P

PAC Identification

▸ PAC Learning

PAC Learning

Thomas Zeugmann
Hokkaido University, Sapporo, Japan

Synonyms

Distribution-free learning; PAC identification; Probably approximately correct learning

Motivation and Background

A very important learning problem is the task of *learning a concept*. ▸ Concept learning has attracted much attention in learning theory. For having a running example, we look at humans who are able to distinguish between different "things," e.g., chair, table, car, airplane, etc. There is no doubt that humans have to learn how to distinguish "things." Thus, in this example, each concept is a thing. To model this learning task, we have to convert "real things" into *mathematical descriptions of things*. One possibility to do this is to fix some language to express a *finite* list of properties. Afterward, we decide which of these properties are relevant for the particular things we want to deal with and which of them have to be fulfilled or not to be fulfilled, respectively. The list of properties comprises qualities or traits such as "has four legs," "has wings," "is green," "has a backrest," "has a seat," etc. So these properties can be regarded as Boolean predicates, and, provided the list of properties is large enough, each thing can be described by a conjunction of these predicates. For example, a chair is described by "has four legs and has a backrest and has a seat and has no wings." Note that the color is not relevant and thus, "is green" has been omitted.

Assume that we have n properties, where n is a natural number. In the easiest case, we can denote the n properties by Boolean variables $x_1, \dots, x_n$, where $range(x_j) \subseteq \{0, 1\}$ for $j = 1, \dots, n$. The semantics is then obviously defined as follows. Setting $x_j = 1$ means property j is fulfilled, while $x_j = 0$ refers to property j is not fulfilled. Now, setting $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$ (set of literals), we can express each thing as a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*.

Therefore, formally we have as *learning domain* (also called ▸ instance space) the set of all Boolean vectors of length n, i.e., $\{0, 1\}^n$, and, in the learner's world, each thing (concept) is just a particular subset of $\{0, 1\}^n$. As far as our example is concerned, the concept chair is then the set of all Boolean vectors for which the monomial "has four legs and has a backrest and has a seat and has no wings" evaluates to 1.

Furthermore, it is usually assumed that the concept c to be learned (the target concept) is taken from a prespecified class $\mathcal{C}$ of possible

© Springer Science+Business Media New York 2017
C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

concepts called the *concept class*. In our example above, the concept class is the set of all concepts describable by a monomial. Consequently, we see that formally learning a concept is equivalent to identifying (exact or approximately) a set from a given set of possibilities by *learning* a suitable description (synonymously called representation) of it.

As in complexity theory, we usually assume that the representations are reasonable ones. Then they can be considered as strings over some fixed alphabet and the set of representations constitutes the ▸ representation language. Note that a concept may have more than one representation in a given representation language (and should have at least one) and that there may be different representation languages for one and the same concept class. For example, every Boolean function can be expressed as a ▸ conjunctive normal form (CNF) and as a ▸ disjunctive normal form (DNF), respectively. For a fixed representation language, the *size* of a concept is defined to be the length of a shortest representation for it. Since we are interested in a model of *efficient* learning, usually the following additional requirements are made: given any string over the underlying alphabet, one can decide in time polynomial in the length of the string whether or not it is a representation. Furthermore, given any element x from the underlying learning domain and a representation r for any concept, one can uniformly decide in time polynomial in the length of both inputs whether or not x belongs to the concept c described by r.

So, we always have a representation language used to define the concept class. As we shall see below, it may be advantageous to choose a possibly different representation language used by the learner. The class of all sets described by this representation language is called ▸ hypothesis space (denoted by $\mathcal{H}$), and the elements of it are said to be hypotheses (commonly denoted by h).

The *learner* is specified to be an algorithm. Further details are given below. We still have to specify the information source, the criterion of success, the hypothesis space, and the prior knowledge in order to define what PAC learning is.

The abbreviation PAC stands for *probably approximately correct* and the corresponding learning model has been introduced by Valiant (1984), while its name was dubbed by Angluin (1988). Valiant's (1984) pioneering paper triggered a huge amount of research the results of which are commonly called computational learning theory (see also the COLT and ALT conference series). Comprehensive treatises of this topic include Anthony and Biggs (1992), Kearns and Vazirani (1994), as well as Natarajan (1991).

Informally, this means that the learner has to find, on input, a randomly drawn set of labeled examples (called *sample*), with high probability a hypothesis such that the error of it is small. Here the error is measured with respect to the same probability distribution D with respect to which the examples are drawn.

Let $X \neq \emptyset$ be any learning domain and let $\mathcal{C} \subseteq \wp(X)$ be any nonempty concept class (here $\wp(X)$ denotes the power set of X). If X is infinite, we need some mild measure theoretic assumptions to ensure that the probabilities defined below exist. We refer to such concept classes as *well-behaved* concept classes. In particular, each $c \in \mathcal{C}$ has to be a Borel set. For a more detailed discussion, see Blumer et al. (1989).

Next, we formally define the *information source*. We assume any unknown probability distribution D over the learning domain X. No assumption is made concerning the nature of D and the learner has no knowledge concerning D. There is a *sampling oracle* $EX(\ )$, which has no input. Whenever $EX(\ )$ is called, it draws an element $x \in X$ according to D and returns the element x together with an indication of whether or not x belongs to the target concept c. Thus, every example returned by $EX(\ )$ may be written as $(x, c(x))$, where $c(x) = 1$ if $x \in c$ (positive examples) and $c(x) = 0$ otherwise (negative examples). If we make s calls to the example $EX(\ )$, then the elements $x_1, \ldots, x_s$ are drawn independently from one another. Thus, the resulting probability distribution over all s-tuples of elements from X is the s-fold product distribution of D, i.e.,

$$\Pr(x_1, \ldots, x_s) = \prod_{i=1}^{s} D(x_i)\ , \qquad (1)$$

where $\Pr(A)$ denotes the probability of event A. Hence, the information source for a target concept c is any randomly drawn s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\,)$.

The *criterion of success*, i.e., *probably approximately correct* learning, is parameterized with respect to two quantities, the *accuracy parameter* ε, and the *confidence parameter* δ, where $\varepsilon, \delta \in (0, 1)$. Next, we define the *difference* between two sets $c,\ c' \subseteq X$ with respect to the probability distribution D as

$$d(c, c') = \sum_{x \in c \triangle c'} D(x),$$

where $c \triangle c'$ denotes the symmetric difference, i.e., $c \triangle c' = c \setminus c' \cup c' \setminus c$. We say that hypothesis h is an *ε-approximation* of a concept c, if $d(c, h) \leq \varepsilon$. A learner is *successful*, if it computes an ε-approximation of the target concept, and it should do so with probability at least $1 - \delta$.

