In [74]:

```
using CSV, JuMP, Gurobi, DataFrames, StatsBase
```

In [12]:

```
utilities = CSV.read("utilities_scenario_s1_d1.csv", header = true);
first(utilities, 2)
```

Out[12]:

2 rows × 57 columns (omitted printing of 55 columns)

	Reduced.relapse...No.adverse.effect	Reduced.relapse...Non.serious.adverse.effect.only
	Float64	Float64
1	0.856235	0.826698
2	0.963721	0.909316

In [27]:

```
utilities.best_treatmemt = categorical(utilities.best_treatmemt );
X = utilities[:, 1:56]
y = utilities[:, 57]
(train_X, train_y), (test_X, test_y) = IAI.split_data(:classification, X, y,
                                                      seed=1);
```

In [35]:

```
train = deepcopy(train_X)
test = deepcopy(test_X)
train[:, 57] = train_y
test[:, 57] = test_y;
```

```
┌ Warning: `setindex!(df::DataFrame, v::AbstractVector, ::Colon, col
_ind::ColumnIndex)` is deprecated, use `begin
│     df[!, col_ind] = v
│     df
│ end` instead.
│   caller = top-level scope at In[35]:3
└ @ Core In[35]:3
┌ Warning: `setindex!(df::DataFrame, v::AbstractVector, ::Colon, col
_ind::ColumnIndex)` is deprecated, use `begin
│     df[!, col_ind] = v
│     df
│ end` instead.
│   caller = top-level scope at In[35]:4
└ @ Core In[35]:4
```

In [51]:

```
rename!(train, Dict(:x57 => :best_treatment))
first(train[:, 56:57],2)
```

Out[51]:

2 rows × 2 columns

	Standard.relapse...Life.threatening.ventricular.arrhythmia.cardiac.arrest	best_treatment
	Float64	Categorical...
1	0.491402	2
2	0.383616	3

In [53]:

```
rename!(test, Dict(:x57 => :best_treatment))
first(test[:, 56:57],2)
```

Out[53]:

2 rows × 2 columns

	Standard.relapse...Life.threatening.ventricular.arrhythmia.cardiac.arrest	best_treatment
	Float64	Categorical...
1	0.605102	3
2	0.597033	3

