Modal Symbolic Learning: Day 2

NATOPS: Interpretable gesture recognition

In [ ]:
```julia
using Pkg
Pkg.activate(".")
Pkg.instantiate()
Pkg.update()
Pkg.status()
```

```
  Activating project at `~/Desktop/modal-symbolic-learning-course`
    Updating registry at `~/.julia/registries/General`
    Updating git-repo `https://github.com/JuliaRegistries/General.git`
   Installed BenchmarkTools ───── v1.4.0
   Installed LoweredCodeUtils ─── v2.3.2
   Installed ModalDecisionTrees ─ v0.3.3
   Installed SoleLogics ───────── v0.6.11
   Installed SoleModels ───────── v0.5.3
    Updating `~/Desktop/modal-symbolic-learning-course/Project.toml`
  [e54bda2e] ↑ ModalDecisionTrees v0.3.2 ⇒ v0.3.3
  [b002da8f] ↑ SoleLogics v0.6.10 ⇒ v0.6.11
  [4249d9c7] ↑ SoleModels v0.5.2 ⇒ v0.5.3
    Updating `~/Desktop/modal-symbolic-learning-course/Manifest.toml`
  [6e4b80f9] ↑ BenchmarkTools v1.3.2 ⇒ v1.4.0
  [6f1432cf] ↑ LoweredCodeUtils v2.3.1 ⇒ v2.3.2
  [e54bda2e] ↑ ModalDecisionTrees v0.3.2 ⇒ v0.3.3
  [b002da8f] ↑ SoleLogics v0.6.10 ⇒ v0.6.11
  [4249d9c7] ↑ SoleModels v0.5.2 ⇒ v0.5.3
Precompiling project...
  ✓ BenchmarkTools
  ✓ LoweredCodeUtils
  ✓ Revise
  ✓ SoleLogics
  ✓ SoleModels
```

In [8]:
```julia
# Import libraries for statistics & Machine Learning
using Random
using DataFrames
using Plots
using StatsBase

using MLJ
using Sole
```

In [13]:
```julia
using JLD2
using DataFrames # (Maybe need version v1.6.1?)
JLD2.@load "COVID-dataset.jld2"
(X1_df, X2_df, y) = dataset;
```

In [9]:
```julia
countmap(y)
```

Out[9]:
```
Dict{String, Int64} with 2 entries:
  "POSITIVE" => 91
  "NEGATIVE" => 106
```

In [21]:
```julia
using SoleData

X_multimodal = MultiModalDataset([X1_df, X2_df])
```

Out[21]:

```
● MultiModalDataset{DataFrame}
   └─ dimensionalities: (1, 1)
- Modality 1 / 2
   └─ dimensionality: 1
197×30 SubDataFrame
 Row │ cough_F1                           cough_F2
c …
     │ SubArray…                          SubArray…
S …
─────┼──────────────────────────────────────────────────────────────────────────
─────
   1 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0658202, 0.0389503, 0.505408,…
[ …
   2 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [1.62005, 0.648436, 0.248455, 0.…
[
   3 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [2.95991, 1.13398, 0.151934, 0.1…
[
   4 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [3.66094, 0.709991, 0.342895, 0.…
[
   5 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [1.26265, 0.966647, 0.281845, 0.…
[ …
   6 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [4.12442, 2.09424, 0.267123, 0.1…
[
   7 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [3.42468, 1.26102, 0.164075, 0.0…
[
   8 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [10.8754, 0.646552, 0.395368, 0.…
[
   9 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [18.1984, 2.61893, 1.77536, 1.02…
[ …
  10 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [3.37013, 1.19436, 0.97079, 0.60…
[
  11 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [2.34154, 0.925767, 0.378433, 0.…
[
  ⋮  │                 ⋮                                  ⋮
⋱
 188 │ [0.0726378, 0.00766837, 0.016578…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 189 │ [0.35084, 0.093121, 0.0471824, 0…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ …
 190 │ [0.00220116, 0.0199147, 0.166516…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 191 │ [0.0770436, 0.00820453, 0.001445…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 192 │ [0.0453022, 0.00469916, 0.001939…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 193 │ [3.23477e-5, 0.000132317, 0.0001…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ …
 194 │ [0.127782, 0.101379, 0.0369538, …  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 195 │ [0.00149723, 0.000397901, 0.0010…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 196 │ [0.00252365, 0.0540424, 0.065236…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 197 │ [0.115361, 0.0464483, 0.0123202,…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ …
                                                  28 columns and 176 rows om
```

```
itted
- Modality 2 / 2
   └─ dimensionality: 1
197×30 SubDataFrame
 Row │ breath_F1                          breath_F2
b ⋯
     │ Array…                             Array…
A ⋯
─────┼──────────────────────────────────────────────────────────────────────────
─────
   1 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0275974, 0.123487, 0.954918, …
[ ⋯
   2 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.00551542, 0.738461, 1.23345, …
[
   3 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0171132, 0.0128541, 0.175568,…
[
   4 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0713483, 0.191903, 0.0599709,…
[
   5 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0346951, 0.026089, 6.83611, 2…
[ ⋯
   6 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0372815, 1.61098, 7.59596, 1.…
[
   7 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0179035, 0.0158171, 6.72805, …
[
   8 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.038656, 0.202587, 0.0588298, …
[
   9 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.177666, 0.296219, 0.00581333,…
[ ⋯
  10 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.407654, 0.4168, 0.0640127, 0.…
[
  11 │ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…  [0.0442603, 0.0363921, 1.94756, …
[
  ⋮   │                 ⋮                                  ⋮
⋱
 188 │ [0.000519218, 0.000857345, 0.000…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 189 │ [0.0134826, 0.495593, 0.0, 0.0, …  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ ⋯
 190 │ [0.000243355, 0.000393873, 0.000…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 191 │ [0.000504018, 0.00280926, 0.0030…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 192 │ [0.00447856, 0.0114002, 0.004106…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 193 │ [0.00081567, 0.00117782, 0.00106…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ ⋯
 194 │ [0.101095, 0.472197, 0.0360748, …  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 195 │ [0.00159912, 0.00231749, 0.00020…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 196 │ [0.0280692, 0.0461481, 0.218395,…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[
 197 │ [0.251001, 0.0259159, 0.0174663,…  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0…
[ ⋯
                                                      28 columns and 176 rows om
itted
```

