# Statistical Learning by Logical Abduction

BY DAIWZ

LAMDA Group
November End, 2015

## 1 Problem Setting

**Input:**

- Training data: $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $x$ is feature, $\boldsymbol{y} = (y_1, ..., y_l)$ is a vector of multiple labels.
- Background knowledge:
  - Label relations: e.g. label hierarchy (`father(label_i,label_j)`,...).
  - Other knowledge: template (meta-rules) for rule abduction; knowledge of constructing meta-rules. For example, `father(P,Q)`$\rightarrow$`metarule(Q(X):-P(X),R(X))`, meaning if label `P` is father of label `Q`, then every rule about `Q` must include `P` in its body.

**Output:**

- A set of rules as hypothesis of each label.
- Predicates in rules are statsitical classifiers, i.e., `P(X):-(`$h_p$`(X)>0).`

**Main Idea:**

Background knowledge $\Longrightarrow^{\spadesuit}$ Meta-rules (template rules) $\Longrightarrow$ Statistical Models learning.

$\spadesuit\Longrightarrow$ means "control"

- Incorporating first-order background knowledge by letting them to control the learning of statistical classifiers. The meta-rules act as templates and enables predicate invention, and the invented "predicates" are statistical classifiers.
- Statistical learning is directly controled by its training data, which is actually controlled by rule induction by variable bindings. For example, when learning a classifier `R(X)` inside of "`tree(X):-plant(X),R(X)`", the training data passed to `R` is filtered by `plant(X)`, which indicates `R(X)` should only learn the difference between `tree` and other `plants`.
- Reuse the learned classifiers as logical predicates. The invented predicate (for exapmle the `R` in last paragraph) might be an useful mid-level concept that can help the learning of other labels.

## 2 Motivation

Statistical learning solves a zero-order problem – learn a first-order representation of target concept (predicate/symbol) from low-level feature space. By assuming i.i.d. among the domain and even conditional independence between features (e.g. $P(x_i, x_j|y) = P(x_i|y)P(x_j|y)$), the problem has been simplified. Moreover, discriminative statistical learning only tries to learn the difference between different concepts rather than the constructive structure (relations) of concepts, which makes current mainstream statistics based machine learning has a huge necessity of examples. In short, both of the modeling process and the learning target ignores lots of relations in domain and tries to solve an over-simplified problem.

In fact, plenty of first-order relations between sub-concepts play important roles in our real cognition and learning problems. The more important thing is, **learning is a constructive process, human always keep trying to interprete new concepts (symbols) with their old knowledge (background knowledge).** We can look at an interesting example: when a child is trying to understand what is "tree", should he/she understand what is "leaf" or what is "branch" in former? In fact, people can recognise trees far before understanding the sub-concpets of tree like branches, trunks, leaves, roots and so on. They only know colors, lights, edges between objects or other relatively lower-level features. After knowing what is a tree, they will continuously learn the concept of branch, and be able to recognise a tree even when it losts all of its leaves, or been cut off and layed down on the ground as a "dead tree". This example gives us a hint that the existence of background knowledge of sub-concepts in learning may be not so important, they could be **abduced** by higher-level concepts.

The next problem is how can we abduce the sub-concepts? Where do they come from? Here is an other observation: Human can learn multiple concepts at one time, clearly, by telling differences and similarities between concepts and objects. We believe that the **frequently appeared similar/different parts** are the sub-concepts. (From this angle of view, discriminative learning does make a certain degree of sense, however it ignores the first-order relations between the differences.) However, in the learning process, it is hard to mining the sub-parts as frequent itemsets (it requires exponentially enumerations). Thus, we try to use label hierarchy to guide the abduction of sub-concept symbols, the difference/similarity between hierarchical labels are relatively more easier to capture.

# 3 Proposed Approach

## 3.1 Meta-rules

Given background knowledge of label hierarchy, we can generate meta-rules to perform the abduction.

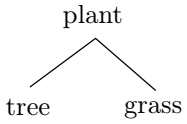


Fig. 1 Label hierarchy

For example in Fig. 1, the label hierarchy is `father(plant,tree)` and `father(plant,grass)`. Since both `tree` and `grass` are children of `tree`, they must share some information, the meta-rule of these two concept should be:

**Rule A**: `tree(X):-plant(X),tree_1(X).`

**Rule B**: `grass(X):-plant(X),grass_1(X).`

The `plant(X)` in their bodies shows the similarity between these two concepts, the `***_1(X)` predicates are the difference between them.

