JsonGrinder.jl: hierarchical multi-instance data

Explaining HMIL

Šimon Mandlík, Matěj Račinský, Viliam Lisý, *Tomáš Pevný*

Gen digital,
AIC, FEE CTU in Prague

August 21, 2023

Motivation


```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 4070,
    protocol: tcp
  },
  {
    port: 4071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  device_id: 8bb8971c-5983-4baa-9753-f0ac21faf162,
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [_workstation._tcp.local., _ssh._tcp.local., _sftp-ssh._tcp.local.]
}
```

Motivation

1


```
{
  device_id: 1fe43d89-329d-40fe-b948-d1cbe0fe6c96,
  ip: 192.168.15.26,
  mac: 00:07:25:15:3a:fb,
  dhcp: [
  {
    classid: unknown,
    paramlist: 1,28,2,3,15,6,119,12,44,47,26,121,42
  }]
}
```

Motivation

1

```
{
  device_id: 1fe43d89-329d-40fe-b948-d1cbe0fe6c96,
  ip: 192.168.15.26,
  mac: 00:07:25:15:3a:fb,
  dhcp: [
  {
    classid: unknown,
    paramlist: 1,28,2,3,15,6,119,12,44,47,26,121,42
  }]
}
```

2

```
{
  services: [
  {
    port: 5353,
    protocol: udp
  },
  {
    port: 41800,
    protocol: tcp
  }],
  device_id: 74ab1b5b-3cb6-4aee-8362-f3b2f016574c,
  ip: 192.168.0.7,
  mac: 40:99:22:41:ac:a6,
  mdns_services: [_spotify-connect._tcp.local.]
}
```

Motivation

1

```
{
  device_id: 1fe43d89-329d-40fe-b948-d1cbe0fe6c96,
  ip: 192.168.15.26,
  mac: 00:07:25:15:3a:fb,
  dhcp: [
  {
    classid: unknown,
    paramlist: 1,28,2,3,15,6,119,12,44,47,26,121,42
  }]
}
```

2

```
{
  services: [
  {
    port: 5353,
    protocol: udp
  },
  {
    port: 41800,
    protocol: tcp
  }],
  device_id: 74ab1b5b-3cb6-4aee-8362-f3b2f016574c,
  ip: 192.168.0.7,
  mac: 40:99:22:41:ac:a6,
  mdns_services: [_spotify-connect._tcp.local.]
}
```

3

```
{
  services: [
  {
    port: 80,
    protocol: tcp
  },
  {
    port: 443,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  },
  {
    port: 5683,
    protocol: udp
  }],
  device_id: 5347ada9-925c-400e-8a7c-9aedd3c142f6,
  ip: 192.168.0.147,
  mac: 80:5e:c0:41:ad:39
}
```

Motivation

1


```
{
  device_id: 1fe43d89-329d-40fe-b948-d1cbe0fe6c96,
  ip: 192.168.15.26,
  mac: 00:07:25:15:3a:fb,
  dhcp: [
  {
    classid: unknown,
    paramlist: 1,28,2,3,15,6,119,12,44,47,26,121,42
  }]
}
```


2


```
{
  services: [
  {
    port: 5353,
    protocol: udp
  },
  {
    port: 41800,
    protocol: tcp
  }],
  device_id: 74ab1b5b-3cb6-4aee-8362-f3b2f016574c,
  ip: 192.168.0.7,
  mac: 40:99:22:41:ac:a6,
  mdns_services: [_spotify-connect._tcp.local.]
}
```


3


```
{
  services: [
  {
    port: 80,
    protocol: tcp
  },
  {
    port: 443,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  },
  {
    port: 5683,
    protocol: udp
  }],
  device_id: 5347ada9-925c-400e-8a7c-9aedd3c142f6,
  ip: 192.168.0.147,
  mac: 80:5e:c0:41:ad:39
}
```