In [23]:
```
modality(X_multimodal, 1) # Cough modality
```

Out[23]: 197×30 SubDataFrame *172 rows omitted*

Row	cough_F1	cough_F2	cough_F3	cough_F4	cough_F5	c
	SubArray…	**SubArray…**	**SubArray…**	**SubArray…**	**SubArray…**	**S**
1	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0658202, 0.0389503, 0.505408, 0.659653, 0.479911, 0.198311, 0.038341, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0130705, 0.00773471, 0.100364, 0.130993, 0.0953002, 0.0393805, 0.00761372, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00454093, 0.00181044, 0.00463918, 0.00812784, 0.0114677, 0.00279443, 0.00452539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 1 0 0 0 0 0
2	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1.62005, 0.648436, 0.248455, 0.272204, 0.115829, 0.701655, 0.730142, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.321709, 0.128766, 0.0493379, 0.0540539, 0.0230012, 0.139334, 0.144991, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0714415, 0.0461039, 0.0216893, 0.0326628, 0.019149, 0.0363546, 0.0479683, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[4 2 3 1 3 4 0 0 0
3	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[2.95991, 1.13398, 0.151934, 0.122773, 0.0847807, 0.0358195, 0.023554, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.587777, 0.225185, 0.0301709, 0.0243801, 0.0168357, 0.00711301, 0.00467733, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.287868, 0.100246, 0.00532329, 0.00555218, 0.00194837, 0.00139787, 0.0025744, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[9 0 0 0 0 0 0 0 0
4	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[3.66094, 0.709991, 0.342895, 0.265306, 0.0468018, 0.0975574, 0.0396136, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.726986, 0.140989, 0.0680918, 0.0526841, 0.00929385, 0.0193729, 0.00786644, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.596604, 0.143737, 0.0243013, 0.017446, 0.00756199, 0.00591798, 0.000982618, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1 2 1 0 0 0 0 0 0 0
5	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1.26265, 0.966647, 0.281845, 0.14006, 0.0330181, 0.0189545,	[0.250736, 0.191956, 0.0559685, 0.027813, 0.00655671, 0.00376397,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.128993, 0.0764966, 0.0233883, 0.0117256, 0.00282783, 0.00201767,	[7 2 1 0 0

