Application of Optimal Search Theory to ILP

Srihari Kalgi[1], Chirag Gosar[1], Ganesh Ramakrishnan[1], and Ashwin Srinivasan[2]

[1] Indian Institute of Technology, Bombay
srihari_kalgi@it.iitb.ac.in
cgosar@cse.iitb.ac.in
ganesh@cse.iitb.ac.in
[2] IBM India Research Lab, Delhi
ashwin.srinivasan@in.ibm.com

1 Introduction

For most ILP systems that attempt to find a hypothesis by searching through a space of possible alternatives, the principal problem is that the size of the search space makes it impossible to perform even a complete (but non-exhaustive) search. Hence, greedy approaches such as FOIL [1] and restricted variants of branch and bound techniques (Aleph [2]), *etc.* are used to restrict the search space. In this paper, we will concern ourselves principally with systems like Progol [3], in which search proceeds by examining subspaces constrained by a set of most-specific clause $\perp_1, \perp_2, \ldots, \perp_n$ (where n here will be the number of positive examples) using a greedy covering procedure.

In each iteration of the greedy procedure a Progol-like system selects a positive example e_i, construct a bottom clause $\perp_i$ using the example and the background knowledge B and then searches for the best clause h_i in a lattice $lat(\perp_i)$ subsuming $\perp_i$. Here "best" is in terms of some objective function such as $pos(h_i) - neg(h_i) - size(h_i)$, where pos and neg are functions that compute the positive and negative examples covered by a clause h_i[3] and $size$ is a function that computes some measure of clause size. Once this best clause h_i in the lattice $lat(\perp_i)$ is found, all positive examples made redundant by $B \wedge h_i$ are removed from further consideration; and the search continues. A number of questions still remain, for example: How is the e_i to be selected? How much effort should we expend on any one search? Current ILP systems answer these in *ad hoc* ways. It is our aim to show how a theory of optimal search described in [4] can be applied to answer such questions.

2 Our Contribution

Specifically, this paper presents a method to find an estimate of the optimal effort (*e.g.* number of nodes to be explored) to be invested in each search subspace (*e.g.* $lat(\perp_i)$). The estimates of effort could then be used to determine the order in which the search subspaces should be searched (that is, the selection of the e_i).

[3] More correctly, these functions should include the background knowledge B.

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

We base our approach on casting the problem of estimating the hyperparameter "effort" as a problem of optimal search. In the discrete case, the basic problem of optimal search [4] is that we are given n bins (each bin being a search subspace in our case) each of which consists of different balls (clauses) and the goal is to find a particular ball (best clause) from the whole lot. Given target distribution $p(j)$ over the bins, allocation function $f(j)$ which specifies the amount of effort to be placed in bin j and conditional probability $b(j, f(j))$ of detecting the target in the j^{th} bin expending $f(j)$ amount of effort assuming target is present in j^{th} bin (see Table 1 for the list of relevant definitions), the total probability of detecting the target is,

$$P[\mathbf{f}] \;=\; \sum_{j=1}^{n} p(j)\; b(j, f(j)) \tag{1}$$

where $\mathbf{f} = [f(1), f(2), \ldots, f(n)]$ is the vector of allocations that needs to be determined. Further, let $c(j, f(j))$ be the cost of applying $f(j)$ amount of effort in bin j. Then the total cost is

$$C[\mathbf{f}] \;=\; \sum_{j=1}^{n} c(j, f(j)) \tag{2}$$

If K is the upperbound on the amount of effort that can be spent to detect the target, the corresponding constraint would be

$$C[\mathbf{f}] \leq K \tag{3}$$

Hence the optimization problem is,

$$\begin{array}{lll} \mathbf{f}^* & = \underset{\mathbf{f}}{argmax}\; P[\mathbf{f}] & \\ \text{s.t. } C[\mathbf{f}] & \leq K & \\ \text{and } \sum_{j=1}^{n} p(j) = 1 & & \end{array} \tag{4}$$

We cast our effort estimation problem for an ILP system as the discrete optimal search problem stated in (4). While the different bottom clause-induced lattices corresponding to search subspaces[4] are natural choices as bins, other choices (such as partitions within a lattice) could also be considered as different bins. The bottom clauses are constructed from positive examples and potentially there can be as many lattices as there are positive examples. The "target" is now the next best clause. Given the search space is $lat(\perp_j)$ constructed from example j, the distribution $p(j)$ indicates how good the lattice $lat(\perp_j)$ is compared to other lattices in terms of containing the target and $b(j, f(j))$ is the probability of detecting the target with efforts $f(j)$ assuming target is present in in lattice $lat(\perp_j)$ (here, the maximum number of nodes to be explored). To be usable, we

[4] Unfortunately, these search subspaces may not be disjoint.

Table 1. Notations in Theory of Optimal Search

Notations	**Description**
$P[\mathbf{f}]$	Probability of detecting the target using **f** allocation function
$p(j)$	Probability that the target can be found in bin j
$f(j)$	Effort allocated to bin j
$b(j, f(j))$	Conditional probability of detecting the target in j^{th} bin given $f(j)$ amount of effort is spent in j^{th} bin and assuming target is present in j^{th} bin
$c(j, f(j))$	Cost of applying $f(j)$ amount of effort in bin j

still need actual distributions $p(j)$ and $b(j, f(j))$. While $p(j)$ can be estimated by sampling from the different lattices[5] $b(j, f(j))$ can be any smooth function that is directly proportional to the number of nodes to be explored $(f(j))$ and inversly proportional to the length of bottom clause $(\perp_j)$). Examples of $b(j, f(j))$ are:

$$\begin{array}{l} (1)\ 1 - e^{-\left(\frac{f(j)}{|\perp_j|}\right)} \\ (2)\ \frac{f(j)}{|\perp_j|} \end{array} \tag{5}$$

where $|\perp_j|$ is number of literals the in bottom clause $\perp_j$ of example j. In this case, the optimization problem would be:

$$\begin{array}{ll} f^* & = \underset{\mathbf{f}}{argmax}\ \sum_{j=1}^{n}(p(j) \cdot b(j, f(j))) \\ \text{s.t. } \sum_{j=1}^{n} f(j) \leq K & \\ \text{and } f(j) \geq 0 \quad \forall\ j & \end{array} \tag{6}$$

The optimal number of nodes $\mathbf{f}^*$ to be explored in each bottom clause lattice, obtained for the above formulation can be used in several ways:

1. Instead of randomly or sequentially picking an example to generate the bottom clause lattice we could use the $\mathbf{f}^*$ information to explore the most promising lattice first to get best clause. The greedy covering approach can then proceed as usual to remove the covered positive examples. We would then recalculate the optimal efforts for remaining example lattices. This process should go on until either the search budget K is exhausted or until no more examples need to be covered.
2. In parallel ILP search, we could rank the different bottom clause lattices based on $\mathbf{f}^*$ and parallelize the search over the $lat(\perp_j)$ for $j \in [1, n]$ using scheduling algorithms to ensure load balancing, where the search process over $lat(\perp_j)$ is restricted to explore $\mathbf{f}^*(j)$ number of nodes.

[5] For example, one could sample a set of clauses and fit a distribution for clause goodness to estimate the maximum value of the population [5].

References

1. Quinlan, J.R., Cameron-Jones, R.M.: FOIL: A midterm report. **667** (1993) 3–20
2. Srinivasan, A.: Aleph Manual
3. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286
4. Stone, L.D.: Theory of Optimal Search. Academic Press (1975)
5. Castillo, E.: Extreme value theory in engineering. Academic Press, New York (1988)