```julia
begin
    using Pkg
    Pkg.activate(joinpath(Pkg.devdir(), "MLCourse"))
    using CSV, DataFrames, Distributions, Plots, MLJ, MLJLinearModels, Random,
          Statistics, OpenML
end
```

Linear Methods

We load the precipitation training and test data from a csv file on the harddisk to a DataFrame. Our goal is to predict whether there is some precipitation (rain, snow etc.) on the next day in Pully, getting measurements from different weather stations in Switzerland.

```julia
precipitation_training = CSV.read(joinpath(@__DIR__, "..", "data", "project",
"trainingdata.csv"), DataFrame);
```

```julia
test_data = CSV.read(joinpath(@__DIR__, "..", "data", "project", "testdata.csv"),
DataFrame);
```

First we have to prepare our data set by filling in the missing values with some standard values with the help of `FillImputer` that fills in the median of all values.

```julia
precipitation_training_med = MLJ.transform(fit!(machine(FillImputer(),
                select(precipitation_training, Not(:precipitation_nextday)))),
                select(precipitation_training, Not(:precipitation_nextday)));
```

```julia
precipitation_training_med.precipitation_nextday =
    precipitation_training[:,:precipitation_nextday];
```

```julia
training_data = coerce!(precipitation_training_med, :precipitation_nextday =>
Binary); # with this we tell the computer to interpret the data in column
precipitation_nextday as binary data.
```

Then we standardize our training and test data sets.

```julia
mach_train = machine(Standardizer(features=[:precipitation_nextday,
:ALT_sunshine_4], ignore=true), training_data);
```

```julia
fit!(mach_train);
```

```
stand_train =
```

	ABO_radiation_1	ABO_delta_pressure_1	ABO_air_temp_1	ABO_sunshine_1	ABO_win
1	-0.750348	-0.495005	-1.51099	-0.362023	-0.39032
2	-0.691886	0.517642	0.124061	-0.362023	-0.54831
3	-0.282652	0.155983	0.79466	-0.362023	-0.56856
4	1.19838	0.155983	0.274757	1.64618	-0.65768
5	-0.185215	0.300647	0.746939	-0.362023	0.051220
6	-0.107266	-0.061013	0.704242	-0.362023	0.950524
7	-0.594449	0.0113189	-0.649513	-0.362023	-0.29715
8	0.418892	0.589974	0.458105	-0.362023	1.04369
9	-0.535987	0.155983	-0.767559	-0.362023	-0.41463
10	0.457866	-0.495005	0.0713174	-0.362023	-0.45919
⋮ more					
3176	1.29582	0.445311	1.59586	1.3419	0.193003

```
stand_train = MLJ.transform(mach_train, training_data)
```

```
mach_test = machine(Standardizer(features=[:ZER_sunshine_1, :ABO_sunshine_4,
:ALT_sunshine_4, :CHU_sunshine_4, :SAM_sunshine_4], ignore=true), test_data);
```

```
fit!(mach_test);
```

```
stand_test =
```

	ABO_radiation_1	ABO_delta_pressure_1	ABO_air_temp_1	ABO_sunshine_1	ABO_wir
1	-0.242687	-1.61917	-0.334016	-0.356409	-0.18241
2	-0.768502	-0.264876	-0.856481	-0.356409	-0.26817
3	-0.242687	0.14141	1.5615	-0.356409	0.250288
4	-0.0284655	0.547697	1.4984	-0.356409	-0.48257
5	-0.242687	1.0217	-0.404688	-0.356409	0.164527
6	-0.223212	0.00598145	-1.09626	-0.356409	0.971458
7	-0.534807	0.683126	-0.755522	-0.356409	-0.58003
8	-0.651655	-2.49945	0.948169	-0.356409	-0.25648
9	-0.671129	-1.00974	0.829542	-0.356409	-0.52155
10	3.20433	-0.874307	-0.366828	3.68604	-0.70087
⋮ more					
1200	0.711572	-0.942021	0.647815	-0.356409	-0.63460

```
stand_test = MLJ.transform(mach_test, test_data)
```

Logistic Regression

Lasso regularization

Now we define a supervised learning machine and tune the hyper-parameters. We first try with an interval from 1e-2 to 10 and find a lambda of approximately 4.3.

```
Machine{ProbabilisticTunedModel{Grid,…},…} trained 1 time; caches data
  args:
    1:  Source @249 ⏎ `ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Cont
    2:  Source @702 ⏎ `AbstractVector{ScientificTypesBase.Multiclass{2}}`
```

```
begin
    model = LogisticClassifier(penalty = :l1)
    Random.seed!(10)
    self_tuning_model0 = TunedModel(model = model,
                                    resampling = CV(nfolds = 5),
                                    tuning = Grid(goal = 50),
                                    range = range(model, :lambda,
                                            scale = :log,
                                            lower = 1e-2, upper = 10),
                                            measure = auc)
    self_tuning_mach0 = machine(self_tuning_model0, select(stand_train,
              Not(:precipitation_nextday)),
              stand_train.precipitation_nextday)  |> fit!
end
```

```
measure = [AreaUnderCurve()], measurement = [0.924738], per_fold = [[⋯ more]]), history =
```

```
rep0 = report(self_tuning_mach0)
```

We decided to reduce our interval to find the best lambda.