Row	cough_F1 SubArray…	cough_F2 SubArray…	cough_F3 SubArray…	cough_F4 SubArray…	cough_F5 SubArray…	c S
		0.0202496, 0.00427777, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.00402115, 0.000849476, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]		0.00842082, 0.00644938, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0 0 0 0 0
6	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[4.12442, 2.09424, 0.267123, 0.100803, 0.104586, 0.0637793, 0.0165705, 0.0220245, 0.0118193, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.819024, 0.415873, 0.053045, 0.0200174, 0.0207686, 0.0126652, 0.00329055, 0.0043736, 0.00234707, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.164946, 0.138568, 0.0176139, 0.00586853, 0.00260732, 0.00219175, 0.00100813, 0.00768144, 0.00797681, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1 1 0 0 0 0 0 0 0 0 0
7	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[3.42468, 1.26102, 0.164075, 0.0827096, 0.0727358, 0.0332375, 0.00864315, 0.0294517, 0.0301729, 0.0451799, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.680069, 0.250411, 0.0325819, 0.0164244, 0.0144438, 0.00660026, 0.00171635, 0.00584848, 0.00599171, 0.00897178, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.201791, 0.102281, 0.0150567, 0.00414152, 0.00349754, 0.00190844, 0.000577779, 0.000441292, 0.00100831, 0.00227998, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[9 1 0 0 0 0 0 0 0 0 0
8	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[10.8754, 0.646552, 0.395368, 0.306463, 0.0464313, 0.0182149, 0.0898324, 0.11998, 0.025443, 0.011258, 0.0263648, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[2.15962, 0.128392, 0.0785118, 0.0608572, 0.00922028, 0.00361709, 0.0178388, 0.0238254, 0.00505245, 0.00223559, 0.00523549, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.531624, 0.0291421, 0.0249703, 0.0202988, 0.00218111, 0.00111567, 0.00265106, 0.00404789, 0.00576916, 0.00679558, 0.00494477, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[2 2 1 0 0 0 0 0 0 0 0 0
9	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[18.1984, 2.61893, 1.77536, 1.02094, 0.197965, 0.506792, 0.250556, 0.112402,	[3.61381, 0.520065, 0.35255, 0.202737, 0.0393117, 0.100638, 0.0497552, 0.0223206,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1.0694, 0.227274, 0.472316, 0.224377, 0.0118801, 0.0112027, 0.0299956, 0.0321187,	[2 4 2 1 1 2 3

Row	cough_F1 SubArray…	cough_F2 SubArray…	cough_F3 SubArray…	cough_F4 SubArray…	cough_F5 SubArray…	c S
		0.234197, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.0465066, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]		0.027308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	2 0 0 0
10	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[3.37013, 1.19436, 0.97079, 0.600305, 0.176715, 0.0773258, 0.0182055, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.669236, 0.237175, 0.192778, 0.119208, 0.0350919, 0.0153553, 0.00361524, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.446583, 0.20182, 0.068445, 0.0334676, 0.0137318, 0.00983118, 0.00542909, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1 6 3 1 0 0 0 0 0
11	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[2.34154, 0.925767, 0.378433, 0.123683, 0.0554886, 0.0348641, 0.0349075, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.464981, 0.183838, 0.0751489, 0.0245608, 0.0110189, 0.00692327, 0.0069319, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.388725, 0.139274, 0.0471144, 0.0156152, 0.0121435, 0.0237099, 0.0155309, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1 4 1 1 2 1 0 0 0
12	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[3.07298, 2.57072, 0.687283, 0.322323, 0.214695, 0.0641531, 0.0246654, 0.0356907, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.61023, 0.510491, 0.13648, 0.0640066, 0.0426338, 0.0127395, 0.00489802, 0.00708743, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.204552, 0.231794, 0.051393, 0.0287529, 0.0233669, 0.00844628, 0.00442164, 0.00340762, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[2 5 2 2 0 0 0 0 0 0
13	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1.77748, 0.796787, 0.0738923, 0.0442614, 0.0555183, 0.0385192, 0.0508226, 0.101736, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.352971, 0.158225, 0.0146735, 0.00878938, 0.0110248, 0.0076491, 0.0100923, 0.0202027, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.203237, 0.0634473, 0.00590952, 0.00739434, 0.0543729, 0.0397721, 0.0168973, 0.0131004, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[6 0 0 5 3 1 1 0 0 0