The *hypothesis space* $\mathcal{H}$ is any set such that $\mathcal{C} \subseteq \mathcal{H}$, and the only *prior knowledge* is that the target concept is from the concept class.

A further important feature of the PAC learning model is the demand to learn efficiently. Usually, in the PAC learning model, the efficiency is measured with respect to the number of examples needed and the amount of computing time needed, and in both cases the requirement is to learn with an amount that is polynomial in the "size of the problem." In order to arrive at a meaningful definition, one has to discuss the problem size and, in addition, to look at the asymptotic difficulty of the learning problem. That is, instead of studying the complexity of some fixed learning problem, we always look at infinite sequences of similar learning problems. Such infinite sequences are obtained by allowing the size (dimension) of the learning domain to grow or by allowing the complexity of the concepts considered to grow. In both cases we use n to denote the relevant parameter.

Definition

A learning method $\mathcal{A}$ is said to *probably approximately correctly learn a target concept c with respect to a hypothesis space $\mathcal{H}$* and with sample complexity $s = s(\varepsilon, \delta)$ (or $s = s(\varepsilon, \delta, n)$), if for any distribution D over X and for all $\varepsilon, \delta \in (0, 1)$, it makes s calls to the oracle $EX(\ )$, and after having received the answers produced by $EX(\ )$ (with respect to the target c), it always stops and outputs a representation of a hypothesis $h \in \mathcal{H}$ such that

$$\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta.$$

A learning method $\mathcal{A}$ is said to *probably approximately correctly identify a target concept class $\mathcal{C}$ with respect to a hypothesis space $\mathcal{H}$* and with sample complexity $s = s(\varepsilon, \delta)$, if it probably approximately correctly identifies every concept $c \in \mathcal{C}$ with respect to $\mathcal{H}$ and with sample complexity s.

A learning method $\mathcal{A}$ is said to be *efficient*, if there exists a polynomial *pol* such that the running time of $\mathcal{A}$ and the number s of examples seen are at most $pol(1/\varepsilon, 1/\delta, n)$.

Remarks

This looks complicated, and so, some explanation is in order. First, the inequality

$$\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta$$

says that with high probability (quantified by δ), there is not too much difference (quantified by ε) between the conjectured concept (described by h) and the target c. Formally, let $\mathcal{A}$ be any fixed learning method, and let c be any fixed target concept. For any fixed $\varepsilon, \delta \in (0, 1)$, let $s = s(\varepsilon, \delta)$ be the actual sample size. We have to consider all possible outcomes of $\mathcal{A}$ when run on every labeled s-sample $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$ returned by $EX(\,)$. Let $h(S(c, \bar{x}))$ be the hypothesis produced by $\mathcal{A}$ when processing $S(c, \bar{x})$. Then we have to consider the set W of all s-tuples over X such that $d(c, h(S(c, \bar{x}))) \leq \varepsilon$. The condition $\Pr(d(c, h) \leq \varepsilon) \geq 1 - \delta$ can now be formally rewritten as $\Pr(W) \geq 1 - \delta$. Clearly, one has to require that

$\Pr(W)$ is well defined. Note that the sample size is *not* allowed to depend on the distribution D.

To exemplify this approach, recall that our set of all concepts describable by a monomial over $\mathcal{L}_n$ refers to the set of all things. We consider a hypothetical learner (e.g., a student, a robot) that has to learn the concept of a chair. Imagine that the learner is told by a teacher whether or not particular things visible by the learner are instances of a chair. What things are visible depends on the environment the learner is in. The formal description of this dependence is provided by the unknown distribution D. For example, the learner might be led to a kitchen, a sitting room, a bookshop, a beach, etc. Clearly, it would be unfair to teach the concept of a chair in a bookshop and then testing the learning success at a beach. Thus, the learning success is measured with respect to the same distribution D with respect to which the sampling oracle has drawn its examples. However, the learner is required to learn with respect to any distribution. That is, independently of whether the learner is led to a kitchen, a bookshop, a sitting room, a beach, etc., it has to learn with respect to the place it has been led to. The sample complexity refers to the amount of information needed to ensure successful learning. Clearly, the smaller the required distance of the hypothesis produced and the higher the confidence desired, the more examples are usually needed. But there might be atypical situations. To have an extreme example, the kitchen the learner is led to turned out to be empty. Since the learner is required to learn with respect to a typical kitchen (described by the distribution D), it may well fail under this particular circumstance. Such failure has to be restricted to atypical situations, and this is expressed by demanding the learner to be successful with confidence $1 - \delta$.

This corresponds to real-life situations. For example, a student who has attended a course in learning theory might well suppose that she is examined in learning theory and not in graph theory. However, a good student, say in computer science, has to pass all examinations successfully, independently of the particular course attended. That is, she must successfully pass examinations in computability theory, complexity theory, cryptology, parallel algorithms, etc. Hence, she has to learn a whole concept class. The sample complexity refers to the time of interaction performed by the student and teacher. Also, the student may come up with a different representation of the concepts taught than the teacher. If we require $\mathcal{C} = \mathcal{H}$, then the resulting model is referred to as *proper* PAC learning.

The Finite Case

Having reached this point, it is natural to ask which concept classes are (efficiently) PAC learnable. We start with the finite case, i.e., learning domains X of finite cardinality. As before, the s-sample of c generated by $\bar{x}$ is denoted by $S(c, \bar{x}) = (x_1, c(x_1), \ldots, x_s, c(x_s))$. A hypothesis $h \in \mathcal{H}$ is called *consistent* for an s-sample $S(c, \bar{x})$, if $h(x_i) = c(x_i)$ for all $1 \leq i \leq s$. A learner is said to be *consistent* if all its outputs are consistent hypotheses. Then the following strategy (also known as ▸ Occam's razor) may be used to design a PAC learner:

(1) Draw a sufficiently large sample from the oracle $EX(\,)$, say s examples.
(2) Find some $h \in \mathcal{H}$ that is consistent with all the s examples drawn.
(3) Output h.

This strategy has a couple of remarkable features. First, provided the learner can find a consistent hypothesis, it allows for a uniform bound of the number of examples needed. That is,

$$s \geq \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right) \tag{2}$$

examples will always suffice (here $|S|$ denotes the cardinality of any set S).

The first insight obtained here is that increasing the confidence is exponentially cheaper than reducing the error.

Second, we see why we have to look at the asymptotic difficulty of the learning problem. If we fix $\{0, 1\}^n$ as learning domain and define $\mathcal{C}$ to be the set of all concepts describable by a Boolean function, then there are 2^{2^n} many concepts over $\{0,1\}^n$. Consequently, $\ln|\mathcal{H}| = O(2^n)$ resulting in a sample complexity that is for sure infeasible if $n \geq 50$. Thus, we set $X_n = \{0,1\}^n$, consider $\mathcal{C}_n \subseteq \wp(X_n)$, and study the relevant learning problem for $(X_n, \mathcal{C}_n)_{n\geq 1}$. So, finite means that all X_n are finite.