In [54]:

```
CSV.write("utilities_s1_d1_train.csv",train)
```

Out[54]:

```
"utilities_s1_d1_train.csv"
```

In [55]:

```
CSV.write("utilities_s1_d1_test.csv",test)
```

Out[55]:

```
"utilities_s1_d1_test.csv"
```

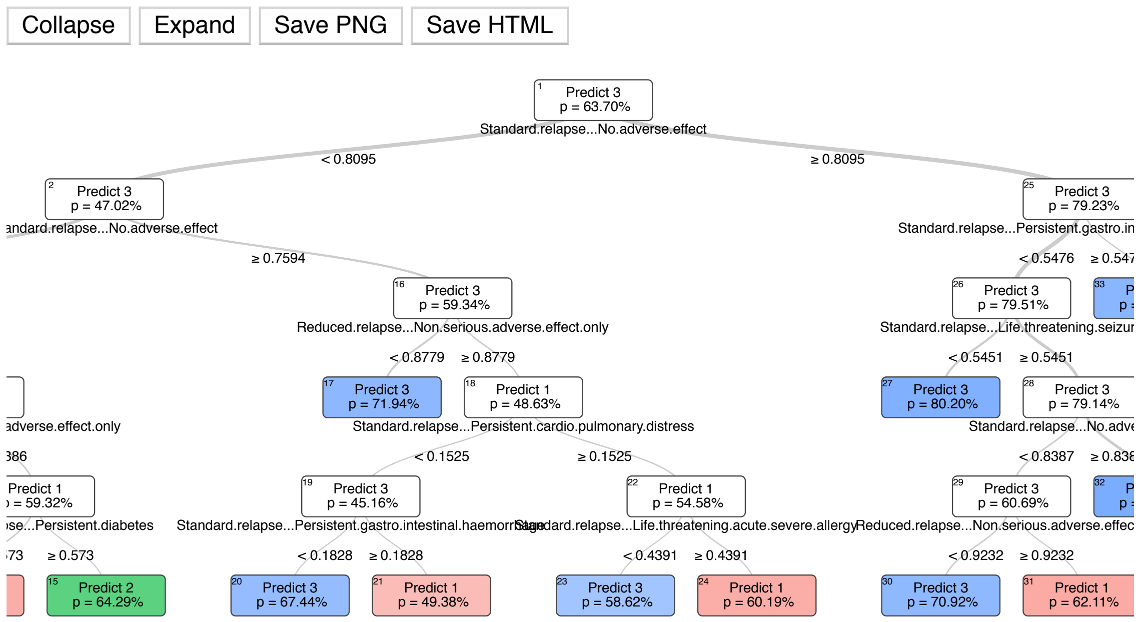
Optimal classification trees

In [37]:

```
lnr = IAI.OptimalTreeClassifier(random_seed=1,max_depth=5, cp=0.001, minbucket=1
0)
IAI.fit!(lnr,train_X,train_y)
```

```
┌ Warning: This copy of Interpretable AI software is for academic pu
rposes only and not for commercial use.
└ @ IAIBase /Users/iai/builds/InterpretableAI/SysImgBuilder/.julia/p
ackages/IAIBase/ymcNn/src/precompile.jl:19
Training trees...100%|████████████████████████████████████████| Time: 0:0
2:00
```

Out[37]:



Cross-vlaidation

We do this twice because we noticed the minbucket picked the first time was at the boundary.

In [1]:

```
grid = IAI.GridSearch(lnr,
    max_depth=3:10,
    minbucket=[10,15,20,25]
)
IAI.fit!(grid, train_X, train_y)
```

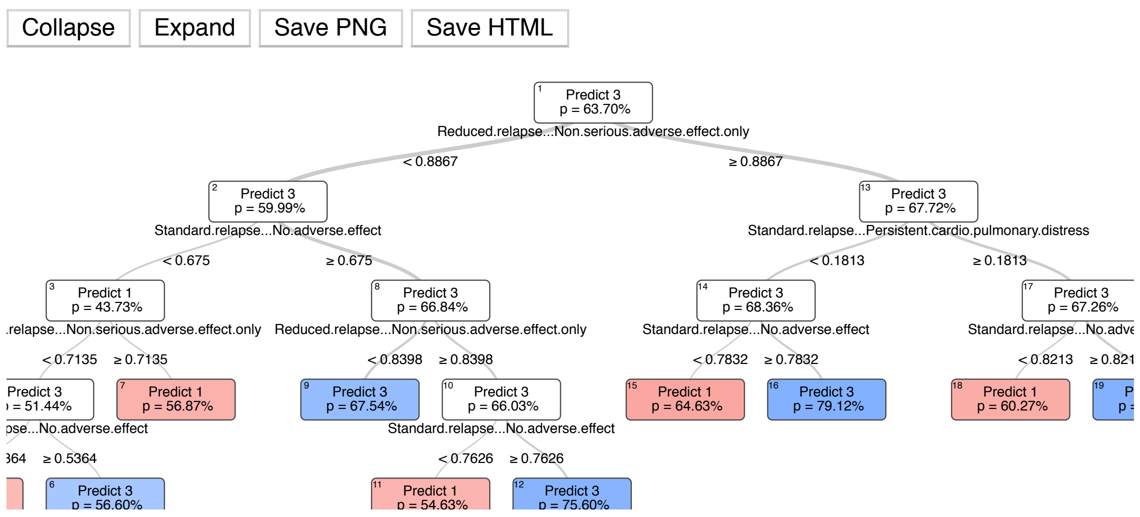
In [2]:

```
grid = IAI.GridSearch(lnr,
    max_depth=3:5,
    minbucket=[25,35,40,45,50]
)
IAI.fit!(grid, train_X, train_y)
```

In [41]:

```
lnr = IAI.get_learner(grid)
```

Out[41]:



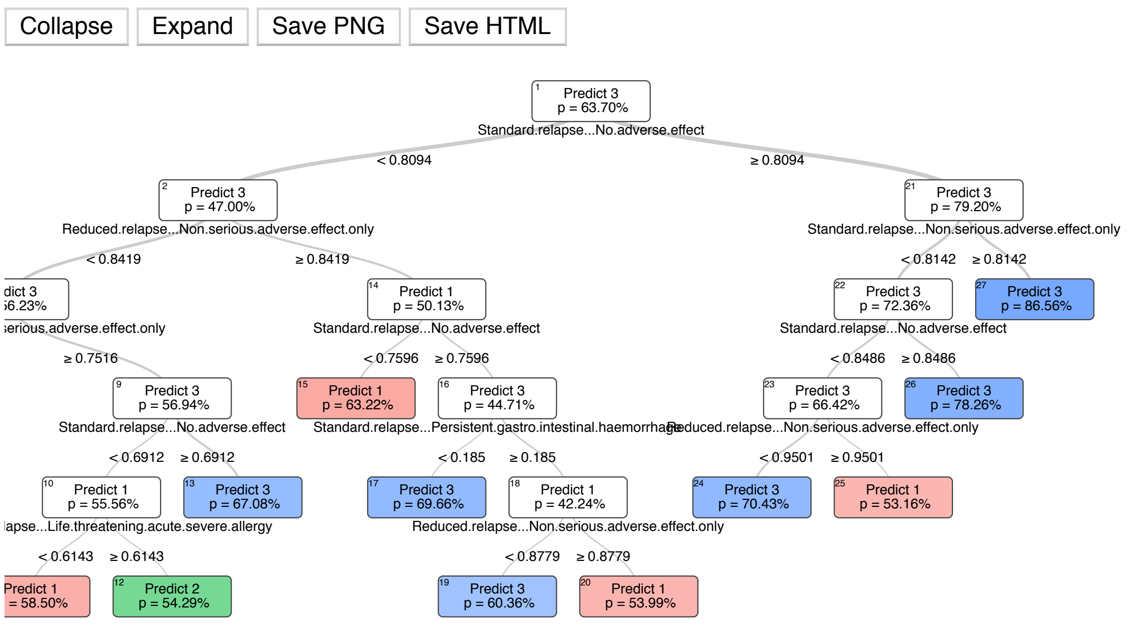
The cross-validated tree never predicts class = 2. Hence we make another tree.

In [59]:

```
lnr2 = IAI.OptimalTreeClassifier(random_seed=1,max_depth=5, cp=0.001, minbucket=
35)
IAI.fit!(lnr2,train_X,train_y)
```

```
Training trees...100%|████████████████████████████████████████| Time: 0:0
1:29
```

Out[59]:



Evaluating the models

In [67]:

```
println("OCT Cross-validated missclassification: ", IAI.score(lnr,test_X,test_y,
criterion=:misclassification))
println("OCT 2 missclassification: ", IAI.score(lnr2,test_X,test_y,criterion=:mi
sclassification))
```

```
OCT Cross-validated missclassification: 0.7076666666666667
OCT 2 missclassification: 0.7023333333333333
```

In [69]:

```
first(IAI.variable_importance(lnr), 10)
```

Out[69]:

10 rows × 2 columns

	Feature	Importance
	Symbol	Float64
1	Standard.relapse...No.adverse.effect	0.559661
2	Reduced.relapse...Non.serious.adverse.effect.only	0.4257
3	Standard.relapse...Persistent.gastro.intestinal.haemorrhage	0.00423166
4	Standard.relapse...Persistent.cardio.pulmonary.distress	0.00282931
5	Standard.relapse...Non.serious.adverse.effect.only	0.00243566
6	Standard.relapse...Life.threatening.acute.severe.allergy	0.00216503
7	Reduced.relapse...No.adverse.effect	0.00204202
8	Standard.relapse...Persistent.diabetes	0.000713477
9	Reduced.relapse...Persistent.diabetes	0.000221424
10	Reduced.relapse...Lethal.acute.severe.allergy	0.0

In [77]:

```
prediction = IAI.predict(lnr2, test_X)
countmap(prediction)
```

Out[77]:

```
Dict{Int64,Int64} with 3 entries:
  2 => 14
  3 => 2410
  1 => 576
```