Row	cough_F1 SubArray…	cough_F2 SubArray…	cough_F3 SubArray…	cough_F4 SubArray…	cough_F5 SubArray…	c S
⋮	⋮	⋮	⋮	⋮	⋮	
186	[0.0174438, 0.00310092, 0.00849625, 0.026699, 0.0209502, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0132189, 0.00241784, 0.0035302, 0.00828464, 0.00804756, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00534833, 0.000978251, 0.00142831, 0.00335194, 0.00325602, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0
187	[0.0291811, 0.0169011, 0.0162059, 0.0177519, 0.121101, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0421715, 0.00966519, 0.00904388, 0.0169365, 0.0497429, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0170625, 0.00391051, 0.00365913, 0.00685247, 0.0201259, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0
188	[0.0726378, 0.00766837, 0.016578, 0.0160573, 0.0146287, 0.011821, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0858145, 0.00341952, 0.00950763, 0.0234871, 0.0317684, 0.0292289, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0347203, 0.00138353, 0.00384676, 0.00950284, 0.0128534, 0.011826, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0 0
189	[0.35084, 0.093121, 0.0471824, 0.766623, 6.82305, 1.83248, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.204067, 0.0762902, 0.0502624, 0.666629, 5.60448, 1.80846, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.082565, 0.0308668, 0.020336, 0.269717, 2.26756, 0.731699, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 8 4 0 0 0 0
190	[0.00220116, 0.0199147, 0.166516, 0.0951876, 0.128625, 0.459631, 0.263766, 0.0405767,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00105089, 0.0269954, 0.201371, 0.124756, 0.0703357, 0.288238, 0.229707, 0.0601887,	[0.000425188, 0.0109223, 0.0814741, 0.0504759, 0.0284576, 0.11662, 0.0929391, 0.0243522,	[0 0 0 0 1 1 0

Row	cough_F1 SubArray…	cough_F2 SubArray…	cough_F3 SubArray…	cough_F4 SubArray…	cough_F5 SubArray…	c S
	2.5724, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]			3.26771, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	1.32211, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	5 0 0 0
191	[0.0770436, 0.00820453, 0.00144554, 0.0066907, 0.143037, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.115118, 0.00808703, 0.00118068, 0.0132742, 0.0605214, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0465763, 0.00327199, 0.000477701, 0.00537071, 0.0244868, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0
192	[0.0453022, 0.00469916, 0.00193911, 0.00619609, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0162656, 0.00264962, 0.000846998, 0.00225862, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00658101, 0.00107203, 0.000342693, 0.000913833, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0 0
193	[3.23477e-5, 0.000132317, 0.000143602, 0.000385296, 0.0215862, 0.0367591, 0.00904464, 0.0214649, 0.0223188, 0.0445251, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[1.90018e-5, 7.08086e-5, 6.93151e-5, 0.000174685, 0.00977338, 0.0182157, 0.00556471, 0.0112506, 0.00797403, 0.0179365, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[7.68809e-6, 2.8649e-5, 2.80447e-5, 7.06772e-5, 0.00395429, 0.00737002, 0.00225147, 0.00455199, 0.00322627, 0.00725706, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[5 5 0 0 0 0 0 0 0 0 0 0
194	[0.127782, 0.101379, 0.0369538, 0.019505, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.136504, 0.0900751, 0.0530071, 0.0228197, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.055229, 0.0364442, 0.0214465, 0.00923281, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0 0 0 0 0 0 0 0 0
195	[0.00149723, 0.000397901, 0.0010245,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,	[0.000619492, 0.000181285, 0.000723292,	[0.000250645, 7.33474e-5, 0.000292642,	[0 0