Third, using Inequality (2), it is not hard to see that the set of all concepts over $\{0,1\}^n$ that are describable by a monomial is efficiently PAC learnable. Let $\mathcal{H}_n$ be the set of all monomials containing each literal from $\mathcal{L}_n$ at most once plus the conjunction of all literals (denoted by m_{all}) (representing the empty concept). Since there are $3^n + 1$ monomials in $\mathcal{H}_n$, by (2), we see that $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ many examples suffice. Note that $2n$ is also an upper bound for the size of any concept from $\mathcal{H}_n$.

Thus it remains to deal with the problem to find a consistent hypothesis. The learning algorithm can be informally described as follows. After having received the s examples, the learner disregards all negative examples received and uses the positive ones to delete all literals from m_{all} that evaluate to 0 on at least one positive example. It then returns the conjunction of the literals not deleted from m_{all}. After a bit of reflection, one verifies that this hypothesis is consistent. This is essentially Haussler's (1987) Wholist algorithm and its running time is $O(1/\varepsilon \cdot (n^2 + \ln(1/\delta)))$. Also note that the particular choice of the representation for the empty concept was crucial here. It is worth noticing that the sample complexity is tight up to constant factors.

Using similar ideas one can easily show that the class of all concepts over $\{0,1\}^n$ describable by a k-CNF or k-DNF (where k is fixed) is efficiently PAC learnable by using as hypothesis space all k-CNF and k-DNF, respectively (cf. Valiant 1984). Note that a k-CNF is a conjunctive normal form in which each clause has at most k literals, and a k-DNF is a disjunctive normal form in which each monomial has at most k literals.

So, what can we say in general concerning the problem to find a consistent hypothesis? Answering this question gives us the insight to understand why it is sometimes necessary to choose a hypothesis space that is different from the target concept class. This phenomenon was discovered by Pitt and Valiant (1988). First, we look at the case where we have to efficiently PAC learn any $\mathcal{C}_n$ with respect to $\mathcal{C}_n$. Furthermore, an algorithm is said to *solve the consistency problem for* $\mathcal{C}_n$ if, on input any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$, it outputs a hypothesis consistent with $S(c, \bar{x})$ provided there is one, and "there is no consistent hypothesis," otherwise.

Since we are interested in efficient PAC learning, we have to make the assumption that $|\mathcal{C}_n| \leq 2^{pol(n)}$ (cf. Inequality (2)). Also, it should be noted that for the proof of the following result, the requirement that $h(x)$ is polynomial time computable is essential (cf. our discussion of representations). Furthermore, we need the notion of an $\mathcal{RP}$-algorithm (randomized polynomial time). The input is any s-sample $S(c, \bar{x})$, where $c \subseteq X_n$ and the running time is uniformly bounded by a polynomial in the length of the input. In addition to its input, the algorithm can flip a coin in every step of its computation and then branch in dependence of the outcome of the coin flip. If there is no hypothesis consistent with $S(c, \bar{x})$, the algorithm must output "there is no consistent hypothesis," independently of the sequence of coin flips made. If there is a hypothesis consistent with $S(c, \bar{x})$, then the $\mathcal{RP}$-algorithm is allowed to fail with probability at most δ.

Interestingly, under the assumptions made above, then one can prove the following equivalence for efficient PAC learning.

PAC learning $\mathcal{C}_n$ with respect to $\mathcal{C}_n$ is equivalent to solving the consistency problem for $\mathcal{C}_n$ by an $\mathcal{RP}$-algorithm.

We continue by looking at the class of all concepts describable by a k-term DNF$_n$. A term is a conjunction of literals from $\mathcal{L}_n$, and a k-term DNF$_n$ is a disjunction of at most k terms. Consequently, there are $(3^n + 1)^k$ many k-term DNFs and thus the condition $|\mathcal{C}_n| \leq 2^{pol(n)}$ is fulfilled. Then one can show the following (see Pitt and Valiant 1988).

For all integers $k \geq 2$, if there is an algorithm that efficiently learns k-term DNF$_n$ *with respect to* k-term DNF$_n$, *then $\mathcal{RP} = \mathcal{NP}$.*

For a formal definition of the complexity classes $\mathcal{RP}$ and $\mathcal{NP}$, we refer the reader to Arora and Barak (2009). This result is proved by showing that deciding the consistency problem for k-term DNF$_n$ is $\mathcal{NP}$-complete for every $k \geq 2$. The difference between deciding and solving the consistency problem is that we only have to decide if there is a consistent hypothesis in k-term DNF$_n$. However, by the equivalence established above, we know that an efficient proper PAC learner for k-term DNF$_n$ can be transformed into an $\mathcal{RP}$-algorithm even solving the consistency problem. It should be noted that we currently do not know whether or not $\mathcal{RP} = \mathcal{NP}$ (only $\mathcal{RP} \subseteq \mathcal{NP}$ has been shown), but it is widely believed that $\mathcal{RP} \neq \mathcal{NP}$. On the other hand, it easy to see that every concept describable by a k-term DNF$_n$ is also describable by a k-CNF$_n$ (but not conversely). Thus, we can finally conclude that there is an algorithm that efficiently PAC learns k-term DNF$_n$ with respect to k-CNF$_n$.

For more results along this line of research, we refer the reader to Pitt and Valiant (1988), Blum and Singh (1990), and Jerrum (1994). As long as we do not have more powerful lower bound techniques allowing one to separate the relevant complexity classes $\mathcal{RP}$ and $\mathcal{NP}$ or $\mathcal{P}$ and $\mathcal{NP}$, no unconditional negative result concerning PAC learning can be shown. Another approach to show hardness results for PAC learning is based on cryptographic assumptions (cf., e.g., Kearns and Valiant 1989, 1994), and recently one has also tried to base cryptographic assumptions on the hardness of PAC learning (cf., e.g., Xiao (2009) and the references therein).

Further positive results comprise the efficient proper PAC learnability of rank k ▸ decision trees (cf. Ehrenfeucht and Haussler 1989) and of k-▸ decision lists for any fixed k (cf. Rivest 1987).

Finally, it must be noted that the bounds on the sample size obtained via Inequality (2) are *not* the best possible. Sometimes, better bounds can be obtained by using the ▸ VC dimension (see Inequality (4) below).

