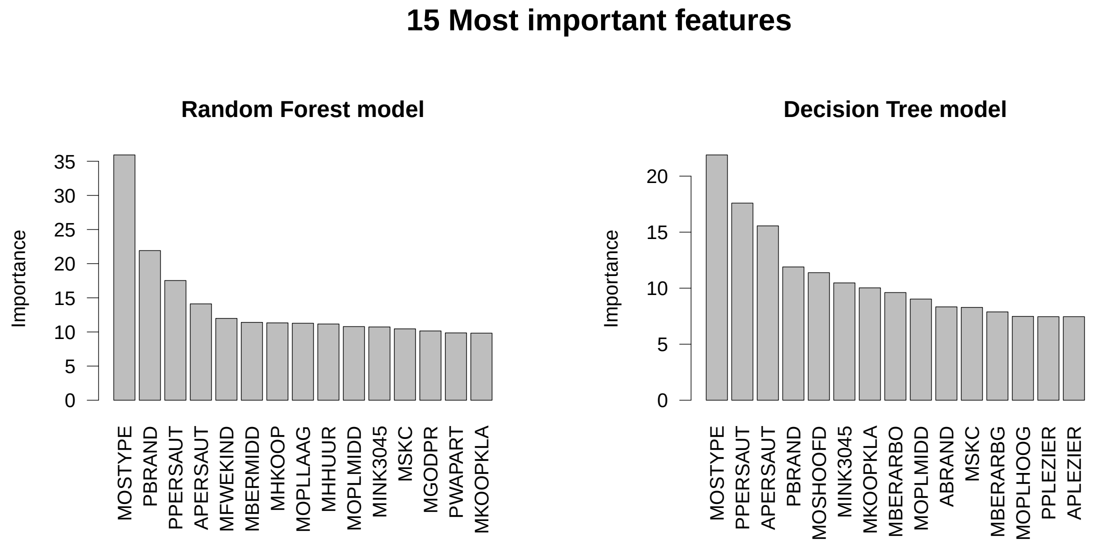
Below are listed features which are above 1 threshold:

Decision Tree	*MOSTYPE, PPERSAUT, APERSAUT, PBRAND, MOSHOOFD, MINK3045, MKOOPKLA*
Random Forest	*MOSTYPE, PBRAND, PPERSAUT, APERSAUT, MFWEKIND, MBERMIDD, MHKOOP, MOPLLAAG*

Four variables appear in both models: *MOSTYPE, PPERSAUT, APERSAUT, PBRAND*. These variables are the most important according to both Decision Tree model and Random Tree model but in different order:

Variable	**Meaning**	**Decision Tree**	**Random Forest**
MOSTYPE	Customer Subtype	21.89	35.93
PPERSAUT	Contribution car policies	17.59	17.53
APERSAUT	Number of car policies	15.56	14.11
PBRAND	Contribution fire policies	11.90	21.92

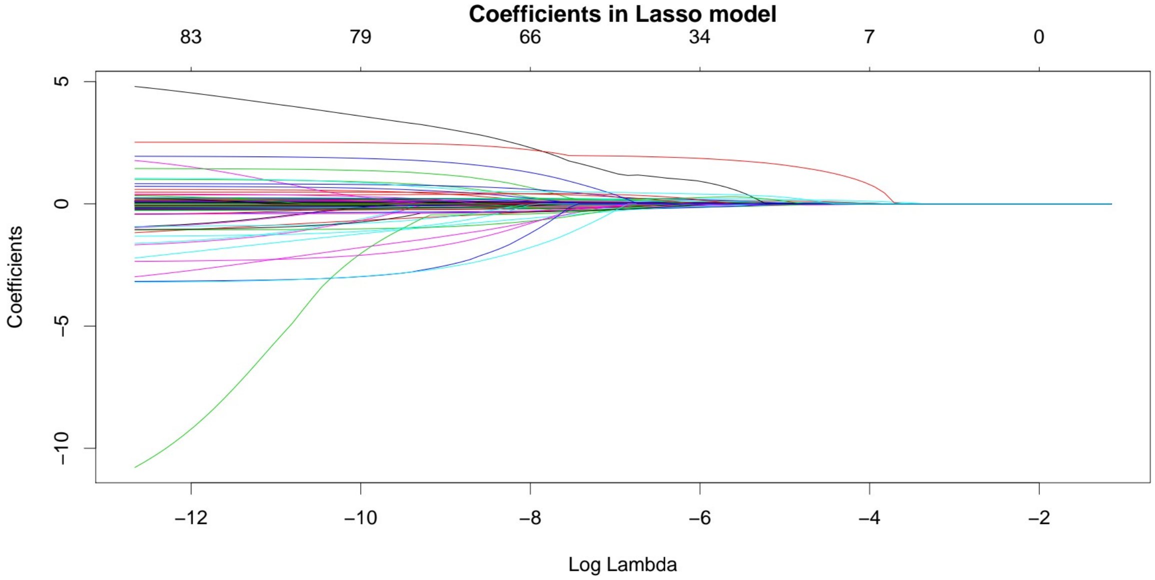
A barchart can be used to see the differences in 15 most important features based on both models:



