Version Space

Javier Béjar

LSI - FIB

Term 2012/2013
1 Learning logical formulas

2 Version space
   • Introduction
   • Search strategy
   • Algorithm
   • Applications
Learning logical formulas

Version space
- Introduction
- Search strategy
- Algorithm
- Applications
Symbolic learning

- We are used to learn (and teach) symbolically and from our perspective it seems the natural way.
- From all the possible space hypothesis, logical formulas are the best space for this task.
- Restricted to propositional logic, examples are represented by expressions that denote their properties and values.
- This representation is not different from the attribute-value pairs representation that we have defined.
Symbolic learning

- As we mentioned before the size of this hypothesis space is $O(2^{2^n})$
- The main advantage of this is that we can define a partial order among the hypothesis
- Logical formulas form a *lattice* with a partial order defined by the generalization relation

\[ A \vdash B \iff A > B \]

- That order can help in the search process allowing to prune unwanted candidates
Hierarchy of logical formulas
1 Learning logical formulas

2 Version space
   - Introduction
   - Search strategy
   - Algorithm
   - Applications
The Version Space algorithm

- General supervised inductive learning algorithm
- Examples are represented as value–attribute pairs (propositional formulas)
- Explores the hypothesis space using the partial order (general/specific)
- The algorithm will have no preference criteria (bias)
  \[\Rightarrow\] All hypothesis are possible
- (In practice we are going to reduce the hypothesis space to pure conjunctive formulas)
Assumptions

- Learning is obtained by searching in the hierarchy for the concept that best fits the examples.
- Two kinds of examples will be used, the positives (examples of the concept to learn) and the negatives (counterexamples of the concept) (Binary classification).
- We define the version space as the set of all hypothesis consistent with the examples that have been presented so far.
- The goal is to reduce the hypothesis set to a single concept.
Search Strategy

- A breadth first bidirectional search is used
- The more general concepts consistent with the examples are stored in \((G)\) and the more specific ones in \((S)\)
- Positive examples are used to prune the more specific hypothesis
- Negative examples will be used to prune the more general hypothesis
- If the set of learning examples is correct the search will converge
Search strategy

- Set $G = \{\text{Most general hypothesis consistent with the examples}\}$
- Set $S = \{\text{Most specific hypothesis consistent with the examples}\}$
- Adequate generalization and specialization operators for the concept representation language must be chosen
- Positive examples allow to generalize the most specific hypothesis (for instance, deleting conditions)
- Negative examples allow to specialize the most general hypothesis
- Also must hold that $S \subset G$
Searching in the hypothesis space
Candidate elimination algorithm (I)

Initialize $G$ to the most general concept
Initialize $S$ to the fist positive example

while there are examples

if it is a positive example ($p$)

* Delete from $G$ any hypothesis inconsistent with $p$
  (Concepts from $G$ that do not include $p$)

* for each concept from $S$ inconsistent with $p$ ($s$)
  - Delete $s$
  - Add to $S$ all minimal generalizations of $s$ that are consistent with $p$ and an element from $G$ is more general than them

* Delete from $S$ all concepts more general than any from $S$
Candidate elimination algorithm (II)

if it is a negative example (\(n\))
  * Delete from \(S\) any hypothesis inconsistent with \(n\)
    (Concepts from \(S\) that include \(n\))
  * For each concept from \(G\) inconsistent with \(n\) (\(g\))
    - Delete \(g\)
    - Add to \(G\) all minimal specializations of \(g\) that are consistent with \(n\) and an element from \(S\) is more specific that them
  * Delete from \(G\) all concepts less general than any from \(G\)
end while

if \(G=S\) and both have only one element this is the goal concept
Otherwise the set of examples is inconsistent
Shortcomings of the algorithm

- The exhaustive search is too costly
- Improvements:
  - To use simpler hypothesis space (some concepts can not be learned)
  - To use heuristics to prune concepts from G and S (give a preference criteria over the hypothesis space, a bias)
- It is not tolerant to misclassified examples (noise)
LEX: An application to symbolic integration

- LEX is a symbolic integrator that learns from experience
- The hypothesis space of LEX is all the algebraic expressions
- Concepts: What integration operators are more adequate for different kinds of indefinite integrals

\[ \text{OP1} : \int rf(x)dx \rightarrow r \int f(x)dx \]
\[ \text{OP2} : \int udv \rightarrow uv - \int vdu \]
\[ \text{OP3} : \int f_1(x) + f_2(x)dx \rightarrow \int f_1(x) + \int f_2(x) \]
LEX: An application to symbolic integration

- The system is able to generate problems and label each operator depending on its success in solving a specific kind of integral as positive or negative example of application.
- Each operator appears has one or more version spaces (disjunction).
- The version spaces are modified with the new positive or negative examples of application of operators.
- If an expression is inside a version space of an operator this means that could be applicable to solve the integration of the expression.
LEX: Example

EV OP2

\[ \int f_1(x) f_2(x) \, dx \]
\[ \int \text{pol}(x)f_2(x) \, dx \]
\[ \int f_1(x) \text{trig}(x) \, dx \]
\[ \int \text{pol}(x) \text{sen}(x) \, dx \]
\[ \int 3x \text{trig}(x) \, dx \]
\[ \int 3x \text{sen}(x) \, dx \]