The Infinite Case

Let us start our exposition concerning infinite concept classes with an example due to Blumer et al. (1989). Consider the problem of learning concepts such as "medium built" animals. For the sake of presentation, we restrict ourselves to the parameters "weight" and "length." To describe "medium built," we use intervals "from-to." For example, a medium built cat might have a weight ranging from 3 to 7 kg and a length ranging from 25 cm to 50 cm. By looking at a finite database of randomly chosen animals giving their respective weight and length and their *classification* (medium built or not), we want to form a rule that *approximates* the true concept of "medium built" for each animal under consideration.

This learning problem can be formalized as follows. Let $X = \mathbb{E}^2$ be the two-dimensional Euclidean space, and let $\mathcal{C} \subseteq \wp(\mathbb{E}^2)$ be the set of all axis-parallel rectangles, i.e., products of intervals on the x-axis with intervals on the y-axis. Furthermore, let D be any probability distribution over X. Next we show that $\mathcal{C}$ is efficiently PAC learnable with respect to $\mathcal{C}$ by the following Algorithm **LR** (cf. Blumer et al. 1989):

Algorithm **LR**: "On input any $\varepsilon, \delta \in (0,1)$, call the oracle $EX(\ )$ s times, where $s = 4/\varepsilon \cdot \ln(4/\delta)$. Let $(r_1, c(r_1), r_2, c(r_2), \ldots, r_s, c(r_s))$ be the s-sample returned by $EX(\ )$, where $r_i = (x_i, y_i)$, $i = 1, \ldots s$.

$$\begin{aligned} \text{Compute } x_{\min} &= \min\{x_i \,|\, 1 \leq i \leq s,\ c(r_i) = 1\} \\ x_{\max} &= \max\{x_i \,|\, 1 \leq i \leq s,\ c(r_i) = 1\} \\ y_{\min} &= \min\{y_i \,|\, 1 \leq i \leq s,\ c(r_i) = 1\} \\ y_{\max} &= \max\{y_i \,|\, 1 \leq i \leq s,\ c(r_i) = 1\} \end{aligned}$$

Output $h = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. In case there is no positive example, return $h = \emptyset$.
end."

It remains to show that Algorithm **LR** PAC learns the concept class $\mathcal{C}$ with respect to $\mathcal{C}$. Let $c = [a,b] \times [c,d]$ be the target concept. Since **LR** computes its hypothesis from positive examples, only, we get $h \subseteq c$. That is, h is consistent. We have to show that $d(c,h) \leq \varepsilon$ with probability at least $1 - \delta$. We distinguish the following cases.

Case 1. $D(c) \le \varepsilon$

Then $d(c,h) = \sum_{r\in c\triangle h} D(r) = \sum_{r\in c\setminus h} D(r) \le D(c) \le \varepsilon$.

Hence, in this case we are done.

Case 2. $D(c) > \varepsilon$

We define four minimal side rectangles within c that each cover an area of probability of at least $\varepsilon/4$. Let

$\mathit{Left} = [a,x] \times [c,d]$, where $x = \inf\{\tilde{x} \mid D([a,\tilde{x}] \times [c,d]) \ge \varepsilon/4\}$,

$\mathit{Right} = [z,b] \times [c,d]$, where $z = \inf\{\tilde{x} \mid D([\tilde{x},b] \times [c,d]) \ge \varepsilon/4\}$,

$\mathit{Top} = [a,b] \times [y,d]$, where $y = \inf\{\tilde{x} \mid D([a,b] \times [\tilde{x},d]) \ge \varepsilon/4\}$, and

$\mathit{Bottom} = [a,b] \times [c,t]$, where $t = \inf\{\tilde{x} \mid D([a,b] \times [c,\tilde{x}]) \ge \varepsilon/4\}$.

All those rectangles are contained in c, since $D(c) > \varepsilon$. If the sample size is s, the probability that a *particular* rectangle from $\{\mathit{Left}, \mathit{Right}, \mathit{Top}, \mathit{Bottom}\}$ contains no positive example is at most $(1-\varepsilon/4)^s$. Thus, the probability that *some* of those rectangles does not contain any positive example is at most $4(1-\varepsilon/4)^s$. Hence, incorporating $s = 4/\varepsilon \cdot \ln(4/\delta)$ gives

$$4(1-\varepsilon/4)^s < 4e^{-(\varepsilon/4)s} = 4e^{-\ln(4/\delta)} = \delta\,.$$

Therefore, with probability at least $1-\delta$, *each* of the four rectangles *Left*, *Right*, *Top*, and *Bottom* contains a positive example. Consequently, we get

$$d(c,h) = \sum_{r\in c\triangle h} D(r) = \sum_{r\in c\setminus h} D(r) = D(c) - D(h)\,.$$

Furthermore, by construction

$$D(h) \ge D(c) - D(\mathit{Left}) - D(\mathit{Right}) - D(\mathit{Top}) - D(\mathit{Bottom}) \ge D(c) - \varepsilon\,,$$

and hence $d(c,h) \le \varepsilon$.

Having reached this point, it is only natural to ask what makes infinite concept classes PAC learnable. Interestingly, there is a single parameter telling us whether or not a concept class is PAC learnable. This is the so-called Vapnik-Chervonenkis dimension commonly abbreviated as ▶ VC dimension. In our example of axis-parallel rectangles, the VC dimension of $\mathcal{C}$ is 4.

In order to state this result, we have to exclude trivial concept classes. A concept class $\mathcal{C}$ is said to be *trivial* if $|\mathcal{C}| = 1$ or $\mathcal{C} = \{c_1, c_2\}$ with $c_1 \cap c_2 = \emptyset$ and $X = c_1 \cup c_2$. A concept class $\mathcal{C}$ is called nontrivial if $\mathcal{C}$ is not trivial. Then Blumer et al. (1989) showed the following:

A nontrivial well-behaved concept class is PAC learnable if and only if its VC dimension is finite.

Moreover, if the VC dimension is finite, essentially the same strategy as in the finite case applies, i.e., it suffices to construct a consistent hypothesis from $\mathcal{C}$ (or from a suitably chosen hypothesis space $\mathcal{H}$ which must be well behaved) in random polynomial time.

So, it remains to estimate the sample complexity. Let d be the VC dimension of $\mathcal{H}$. Blumer et al. (1989) showed that

$$s \ge \max\left\{\frac{4}{\varepsilon}\log\frac{2}{\delta},\ \frac{8d}{\varepsilon}\log\frac{13}{\varepsilon}\right\} \tag{3}$$

examples do suffice. This upper bound has been improved by Anthony et al. (1990) to

$$s \ge \frac{1}{\varepsilon(1-\sqrt{\varepsilon})}\left[\log\left(\frac{d/(d-1)}{\delta}\right) + 2d\log\left(\frac{6}{\varepsilon}\right)\right]. \tag{4}$$

Based on the work of Blumer et al. (1989) (and the lower bound they gave), Ehrenfeucht et al. (1988) showed that if $\mathcal{C}$ is nontrivial, then no learning function exists (for any $\mathcal{H}$) if $s < \frac{1-\varepsilon}{2\varepsilon}\log\frac{2}{\delta} + \frac{d-1}{64\varepsilon}$. These results give a precise characterization of the number of examples needed (apart from the gap of a factor of $O(\log \frac{1}{\varepsilon}))$ in terms of the VC dimension. Also note the sharp dichotomy here, either any consistent learner (computable or not) will do or no learner at all exists.