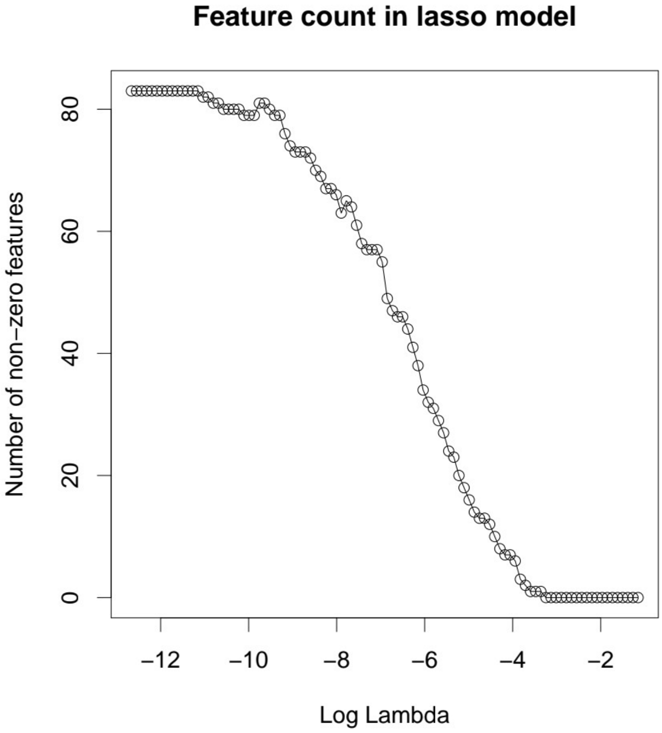
Let’s now compare the feature importance based on the Decision Tree and Random Forest models with the reduced subset of features produced by the Lasso model.

Lasso Model

Lasso regularization is controlled by the *lambda* parameter. The higher the *lambda*, the more strict the model is, meaning that there is a higher penalty. As a result, with higher values of *lambda*, more coefficients are shrinked to 0. This can be visuallized as follows:



We can also observe the number of non-zero features at different values of *lambda*:



It is possible to create a subset of features by selecting a value of *lamba* which will shrink some coefficients to zero.

index	**lambda**	**features**
s30	0.00965713905296605	13
s31	0.00859693086719079	13
s32	0.00765311754650134	14
s33	**0.00681292069057961**	**16**
s34	0.0060649647746946	18

index	lambda	features
s35	0.0053991231351259	20
s36	0.00480638086306439	23
s37	0.00427871275069427	24
s38	0.00380897463695407	27
s39	0.00339080668189405	29
s40	0.00301854726004992	31

Let’s look at the 16 features produced by the Lasso model when setting *lambda* to **0.0068129**. These features are: *MRELGE, MOPLHOOG, MOPLLAAG, MBERBOER, MHHUUR, MAUT1, MINKGEM, MKOOPKLA, PWAPART, PPERSAUT, PBRAND, APLEZIER, AFIETS, ABYSTAND*.

Task 4 – Final prediction on the blind test set

Final prediction on the blind test set was done using the best model (see Task 2e) with the best parameters. The goal is to be as precise as possible, ie. identify as many true positives as possible from 100 examples marked as positive.

4 examples contained yet unseen levels of factors for some features. These examples were marked as ignored – the values of the features were changed to the most frequent level in data and it was checked that all of them are classified as “No”.