4


```
{
  services: [
  {
    port: 80,
    protocol: tcp
  },
  {
    port: 80,
    protocol: tcp
  },
  {
    port: 111,
    protocol: tcp
  },
  {
                    :
    user_agent: unknown
  },
  {
    location: http://192.168.1.121:60006/upnp/desc/aios_device/aios_device.xml,
    st: ,
    nt: urn:schemas-denon-com:device:AiosServices:1,
    server: LINUX UPnP/1.0 Denon-Heos/139046,
    user_agent: unknown
  },
  {
    location: http://192.168.1.121:60006/upnp/desc/aios_device/aios_device.xml,
    st: ,
    nt: uuid:bf87348f-27aa-1691-0080-0005cde2625e,
    server: LINUX UPnP/1.0 Denon-Heos/139046,
    user_agent: unknown
  },
  {
    location: http://192.168.1.121:60006/upnp/desc/aios_device/aios_device.xml,
    st: ,
    nt: uuid:5cdaafbc-2db2-1022-0080-0005cde2625e,
    server: LINUX UPnP/1.0 Denon-Heos/139046,
    user_agent: unknown
  },
  {
    location: http://192.168.1.121:60006/upnp/desc/aios_device/aios_device.xml,
    st: ,
    nt: urn:schemas-upnp-org:service:AVTransport:1,
    server: LINUX UPnP/1.0 Denon-Heos/139046,
    user_agent: unknown
  },
  {
    location: http://192.168.1.121:60006/upnp/desc/aios_device/aios_device.xml,
    st: ,
    nt: urn:schemas-upnp-org:device:MediaRenderer:1,
    server: LINUX UPnP/1.0 Denon-Heos/139046,
    user_agent: unknown
  },
                    :
    description: unknown,
  services: [urn:denon-com:serviceId:ACT]
  },
  {
    model_name: Denon AVR-X2400H,
    manufacturer: Denon,
    device_type: urn:schemas-upnp-org:device:MediaServer:1,
    model_description: Shares User defined folders and files to other Universal Plug and Play media devices.,
    services: [urn:upnp-org:serviceId:ContentDirectory, urn:upnp-org:serviceId:ConnectionManager]
  }],
  mdns_services: [_airplay._tcp.local., _spotify-connect._tcp.local., _raop._tcp.local., _http._tcp
}
```

Our options

- Feature engineering
- JSON as an image
- JSON as text
- Heterogenous graph
- HMIL

HMIL's design

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```

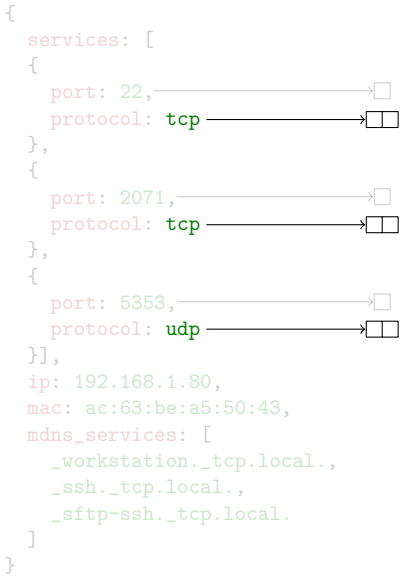
Project each node to vector
assuming all his childs are vectors.