OCT Hyperplane

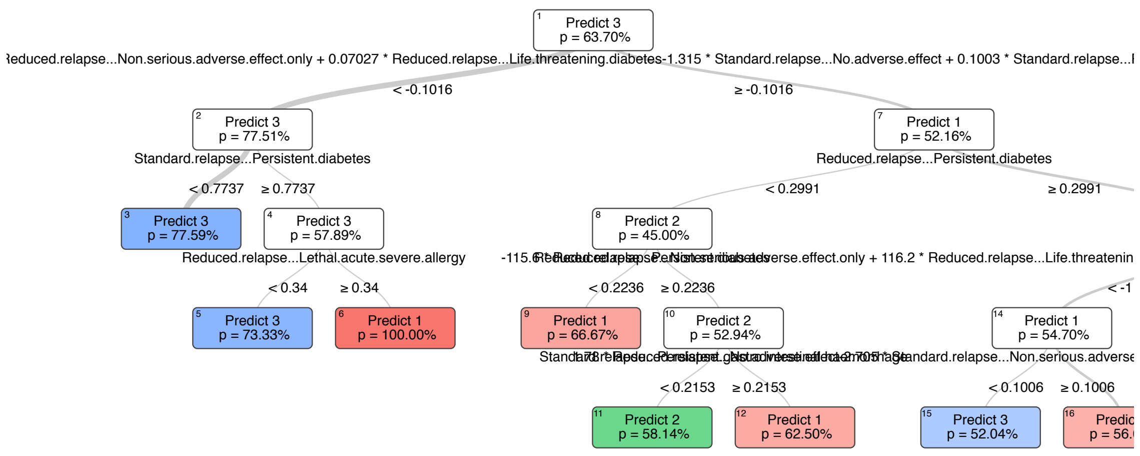
In [78]:

```
lnr_hyper = IAI.OptimalTreeClassifier(
        random_seed=3,
    max_depth = 4,
    cp=0.001,
    hyperplane_config=(sparsity=:all,))
IAI.fit!(lnr_hyper,train_X,train_y)
```

```
Training trees...100%|██████████████████████████████████| Time: 0:5
3:415:51
```

Out[78]:

Collapse Expand Save PNG Save HTML



In [79]:

```
println("OCT Hyperplane missclassification: ", IAI.score(lnr_hyper,test_X,test_y
,criterion=:misclassification))
```

```
OCT Hyperplane missclassification: 0.7033333333333334
```

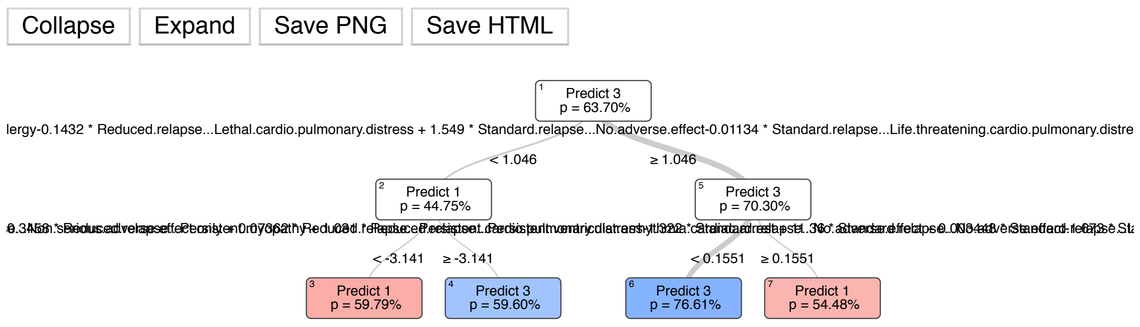
In [3]:

```
grid_h = IAI.GridSearch(lnr_hyper,
    max_depth=2:5,
    minbucket=[20,30,35]
)
IAI.fit!(grid_h, train_X, train_y)
```

In [84]:

```
lnr_hyperplane = IAI.get_learner(grid_h)
```

Out[84]:



In [85]:

```
println("OCT Hyperplane missclassification: ", IAI.score(lnr_hyperplane,test_X,t
est_y,criterion=:misclassification))
```

```
OCT Hyperplane missclassification: 0.7110000000000001
```