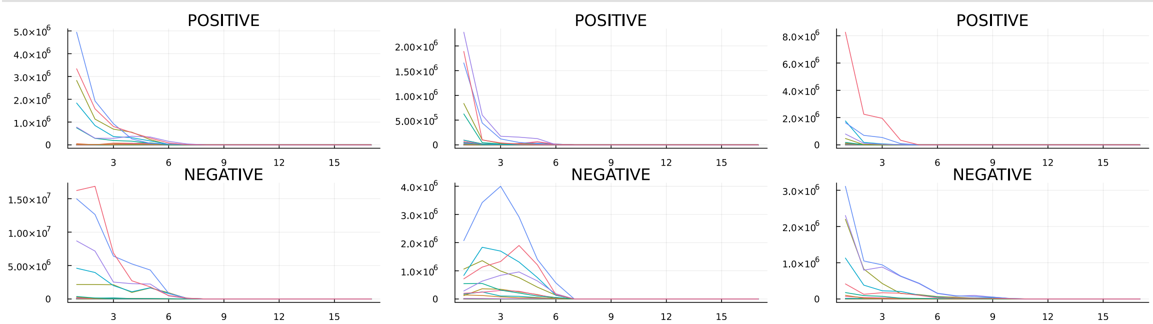
Row	cough_F1 SubArray…	cough_F2 SubArray…	cough_F3 SubArray…	cough_F4 SubArray…	cough_F5 SubArray…	c S
	0.17157, 0.276365, 0.0568549, 0.0536703, 0.439603, 0.599519, 0.126497, 0.00765833, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.162552, 0.357302, 0.139281, 0.0253212, 0.291274, 0.288937, 0.064765, 0.00581977, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	0.0657682, 0.144563, 0.0563528, 0.0102449, 0.117849, 0.116903, 0.0262038, 0.00235467, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	C C C C C C C C C C
196	[0.00252365, 0.0540424, 0.0652368, 0.019338, 0.0124232, 0.00385076, 0.00511448, 0.00346474, 0.00342869, 0.0238765, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00247622, 0.117222, 0.19843, 0.0274702, 0.0143665, 0.00181369, 0.00400859, 0.00402005, 0.00280431, 0.0134033, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.00100187, 0.0474279, 0.0802844, 0.0111144, 0.00581265, 0.000733817, 0.00162187, 0.0016265, 0.00113462, 0.00542295, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[C C C C C C C C C C C C C
197	[0.115361, 0.0464483, 0.0123202, 0.0632743, 0.11675, 0.0424258, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.206398, 0.0766185, 0.0210895, 0.0674126, 0.107959, 0.0308161, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0835083, 0.0309996, 0.00853277, 0.027275, 0.0436798, 0.0124681, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[C C C C C C C C C

In [26]:
```
names(X1_df)
```

Out[26]:
```
30-element Vector{String}:
 "cough_F1"
 "cough_F2"
 "cough_F3"
 "cough_F4"
 "cough_F5"
 "cough_F6"
 "cough_F7"
 "cough_F8"
 "cough_F9"
 "cough_F10"
 "cough_F11"
 "cough_F12"
 "cough_F13"
 ⋮
 "cough_F19"
 "cough_F20"
 "cough_F21"
 "cough_F22"
 "cough_F23"
 "cough_F24"
 "cough_F25"
 "cough_F26"
 "cough_F27"
 "cough_F28"
 "cough_F29"
 "cough_F30"
```

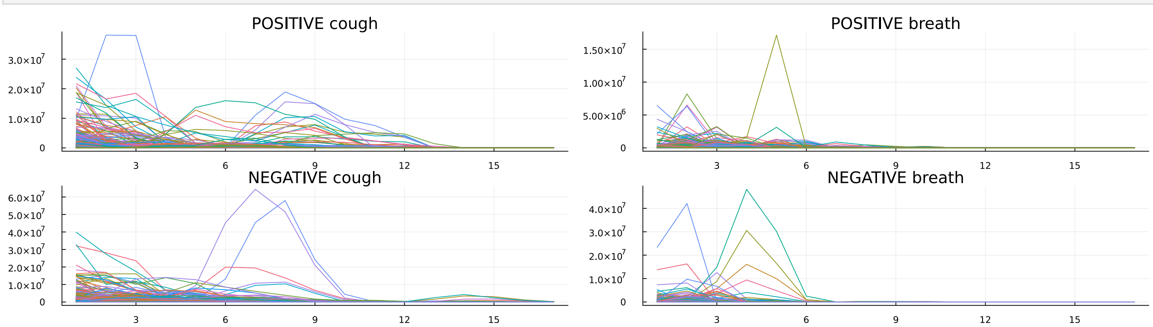
In [39]:
```
# Let's inspect some instances
plot(map(i->plot(collect(X1_df[i,:]), labels=nothing,title=y[i]), 1:35:(35*6
```

Out[39]:



In [41]:
```
# All instances, grouped by class
plot(map((((X_mod, mod_name), _y),)->plot(collect.(eachrow(X_mod[findall(__y
```

Out[41]:



In [72]:
```julia
using ModalDecisionTrees
```

Out[72]:
```
untrained Machine; caches model-specific representations of data
  model: ModalDecisionTree(max_depth = nothing, …)
  args:
    1:  Source @567 ⏎ Table{AbstractVector{AbstractVector{Continuous}}}
    2:  Source @759 ⏎ AbstractVector{Textual}
```

In [ ]:
```julia
# Train in cross-validation!
e = evaluate!(machine(ModalDecisionTree(; relations = :RCC8), X_multimodal,
        resampling=StratifiedCV(shuffle=true, nfolds = 3),
        measures=[accuracy],
        verbosity=0,
        check_measure=false
)
```

```
[ Info: Precomputing logiset...
```