Leafs — numbers

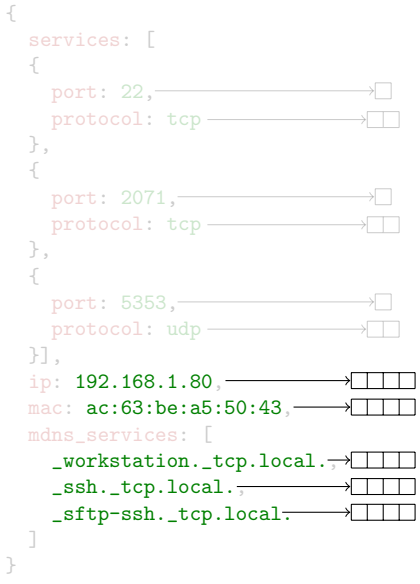
```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



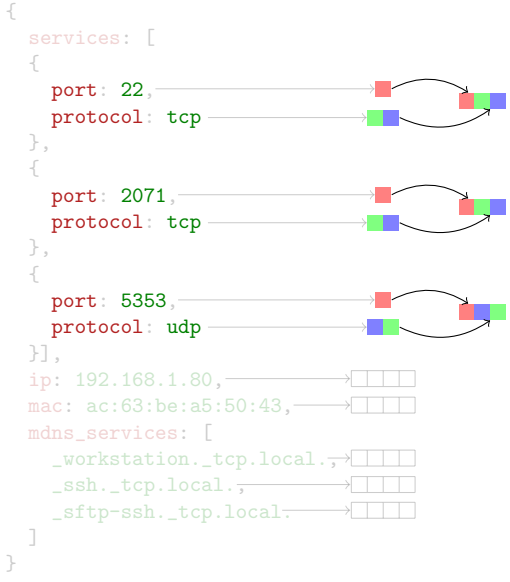
Leafs — categorical features



Leafs — string features

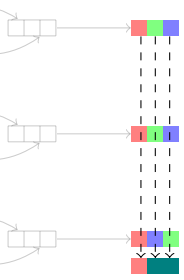


Dictionaries of ports and protocols



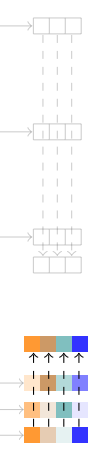
Array of services

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



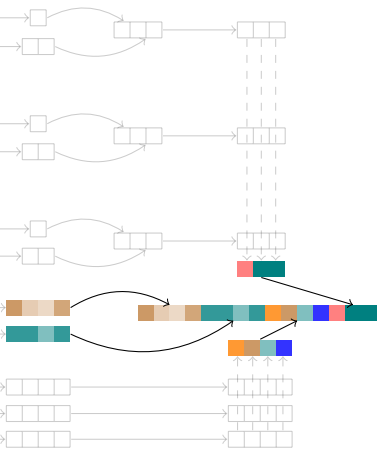
Array of mdns services

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



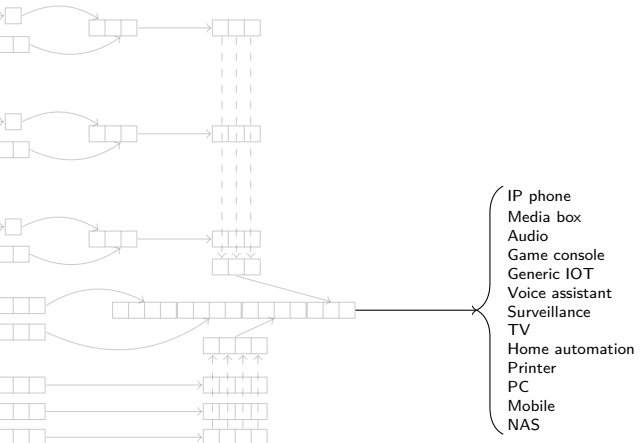
Topmost dictionary

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



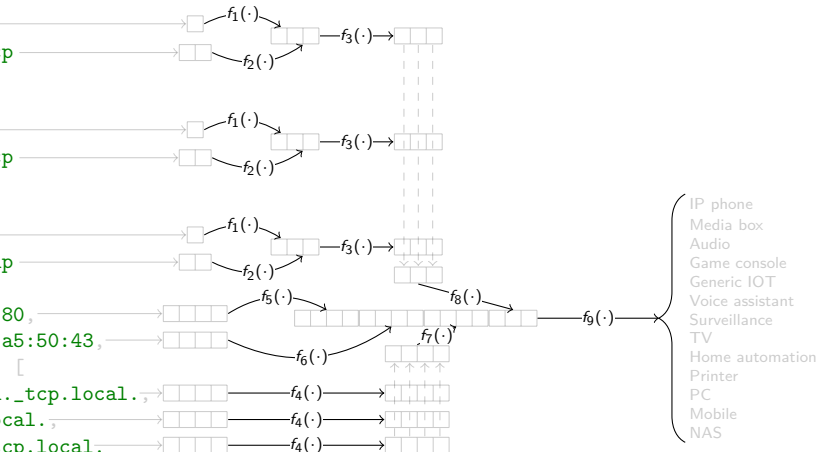
Predicting classes

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



Complete model

```
{
  services: [
  {
    port: 22,
    protocol: tcp
  },
  {
    port: 2071,
    protocol: tcp
  },
  {
    port: 5353,
    protocol: udp
  }],
  ip: 192.168.1.80,
  mac: ac:63:be:a5:50:43,
  mdns_services: [
    _workstation._tcp.local.,
    _ssh._tcp.local.,
    _sftp-ssh._tcp.local.
  ]
}
```



Example of use

Code | Blame | 29 lines (22 loc) · 960 Bytes | Raw

```julia
using JsonGrinder, Mill, Flux, MLDatasets, JSON, CSV, DataFrames, Statistics, Random

Random.seed!(42)

include("prepare_data.jl")

BATCH_SIZE = 50

x_train, y_train = load_train_data()
schema = JsonGrinder.schema(x_train)
extractor = suggestextractor(schema)
ds_train = map(extractor, x_train)

encoder = reflectinmodel(schema, extractor, d -> Chain(Dense(d, 50, relu), Dropout(0.25)))
model = Dense(50, length(LABELS)) ∘ encoder
loss = (ds, y) -> Flux.Losses.logitcrossentropy(model(ds), Flux.onehotbatch(y, eachindex(LABELS)))
accuracy = (ds, y) -> mean(Flux.onecold(model(ds), eachindex(LABELS)) .== y)

opt = AdaBelief()
ps = Flux.params(model)
data_loader = Flux.Data.DataLoader((ds_train, y_train), batchsize=BATCH_SIZE, shuffle=true)
Flux.@epochs 3 begin
    Flux.Optimise.train!(loss, ps, data_loader, opt)
    @show accuracy(ds_train, y_train)
end

x_test, y_test = load_test_data()
ds_test = map(extractor, x_test)
@show accuracy(ds_test, y_test)
```

Comparison to SOTA

Dataset	Sample Size	Accuracy Default	Tuned	Competing
`Mutagenesis`	4k-8k	0.886	0.932	0.912
`Device ID`	0.1k-0.3M	0.932	0.971	0.967
`EMBER`	3k-6M	0.948	0.996	0.974

Try it

https://github.com/CTUAvastLab/JsonGrinder.jl