Two more remarks are in order here. First, these bounds apply to *uniform* PAC learning, i.e., the learner is taking ε and δ as input, only. As outlined in our discussion just before we gave the formal definition of PAC learning, it is meaningful to look at the asymptotic difficulty of learning. In the infinite case, we can increment the dimension n of the learning domain as we did in the finite case. We may set $X_n = \mathbb{E}^n$ and then consider similar concept classes $\mathcal{C}_n \subseteq \wp(X_n)$. For example, the concept classes similar to axis-parallel rectangles are axis-parallel parallelepipeds in $\mathbb{E}^n$. Then the VC dimension of $\mathcal{C}_n$ is $2n$, and all that is left is to add n as input to the learner and to express d as a function of n in the bound (4). Clearly, the algorithm **LR** can be straightforwardly generalized to a learner for $(X_n, \mathcal{C}_n)_{n\geq 1}$.

Alternatively, we use n to parameterize the complexity of the concepts to be learned. As an example consider $X = \mathbb{E}$ and let $\mathcal{C}_n$ be the set of all unions of at most n (closed or open) intervals. Then the ▸ VC dimension of $\mathcal{C}_n$ is $2n$, and one can design an efficient learner for $(X, \mathcal{C}_n)_{n\geq 1}$. Another example is obtained for $X = \mathbb{E}^2$ by defining $\mathcal{C}_n$ to be the class of all convex polygons having at most n edges (cf. Linial et al. 1991).

Second, all the results discussed so far are dealing with *static sampling*, i.e., any sample containing the necessary examples is drawn before any computation is performed. So, it is only natural to ask what can be achieved when *dynamic sampling* is allowed. In dynamic sampling mode, a learner alternates between drawing examples and performing computations. Under this sampling mode, even concept classes having an infinite VC dimension are learnable (cf. Linial et al. 1991 and the references therein). The main results in this regard are that enumerable concept classes and decomposable concept classes are PAC learnable when using dynamic sampling.

Let us finish the general exposition of PAC learning by pointing to another interesting insight, i.e., learning is in some sense data compression. As we have seen, finding consistent hypotheses is a problem of fundamental importance in the area of PAC learning. Clearly, the more expressive the representation language for the hypothesis space, the easier it may be to find a consistent hypothesis, but it may be increasingly difficult to say something concerning its accuracy (in machine learning this phenomenon is also known as the over-fitting problem). At this point, ▸ Occam's razor comes into play. If there is more than one explanation for a phenomenon, then Occam's razor requires to "prefer simple explanations." So, an Occam algorithm is an algorithm which, given a sample of the target concept, outputs a consistent and relatively simple hypothesis. That is, it is capable of some *data compression*. Let us first look at the Boolean case, i.e., $X_n = \{0,1\}^n$. Then an Occam algorithm is a randomized polynomial time algorithm $\mathcal{A}$ such that there is a polynomial p and a constant $\alpha \in [0,1)$ fulfilling the following demands:

For every $n \geq 1$, every target concept $c \in \mathcal{C}_n$ of size at most m and every $\varepsilon \in (0,1)$, on input any s-sample for c, algorithm $\mathcal{A}$ outputs with probability at least $1-\varepsilon$ the representation of a consistent hypothesis from $\mathcal{C}_n$ having size at most $p(n, m, 1/\varepsilon) \cdot s^{\alpha}$.

So, the parameter $\alpha < 1$ expresses the amount of compression required. If we have such an Occam algorithm, then $(X_n, \mathcal{C}_n)$ is properly PAC learnable (cf. Blumer et al. 1987). The proof is based on the observations that a hypothesis with large error is unlikely to be consistent with a large sample and that there are only few short hypotheses. If we replace in the definition of an Occam algorithm the demand on the existence of a short hypothesis by the existence of a hypothesis space having a small VC dimension, then a similar result can be obtained for the continuous case (cf. Blumer et al. 1989). To a certain extent, the converse is also true, that is, under quite general conditions, PAC learnability implies the existence of an Occam algorithm. We refer the reader to Kearns and Vazirani (1994) for further details.

Variations

Further variations of PAC learning are possible and have been studied. So far, we have only considered one sampling oracle. Hence, a natural variation is to have two sampling oracles $EX_+(\ )$ and $EX_-(\ )$ and two distributions D_+ and D_-, i.e., one for positive examples and one for negative examples. Clearly, further natural variations are possible. A larger number of them has been shown to be roughly equivalent and we refer the reader to Haussler et al. (1991) for details.

We continue with another natural variation that turned out to have a fundamental impact to the whole area of machine learning, i.e., weak learning.

Weak Learning

An interesting variation of PAC learning is obtained if we weaken the requirements concerning the confidence and the error. That is, instead of requiring the PAC learner to succeed for every ε and δ, one may relax this demand as follows. We only require the learner to succeed for $\varepsilon = 1/2 - 1/pol(n)$ (n is as above) and $\delta = 1/poly(n)$ (n is as above), where *pol* and *poly* are any two fixed polynomials. The resulting model is called *weak* PAC learning.

Quite surprisingly, Schapire (1990) could prove that every weak learner can be efficiently transformed into an ordinary PAC learner. While it is not too difficult to *boost* the confidence, *boosting* the error is much more complicated and has subsequently attracted a lot of attention. We refer the reader to Schapire (1990, 1999) as well as Kearns and Vazirani (1994) and the references therein for a detailed exposition. Interestingly enough, the techniques developed to prove the equivalence of weak PAC learnability and PAC learnability have an enormous impact to machine learning and may be subsumed under the title ▸ boosting.

Relations to Other Learning Models

Finally, we point out some relations of PAC learning to other learning models. Let us start with the mistake bound model also called online prediction model. The mistake bound model has its roots in ▸ inductive inference and was introduced by Littlestone (1988). It is conceptionally much simpler than the PAC model, since it does not involve probabilities. For the sake of presentation, we assume a finite learning domain X_n and any $\mathcal{C}_n \subseteq \wp(X_n)$ here.