In [ ]:
```julia
# Accuracies per fold
e.per_fold
```

In [71]:
```julia
e.report_per_fold[2].solemodel
```

Out[71]:

```
▣ {1}(⟨G⟩(min[V31] ≥ 0.001853897887096119))
├✔ {1}(⟨G⟩((min[V31] ≥ 0.001853897887096119) ∧ ⟨A̅O̅⟩(min[V27] < 76685.17720224746)))
│├✔ {1}(⟨G⟩((min[V31] ≥ 0.001853897887096119) ∧ ⟨A̅O̅⟩((min[V27] < 76685.17720224746) ∧ (min[V28] < 44762.952262639636))))
││├✔ {1}(⟨G⟩((min[V31] ≥ 0.001853897887096119) ∧ ⟨A̅O̅⟩((min[V27] < 76685.17720224746) ∧ (min[V28] < 44762.952262639636) ∧ ⟨D̅B̅E̅⟩(min[V50] ≥ 1885.5712367271199))))
│││├✔ {1}(⟨G⟩((min[V31] ≥ 0.001853897887096119) ∧ ⟨A̅O̅⟩((min[V27] < 76685.17720224746) ∧ (min[V28] < 44762.952262639636) ∧ ⟨D̅B̅E̅⟩((min[V50] ≥ 1885.5712367271199) ∧ (min[V13] ≥ 1.400957923736261)))))
││││├✔ NEGATIVE
││││└✘ POSITIVE
│││└✘ NEGATIVE
││└✘ POSITIVE
│└✘ NEGATIVE
└✘ {1}(⟨G⟩(min[V27] ≥ 2.170219509362793e6))
 ├✔ {1}(⟨G⟩((min[V27] ≥ 2.170219509362793e6) ∧ ⟨G⟩(min[V46] ≥ 5838.390775457752)))
 │├✔ POSITIVE
 │└✘ {1}(⟨G⟩((min[V27] ≥ 2.170219509362793e6) ∧ (min[V19] ≥ 30000.48907253628)))
 │ ├✔ NEGATIVE
 │ └✘ {1}(⟨G⟩((min[V27] ≥ 2.170219509362793e6) ∧ (min[V21] < 20414.774427686534)))
 │  ├✔ NEGATIVE
 │  └✘ POSITIVE
 └✘ {1}(⟨G⟩(min[V28] ≥ 553537.8712413169))
  ├✔ POSITIVE
  └✘ {1}(⟨G⟩(min[V2] ≥ 0.17083754207872534))
   ├✔ {1}(⟨G⟩((min[V2] ≥ 0.17083754207872534) ∧ (min[V11] ≥ 1108.1113127342035)))
   │├✔ NEGATIVE
   │└✘ POSITIVE
   └✘ NEGATIVE
```

In [46]:

```
report(mach).solemodel
```

Out[46]:
```
▣ {1}(⟨G⟩(min[V37] ≥ 0.0013205573259328035))
├✔ {1}(⟨G⟩((min[V37] ≥ 0.0013205573259328035) ∧ ⟨L̅⟩(min[V27] ≥ 226321.99121
13276)))
│├✔ NEGATIVE
│└✘ {1}(⟨G⟩((min[V37] ≥ 0.0013205573259328035) ∧ (min[V48] ≥ 1242.190476996
1468)))
│ ├✔ {1}(⟨G⟩((min[V37] ≥ 0.0013205573259328035) ∧ (min[V48] ≥ 1242.19047699
61468) ∧ ⟨AO⟩(min[V40] ≥ 3.51978641624239)))
│ │├✔ {1}(⟨G⟩((min[V37] ≥ 0.0013205573259328035) ∧ (min[V48] ≥ 1242.1904769
961468) ∧ ⟨AO⟩((min[V40] ≥ 3.51978641624239) ∧ ⟨AO⟩(min[V31] ≥ 1.6207745800
042226e-5))))
│ ││├✔ POSITIVE
│ ││└✘ NEGATIVE
│ │└✘ POSITIVE
│ └✘ NEGATIVE
└✘ {1}(⟨G⟩(min[V27] ≥ 2.170219509362793e6))
 ├✔ {1}(⟨G⟩((min[V27] ≥ 2.170219509362793e6) ∧ ⟨G⟩(min[V46] ≥ 5838.39077545
7752)))
 │├✔ POSITIVE
 │└✘ {1}(⟨G⟩((min[V27] ≥ 2.170219509362793e6) ∧ (min[V30] < 1.6492884918659
376e6)))
 │ ├✔ POSITIVE
 │ └✘ NEGATIVE
 └✘ {1}(⟨G⟩(min[V28] ≥ 672777.1743629333))
  ├✔ POSITIVE
  └✘ {1}(⟨G⟩(min[V30] ≥ 1.0114283593133552e6))
   ├✔ NEGATIVE
   └✘ POSITIVE
```