The **meta-rules** are templates of first-order logic rules expressed by predicates in *list forms*. In *list form*, each predicate is expressed by a list, the first item is the name of the predicate (first-order), the rests are arguments (zero-order). For example, the **rule A** can be written in *list form* as "`[tree,X]:-[plant,X],[tree_1,X]`". By substituting all the constants by variables (here we use `P,Q,R,...` to represent predicate variables, `X,Y,Z,...` to represent object variables), the **meta-rule** of these two rule are:

**Meta Rule A**: `[P,X]:-[Q,X],[R,X].`

If we want to enforce the learned rule to contain information about `plant/2`, we just modify the meta-rule as:

**Meta Rule B**: `[P,X]:-[plant,X],[R,X].`

There could be numbers of meta-rules, according to [Muggleton et.al., 20xx], few of them is already enough for learning a universal Turing machine.

## 3.2 Abduction

In this section, we explain the implementation of the abduction process.

Suppose we have already obtained label hierarchy like Fig. 1, we have following instructions for meta-rule construction:

i. Body of meta-rule of child concepts must contain its parent concept. i.e., `child(X):-parent(X),...` This rule indicates the `is-a` relation between concepts.

ii. [More details to be added]...

### 3.2.1 Searching meta-rules

Given a label to be learned, generate candidate programs. For example, if we want to learn `tree(X)`, and given meta-rules: `[P,X]:-[Q,X],[R,X]` and `[P,X]:-[Q,X],[R,X],[S,X]`. The candidate programs are:

```
ps(metarule([P,X]:-[Q,X],[R,X]),metasub([P,tree],[Q,plant])).
ps(metarule([P,X]:-[Q,X],[R,X],[S,X]),metasub([P,tree],[Q,plant])).
```

After generating candidate programs, the learning procedure will start substuting `Q,S` and assessing the quality of different substitution. For example, if there already exists rules of `grass/1`, then there are multiple choices to continue this search:

1. a possible `metasub` will be `[P,tree],[Q,plant],[R,grass]`;
2. another choice is to generate a new predicate, which results in `metasub` as `[P,tree],[Q,plant],[R,tree_1]`,the new predicate can be invented by:
   a) learning a new rule: (e.g. `[tree_1,X]:-[Q,X],[R,X]`);
   b) or simply learn a statistical classifier (e.g. $\text{tree}(x) = \text{sgn}(h(x))$).

### 3.2.2 Training statistical models

The meta-rules in abduction also act as training data filters that selects training data for statistical model learning.

Moreover, when learning statistical models, the biases can be flexibly declared in first-order logical background knowledge. For example, if the models to be learned are decision trees, and the background knowledge holds the belief that the invented mid-level concepts shouldn't be very complex, then the parameter of shallow depth can be passed to the "statistical classifier invention" process. If the models are SVMs, we can even declare sparsity or other regularization in logical form and pass them to statistical learning process.

# 4 Implementation

## 4.1 Out line of the system

The *Symblearn* system consist of two major parts: `c++ data handler` and `prolog abductor`. The `c++` part takes charge of data processing and memory handling; the `prolog` is used for rule abduction.

→ **core**/: [implementing] directory of main functionalities

`symblearn.cpp`: [implementing] The main program, initialize Prolog engine, read input data, organise data into prolog-readable formats.

`dataset.hpp`: [implementing] Dataset interface, controls the data accessor and in charges of partitioning data and passing them to Prolog abductor.

`classifier.cpp`: [done] C++-implemented logical predicates, reads data and classifier and output TRUE or FALSE etc. The predicates it contains are listed as follow:

- `classify_rv|ui_set|inst(&model,&data,Class)`: input `&model` and `&data`, output `class` of data. Depend on different classifier, the output could be `unsinged int` or `RealVector`; If `&data` is an adress of dataset then `class` is a list of label (e.g. `[1,0,1,1,...]`) , if `&data` is a instance (`RealVector`) then `class` is a real value (-1 or 1).
- `filter_data(&dataset,Selector,&outputData)`: select a subset of data according to a binary selector `[1,0,0,1,...]`.
- `stat_classifier(&data,&model,Eval)`: train/evaluate a statistical classifier `&model` with `&data`; `Eval` is the evaluation of `&model` on `&data`, it could be accuracy, F1 score, etc. (For simplicity, now we are just using fixed depth CART Tree implemented by Shark toolbox. [more to come]).
- `free_ptr(ptr)`: free the memory address of `ptr` in c-stack (not Prolog stack)
- ...

→ **arff**/: [done] directory of reading benchmark HMC arff data(https://dtai.cs.kuleuven.be/clus/hmc-ens/)

`arff_data.h`: interface of ARFF data (handles hierarchical labels).

`arff_parser.h`: parses ARFF files.

...

→ [More to add]

### 4.2 Algorithm

**Step 1**: *Reading data*. This job is done by `c++` part. The label hierarchy or other background knowledge are also inputed in this stage. For example, label hierarchy are parsed into `father(L1, L2)`, and asserted in the prolog engine.

[More to add]

## 5 Experiments

### 5.1 Datasets

Hierarchical Multi-label datasets from Clus-HMC (https://dtai.cs.kuleuven.be/clus/hmc-ens/).

### 5.2 Compared Methods

[More to add]