In this model the following scenario is repeated indefinitely. The learner receives an instance x and has to predict $c(x)$. Then it is given the true label $c(x)$. If the learner's prediction was incorrect, then a *mistake* occurred. The learner is successful, if the total number of mistakes is finite. In order to make this learning problem nontrivial, one additionally requires that there is a polynomial *pol* such that for every $c \in \mathcal{C}_n$ and any ordering of the examples, the total number of mistakes is bounded by $pol(n, size(c))$. In the mistake bound model, a learner is said to be efficient if its running time per stage is uniformly polynomial in n and $size(c)$.

Then, the relation to PAC learning is as follows:

If algorithm $\mathcal{A}$ learns a concept class $\mathcal{C}$ in the mistake bound model, then $\mathcal{A}$ also PAC learns $\mathcal{C}$. Moreover, if $\mathcal{A}$ makes at most M mistakes, then the resulting PAC learner needs $\frac{M}{\varepsilon} \cdot \ln \frac{M}{\delta}$ many examples.

So, efficient mistake bound learning translates into efficient PAC learning.

Another interesting relation is obtained when looking at the ▸ query-based learning model, where the only queries allowed are equivalence queries. As pointed out by Angluin (1988, 1992), any learning method that uses equivalence queries only and achieves *exact* identification can be transformed into a PAC learner. The number of equivalence queries necessary to achieve success in the query learning model is polynomially related to the number of calls made to the sample oracle.

However, the converse is not true. This insight led to the definition of a *minimally adequate teacher* (cf. Angluin (1988) and the references therein). In this setting, the teacher answers equivalence queries and membership queries. Maas and Turán (1990) provide a detailed discussion of the relationship between the different models.

These results in turn led to another modification of the PAC model, where the learner is, in addition to the s-sample returned, also allowed to ask membership queries, i.e., PAC learning with membership queries. This and the original PAC learning model may be further modified by restricting the class of probability distributions, e.g., by considering PAC learning (with or without membership queries) with respect to the uniform distribution. Having the additional power of membership queries allowed for a series of positive polynomial time learnability results, e.g., the class of deterministic finite automata (cf. Angluin 1987), monotone DNF formulae (cf. Angluin 1988), polynomial size decision trees (cf. Bshouty 1993), and sparse multivariate polynomials over a field (cf. Schapire and Sellie 1996). Furthermore, Jackson (1997) showed the class of DNF formulae to be PAC learnable with membership queries under the uniform distribution, and Bshouty et al. (2004) presented a modification of Jackson's (1997) algorithm that substantially improves its asymptotic efficiency. Further variations of the PAC learning model are presented in Bshouty et al. (2005).

Let us finish this entry by mentioning that the PAC model has been criticized for two reasons. The first one is the independence assumption, that is, the requirement to learn with respect to any distribution. This is, however, also a very strong part of the theory, since it provides universal performance guarantees. Clearly, if one has additional information concerning the underlying distributions, one may be able to prove better bounds. The second reason is the "noise-free" assumption, i.e., the requirement to the sample oracle to return exclusively correct labels. Clearly, in practice we never have noise-free data. So, one has also studied learning in the presence of noise, and we refer the reader to Kearns and Vazirani (1994) as well as to conference series COLT and ALT for results along this line.

Cross-References

▸ Statistical Machine Translation
▸ Stochastic Finite Learning
▸ VC Dimension

Recommended Reading

Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75(2):87–106

Angluin D (1988) Queries and concept learning. Mach Learn 2(4):319–342

Angluin D (1992) Computational learning theory: survey and selected bibliography. In: Proceedings of the 24th annual ACM symposium on theory of computing. ACM Press, New York, pp 351–369

Anthony M, Biggs N (1992) Computational learning theory. Cambridge tracts in theoretical computer science, vol 30. Cambridge University Press, Cambridge

Anthony M, Biggs N, Shawe-Taylor J (1990) The learnability of formal concepts. In: Fulk MA, Case J (eds) Proceedings of the third annual workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 246–257

Arora S, Barak B (2009) Computational complexity: a modern approach. Cambridge University Press, Cambridge

Blum A, Singh M (1990) Learning functions of k terms. In: Proceedings of the third annual workshop on computational learning theory. Morgan Kaufmann, San Mateo, pp 144–153

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1987) Occam's razor. Inf Process Lett 24(6):377–380

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965

Bshouty NH (1993) Exact learning via the monotone theory. In: Proceedings of the 34rd annual symposium on foundations of computer science. IEEE Computer Society Press, Los Alamitos, pp 302–311

Bshouty NH, Jackson JC, Tamon C (2004) More efficient PAC-learning of DNF with membership queries under the uniform distribution. J Comput Syst Sci 68(1):205–234

Bshouty NH, Jackson JC, Tamon C (2005) Exploring learnability between exact and PAC. J Comput Syst Sci 70(4):471–484

Ehrenfeucht A, Haussler D (1989) Learning decision trees from random examples. Inf Comput 82(3):231–246

Ehrenfeucht A, Haussler D, Kearns M, Valiant L (1988) A general lower bound on the number of

examples needed for learning. In: Haussler D, Pitt L (eds) Proceedings of the 1988 workshop on computational learning theory (COLT'88), 3–5 Aug. MIT/Morgan Kaufmann, San Francisco, pp 139–154
Haussler D (1987) Bias, version spaces and Valiant's learning framework. In: Langley P (ed) Proceedings of the fourth international workshop on machine learning. Morgan Kaufmann, San Mateo, pp 324–336
Haussler D, Kearns M, Littlestone N, Warmuth MK (1991) Equivalence of models for polynomial learnability. Inf Comput 95(2):129–161
Jackson JC (1997) An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. J Comput Syst Sci 55(3):414–440
Jerrum M (1994) Simple translation-invariant concepts are hard to learn. Inf Comput 113(2):300–311
Kearns M, Valiant L (1994) Cryptographic limitations on learning Boolean formulae and finite automata. J ACM 41(1):67–95
Kearns M, Valiant LG (1989) Cryptographic limitations on learning Boolean formulae and finite automata. In: Proceedings of the 21st symposium on theory of computing. ACM Press, New York, pp 433–444
Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press, Cambridge
Linial N, Mansour Y, Rivest RL (1991) Results on learnability and the Vapnik-Chervonenkis dimension. Inf Comput 90(1):33–49
Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318
Maas W, Turán G (1990) On the complexity of learning from counterexamples and membership queries. In: Proceedings of the 31st annual symposium on foundations of computer science (FOCS 1990), St. Louis, 22–24 Oct 1990. IEEE Computer Society, Los Alamitos, pp 203–210
Natarajan BK (1991) Machine learning: a theoretical approach. Morgan Kaufmann, San Mateo
Pitt L, Valiant LG (1988) Computational limitations on learning from examples. J ACM 35(4):965–984
Rivest RL (1987) Learning decision lists. Mach Learn 2(3):229–246
Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
Schapire RE (1999) Theoretical views of boosting and applications. In: Algorithmic learning theory, 10th international conference (ALT '99), Tokyo, Dec 1999, Proceedings. Lecture notes in artificial intelligence, vol 1720. Springer, pp 13–25
Schapire RE, Sellie LM (1996) Learning sparse multivariate polynomials over a field with queries and counterexamples. J Comput Syst Sci 52(2):201–213
Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142
Xiao D (2009) On basing $ZK \neq BPP$ on the hardness of PAC learning. In: Proceedings of the 24th annual IEEE conference on computational complexity (CCC 2009), Paris, 15–18 July 2009. IEEE Computer Society, Los Alamitos, pp 304–315