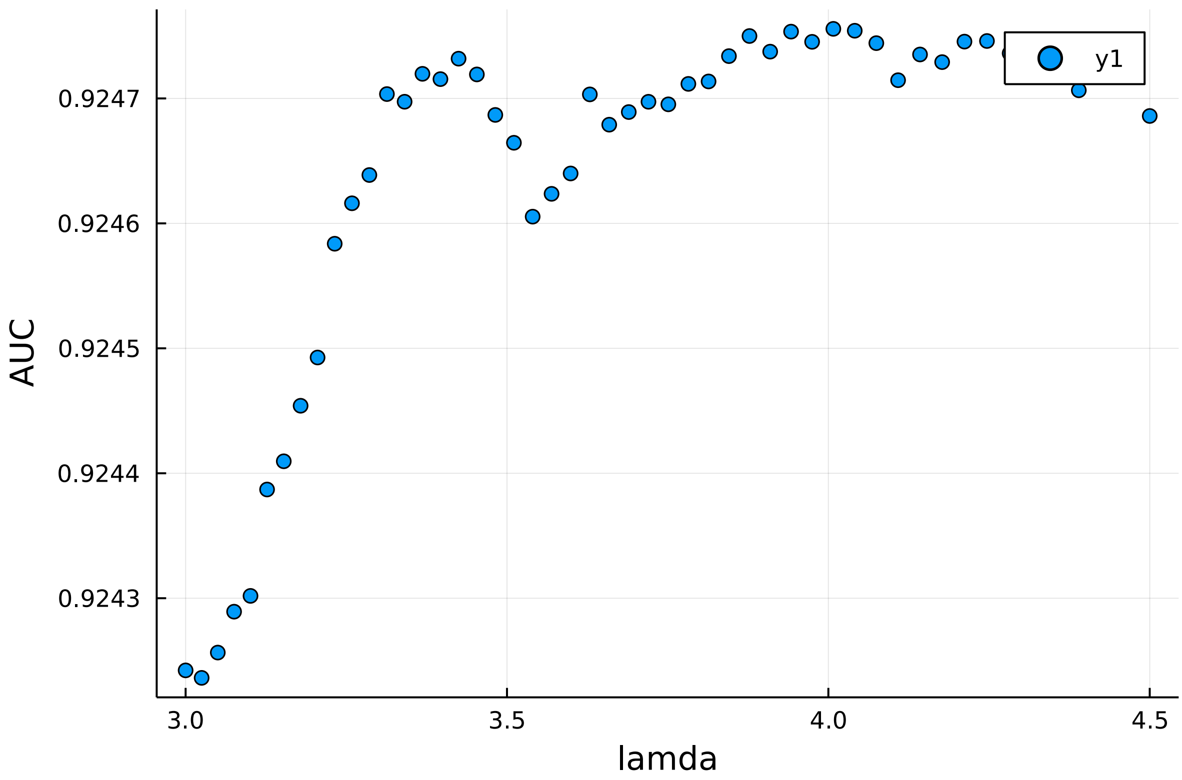
```
Machine{ProbabilisticTunedModel{Grid,…},…} trained 1 time; caches data
  args:
    1:  Source @914 ⏎ `ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.Cont
    2:  Source @230 ⏎ `AbstractVector{ScientificTypesBase.Multiclass{2}}`
```

```
begin
    Random.seed!(10)
    self_tuning_model = TunedModel(model = model,
                                    resampling = CV(nfolds = 5),
                                    tuning = Grid(goal = 50),
                                    range = range(model, :lambda,
                                            scale = :log,
                                            lower = 3, upper = 4.5),
                                            measure = auc)
    self_tuning_mach = machine(self_tuning_model, select(stand_train,
              Not(:precipitation_nextday)),
              stand_train.precipitation_nextday)  |> fit!
end
```

```
, measure = [AreaUnderCurve()], measurement = [0.924756], per_fold = [[⋯ more]]), hi
```

```
false,
```

```
rep = report(self_tuning_mach)
```




```
mach = machine(LogisticClassifier(penalty = :l1, lambda = 4.00775),
               select(stand_train, Not(:precipitation_nextday)),
               stand_train.precipitation_nextday);
```

```
fit!(mach, verbosity = 2);
```

Let's prepare these results for a submission data set. First we have to apply our machine on the test data. Then we construct our submission data and download it.

```
pred = predict(mach, stand_test);
```

true_pred =

[0.802541, 0.840087, 0.937998, 0.102226, 0.977936, 0.125002, 0.0206163, 0.985304, 0.3028

```
true_pred = pdf.(pred, true)
```

```
submission = DataFrame(id = 1:1200, precipitation_nextday = true_pred);
```

```
CSV.write("../data/project/submission_regression.csv", submission);
```