In [ ]:
```
X = scalarlogiset(X_df)
println(X)
```

In [ ]:
```
```

In [ ]:
```
# Accessibles on multirelation frames
# Get the structure ("frame") of the first instance. It is a "dimensional" f
fr = SoleLogics.frame(X, 1)
```

In [ ]:
```
# Enumerate all worlds
collect(allworlds(fr))
```

In [ ]:
```
using SoleLogics: Interval
# Enumerate the intervals that are "Later" than [1,10]
accessibles(fr, Interval(1,10), IA_L) |> collect
```

In [ ]:
```
# Remember that features are computed on each world
# Let's compute the minimum of the first variable on an arbitrary interval,
feature = UnivariateMin(1)
Sole.featvalue(feature, X, 1, Interval(10,30))
```

In [ ]:
```
# Remember that atoms are *scalar conditions on features*
# Let's check one on an interval of the first instance
```

```
p = Atom(ScalarCondition(feature, >, -0.5))
check(p, X, 1, Interval(10,30))
```

In [ ]:
```
# I can check any formula
p = Atom(ScalarCondition(UnivariateMin(1), >, -0.5))
q = Atom(ScalarCondition(UnivariateMin(2), <=, 10))
φ = p ∨ q
check(φ, X, 1, Interval(10,30))
```

In [ ]:
```
# Generate a random HS formula with scalar conditions on features, and check
features = [UnivariateMin(i_variable) for i_variable in 1:ncol(X_df)]
alpha = [Atom(ScalarCondition(feat, >, thresh)) for feat in features for thr

HS_connectives = SoleLogics.diamondsandboxes(SoleLogics.IARelations)
propo_connectives = SoleLogics.BASE_PROPOSITIONAL_CONNECTIVES

println("Propositional connectives: $(join(syntaxstring.(propo_connectives),
println("HS connectives: $(join(syntaxstring.(HS_connectives), ", "))")

propo_weights = fill(1/length(propo_connectives), length(propo_connectives))
HS_weights = fill(1/length(HS_connectives), length(HS_connectives))

connectives = vcat(propo_connectives, IA_connectives)

opweights = vcat(propo_weights, HS_weights)

treeheight = 3
φ2 = randformula(Random.MersenneTwister(30), treeheight, alpha, connectives;
println()
println("Random formula:")
println(syntaxstring(φ2))

check(φ2, X, 1, Interval(10,30))
```

In [ ]:
```
# Let's check a formula on all the instances
check_mask = check(φ2, X, Interval(10,30))
```

In [ ]:
```
# It holds on part of the instances
sum(check_mask)
```

In [ ]:
```
# Let's ask whether the formula holds *all* intervals, instead of checking i
println("Applying the universal global operator: ", SoleLogics.globalbox)
println()

universal_φ = globalbox(φ2)
println("Formula: ", syntaxstring(universal_φ))
check_mask = check(universal_φ, X)

# It holds on no instance... Too restrictive!
sum(check_mask)
```

In [ ]:
```
# Let's ask whether there exists any interval where the formula holds
println("Applying the existential global operator: ", SoleLogics.globaldiamc
println()
```

```
existential_φ = globaldiamond(φ2)
println("Formula: ", syntaxstring(existential_φ))
check_mask = check(existential_φ, X)

# It holds on more instances
sum(check_mask)
```

In [ ]:
```
# Question: does it lead to a good rule?

println(syntaxstring(existential_φ))

println()
println(SoleLogics.experimentals.formula2natlang(existential_φ))
```

In [ ]:
```
neg_existential_φ = normalize(¬ existential_φ;
    profile = :readability,
    remove_implications = true,
    allow_atom_flipping = true
)

syntaxstring(neg_existential_φ; remove_redundant_parentheses = false)
```