PAC-MDP Learning

▸ Efficient Exploration in Reinforcement Learning

Pairwise Classification

▸ Class Binarization

Parallel Corpus

A parallel corpus (pl. corpora) is a document collection composed of two or more disjoint subsets, each written in a different language, such that documents in each subset are translations of documents in each other subset. Moreover, it is required that the translation relation is known, i.e., that given a document in one of the subset (i.e., languages), it is known what documents in the other subset are its translations. The statistical analysis of parallel corpora is at the heart of most methods for ▸ cross-language text mining.

Part of Speech Tagging

▸ POS Tagging

Partially Observable Markov Decision Processes

Pascal Poupart
University of Waterloo, Waterloo, ON, Canada

Synonyms

Belief state Markov decision processes; Dual control; Dynamic decision networks; POMDPs

Definition

A partially observable Markov decision process (POMDP) refers to a class of sequential decision-making problems under uncertainty. This class includes problems with partially observable states and uncertain action effects. A POMDP is formally defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ where $\mathcal{S}$ is the set of states s, $\mathcal{A}$ is the set of actions a, $\mathcal{O}$ is the set of observations o, $T(s, a, s') = \Pr(s'|s, a)$ is the transition function indicating the probability of reaching s' when executing a in s, $Z(a, s', o') = \Pr(o'|a, s')$ is the observation function indicating the probability of observing o' in state s' after executing a, $R(s, a) \in \Re$ is the reward function indicating the (immediate) expected utility of executing a in s, $b_0 = \Pr(s_0)$ is the distribution over the initial state (also known as initial belief), h is the planning horizon (which may be finite or infinite), and $\gamma \in [0, 1]$ is a discount factor indicating by how much rewards should be discounted at each time step. Given a POMDP, the goal is to find a policy to select actions that maximize rewards over the planning horizon.

Motivation and Background

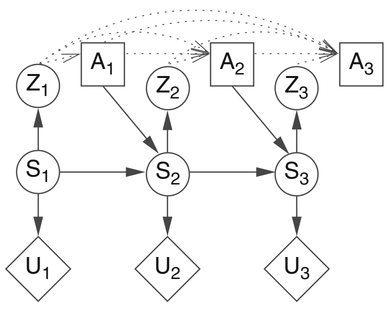
Partially observable Markov decision processes (POMDPs) were first introduced in the Operations Research community (Drake 1962; Aström 1965) as a framework to model stochastic dynamical systems and to make optimal decisions. This framework was later considered by the artificial intelligence community as a principled approach to planning under uncertainty (Kaelbling et al. 1998). Compared to other methods, POMDPs have the advantage of a well-founded theory. They can be viewed as an extension of the well-known, fully observable ▸ Markov decision process (MDP) model (Puterman 1994), which is rooted in probability theory, utility theory, and decision theory. POMDPs do not assume that states are fully observable, but instead that only part of the state features are observable, or more generally, that the observable features are simply correlated with the underlying states. This naturally captures the fact that in many real-world problems, the information available to the decision maker is often incomplete and typically measured by noisy sensors. As a result, the decision process is much more difficult to optimize. POMDP applications include robotics (Pineau and Gordon 2005), assistive technologies (Hoey et al. 2010), health informatics (Hauskrecht and Fraser 2010), spoken dialogue systems (Thomson and Young 2010), and fault recovery (Shani and Meek 2009).

Structure of Model and Solution Algorithms

We describe below the POMDP model, some policy representations, the properties of optimal value functions, and some solution algorithms.

POMDP Model

Figure 1 shows the graphical representation of a POMDP, using the notation of influence diagrams: circles denote random variables (e.g., state variables S_t and observation variables O_t), squares denote decision variables (e.g., action variables A_t), and diamonds denote utility variables (e.g., U_t's). The variables are indexed by time and grouped in time slices, reflecting the fact that each variable may take a different value at each time step. Arcs indicate how nodes influence each other over time. There are two types of arcs: probabilistic and informational arcs. Arcs pointing to a chance node or a utility node indicate

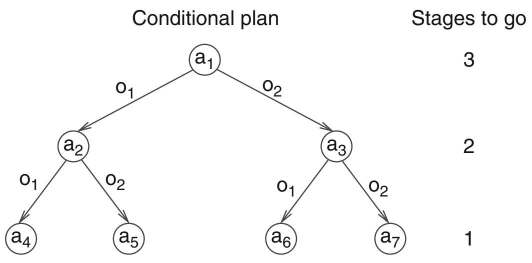


Partially Observable Markov Decision Processes, Fig. 1 POMDP represented as an influence diagram a probabilistic dependency between a child and its parents, whereas arcs pointing to a decision node indicate the information available to the decision maker (i.e., which nodes are observable at the time of each decision). Probabilistic dependencies for the state and observation variables are quantified by the conditional distributions $\Pr(S_{t+1}|S_t, A_t)$ and $\Pr(O_{t+1}|S_{t+1}, A_t)$, which correspond to the transition and observation functions. Note that the initial state variable S_0 does not have any parent, hence its distribution $\Pr(S_0)$ is unconditioned and corresponds to the initial belief b_0 of the decision maker. Probabilistic dependencies for the utility variables are also quantified by a conditional distribution $\Pr(U_t|S_t, A_t)$ such that its expectation $\sum_u \Pr(u|S_t,\ A_t)u = R(S_t,\ A_t)$ corresponds to the reward function.

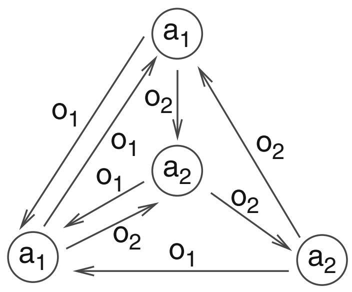
Fully observable MDPs are a special case of POMDPs since they arise when the observation function deterministically maps each state to a different unique observation. POMDPs can also be viewed as ▶ hidden Markov models (HMMs) (Rabiner 1989) extended with decision and utility nodes since the transition and observation distributions essentially define an HMM. POMDPs also correspond to a special case of decision networks called *dynamic decision networks* (Buede 1999) where it is assumed that the transition, observation, and reward functions are *stationary* (i.e., they do not depend on time) and *Markovian* (i.e., the parents of each variable are in the same time slice or immediately preceding time slice).