In [ ]:
```
(SoleLogics.precedence(∧), SoleLogics.precedence(∨))
```

In [ ]:
```
countmap(y)
```

In [ ]:
```
println(existential_φ)
countmap(y[check_mask])
```

In [ ]:
```
println(neg_existential_φ)
countmap(y[(!).(check_mask)])
```

In [ ]:
```
branch = Branch(neg_existential_φ, "I have command", "Spread wings")
```

In [ ]:
```
y_preds = SoleModels.apply(branch, X)
println("Accuracy of this branch: $(sum(y .== y_preds)/length(y))")
println("Random chanc e: $(60/length(y))")
```

In [ ]:
```

```

In [ ]:
```
println(X)
```

In [ ]:
```
# Randomly split the data: 20% training, 80% testing
N = nrow(X_df)
perm = randperm(Random.MersenneTwister(1), N)
train_idxs, test_idxs = perm[1:round(Int, N*.2)], perm[round(Int, N*.2)+1:en
println("Using $(length(train_idxs)) instances for training")
println("Using $(length(test_idxs)) instances for testing")
```

In [ ]:
```
using ModalDecisionTrees
```

```
# Bind a machine learning algorithm to logiset & labels
mach = machine(ModalDecisionTree(; relations = :IA7, features = [minimum]),

# Train!
@time fit!(mach; rows=train_idxs);

# Compute accuracy
yhat = predict_mode(mach; rows=test_idxs)
MLJ.accuracy(yhat, y[test_idxs])
```

In [ ]:
```
# Show the restricted MDT learnt
printmodel(report(mach).rawmodel_full; hidemodality = true)
```

In [ ]:
```
# Show its *pure* version
printmodel(report(mach).solemodel_full; show_metrics = true, hidemodality =
```

In [ ]:
```
simplified_restricted_tree = ModalDecisionTrees.prune(report(mach).rawmodel_
printmodel(simplified_restricted_tree)

println()
println("# Leaves: ", nleaves(simplified_restricted_tree))
```

In [ ]:
```
solemodel = ModalDecisionTrees.translate(simplified_restricted_tree)
```

In [ ]:
```
# Print leaf rules + their training performances
ruleset = listrules(solemodel)
printmodel.(ruleset; show_metrics = true, threshold_digits = 2, variable_nam
```

In [ ]:
```
last_rule = ruleset[end]
last_antd = antecedent(last_rule)

println("First formula, translated:")
println(SoleLogics.experimentals.formula2natlang(last_antd; threshold_digits


for (i_rule, rule) in enumerate(ruleset)
    println()
    println("[$i_rule]")
    antd = antecedent(rule)
    println(SoleLogics.experimentals.formula2natlang(antd; threshold_digits
end
```

In [ ]:
```
# Print rules + their *test* performances

# Sprinkle the model with the test instances!
predictions, tree_test = report(mach).sprinkle(X_df[test_idxs,:], y[test_idx

# Extract ruleset and print its metrics
ruleset_test = listrules(tree_test);

# printmodel.(ruleset_test; show_metrics = true, threshold_digits = 2, varia
printmodel.(ruleset_test; show_metrics = true, threshold_digits = 2, parenth
```

In [ ]:
```
println("IF\n\t", SoleLogics.experimentals.formula2natlang(antecedent(rulese
println("THEN\n\t", consequent(ruleset_test[4]))
```

In [ ]:
```
# Obtain class rules & show their *test* metrics
condensed_ruleset_test = joinrules(ruleset_test)
printmodel.(condensed_ruleset_test; show_metrics = true, threshold_digits =
```

Exercise

`ModalDecisionTrees.jl` can also handle images! In which case, they use a 2D logic of rectangles instead of a 1D logic of intervals.

Apply ModalDecisionTrees to Indian Pines, a benchmark dataset for Land Cover Classification with 16 classes. The dataset consists of a hyperspectral image (i.e., 200 color channels instead of the typical 3 RGB channels) where many pixels have been labelled as belonging to one of the classes.

Sketch of the idea:

- Load the image and the ground truths. The package MAT.jl can be helpful;
- From the 145×145 image provided, sample a (small) number m of 3×3 patches for each class, and label each patch with the class label for the central pixel;
- Gather the ground truths into a vector of strings `y`, and the samples into a Julia `DataFrame` `X` with 200 columns, $16m$ rows and 3×3 *matrices* in the cells;
- Use MLJ to train a ModalDecisionTree on `X` and `y`, similarly to the above case.

Suggestion: since the formulas are desirably rotation-invariant, ask the algorithm to use *topological* relations instead of *directional* relations. Rrefer to the doc to know more.