Policies

Given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, h, \gamma \rangle$ specifying a POMDP, the goal is to find a policy π to select actions that maximize the rewards. The informational arcs indicate that each action *at* can be selected based on the history of past actions and observations. Hence, in its most general form, a policy $\pi : \langle b_0, h_t \rangle \rightarrow a_t$ is a mapping from initial beliefs b_0 and histories $h_t = \langle o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t$r to actions a_t. For a fixed initial belief, the mapping can be represented by a tree such as the one in Fig. 2. We will refer to such policy trees as conditional plans since in general a policy may consist of several conditional plans for different initial beliefs. The execution of a conditional plan follows a branch from the root to some leaf by executing the actions of the nodes traversed and following the edges labeled by the observations received.



Partially Observable Markov Decision Processes, Fig. 2 Three representation of a three-step conditional plan



Partially Observable Markov Decision Processes, Fig. 3 Finite state controller for a simple POMDP with two actions and two observations

Unfortunately, as the number of steps increases, the number of histories grows exponentially and it is infeasible to represent mappings over all such histories. Furthermore, infinite-horizon problems require mappings over arbitrarily long histories, which limit the use of trees to problems with a short horizon. Note, however, that it is possible to have mappings over infinite *cyclic* histories. Such mappings can be represented by a *finite state controller* (Hansen 1997), which is essentially a cyclic graph of nodes labeled by actions and edges labeled by observations (see Fig. 3 for an example). Similar to conditional plans, finite state controllers are executed by starting at an initial node, executing the actions of the nodes traversed, and following the edges of the observations received.

Alternatively, it is possible to summarize histories by a sufficient statistic that encodes all the relevant information from previous actions and observations for planning purposes. Recall that the transition, reward, and observation functions exhibit the Markov property, which means that the outcome of future states, rewards, and observations depend only on the current state and action. If the decision maker knew the current state of the world, then she would have all the desired information to make an optimal action choice. Thus, histories of past actions and observations are only relevant to the extent that they provide information about the current state of the world. Let b_t be the belief of the decision maker about the state of the world at time step t, which we represent by a probability distribution over the state space $\mathcal{S}$. Using Bayes theorem (see Bayes Rules), one can compute the current belief b_t from the previous belief b_{t-1}, previous action a_{t-1}, and current observation o_t:

$$b_t(s') = k \sum_{s \in \mathcal{S}} b_{t-1}(s) \Pr(s'|s, a_{t-1}) \Pr(o_t|a_{t-1}, s') \quad (1)$$

where k denotes a normalizing constant. Hence, a policy π can also be represented as a mapping from beliefs b_t to actions a_t. While this gets around the exponentially large number of histories, the space of beliefs is an $|\mathcal{S}| - 1$-dimensional continuous space, which is also problematic. However, a key result by Smallwood and Sondik (1973) allows us to circumvent the continuous nature of the belief space. But first, let us introduce value functions and then discuss Smallwood and Sondik's solution.

Value Functions

Given a set of policies, we need a mechanism to evaluate and compare them. Roughly speaking, the goal is to maximize the amount of reward earned over time. This loosely defined criterion can be formalized in several ways: one may wish to maximize *total* (accumulated) or *average* reward, *expected* or *worst-case* reward, *discounted* or *undiscounted* reward. The rest of this article assumes an *expected total discounted* reward criterion, since it is by far the most popular in the literature. We define the value $V^\pi(b_0)$ of executing some policy π starting at belief b_0 to be the expected sum of the discounted rewards earned at each time step:

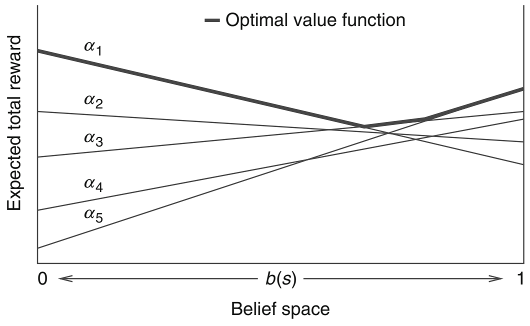
$$V^\pi(b_0) = \sum_{t=0}^{h} \gamma^t \sum_{s \in \mathcal{S}} b_t(s) R(s, \pi, (b_t)) \quad (2)$$

where $\pi(b_t)$ denotes the action prescribed by policy π at belief b_t. A policy π^* is optimal when its value function V^* is at least as high as any other policy for all beliefs (i.e., $V^*(b) \geq V^\pi(b) \forall b$).

As with policies, representing a value function can be problematic because its domain is an $(|\mathcal{S}| - 1)$-dimensional continuous space corresponding to the belief space. However, Smallwood and Sondik (1973) showed that optimal value functions for finite-horizon POMDPs are piecewise-linear and convex. The value of executing a conditional plan from any state is constant. If we do not know the precise underlying state, but instead we have a belief corresponding to a distribution over states, then the value of the belief is simply a weighted average (according to b) of the values of the possible states. Thus, the value function $V^\beta(b)$ of a conditional plan β is linear with respect to b. This means that $V^\beta(b)$ can be represented by a vector α_β of size $|\mathcal{S}|$ such that $V^\beta(b) = \Sigma_s b(s)\alpha_\beta(s)$.

For a finite horizon h, an optimal policy π^h consists of the best conditional plans for each initial belief. More precisely, the best conditional plan β^* for some belief b is the one that yields the highest value: $\beta^* = \text{argmax}_\beta V^\beta(b)$. Although there are uncountably many beliefs, the set of h-step conditional plans is finite and therefore an h-step optimal value function can be represented by a finite collection Γ^h of α-vectors. For infinite horizon problems, the optimal value function may require an infinite number of α-vectors.

Figure 4 shows an optimal value function for a simple two-state POMDP. The horizontal axis represents the belief space and the vertical axis indicates the expected total reward. Assuming the two world states are s and $\bar{s}$, then a belief is completely determined by the probability of s. Therefore, the horizontal axis represents a continuum of beliefs determined by the probability



Partially Observable Markov Decision Processes, Fig. 4 Geometric view of value function

$b(s)$. Each line in the graph is an α-vector, which corresponds to the value function of a conditional plan. The upper surface of those α-vectors is a piecewise-linear and convex function corresponding to the optimal value function $V^* = \max_{\alpha \in \Gamma^h} \alpha(b)$.

Note that an optimal policy can be recovered from the optimal value function represented by a set Γ of α-vector. Assuming that an action is stored with each α-vector (this would typically be the root action of the conditional plan associated with each α-vector), then the decision maker simply needs to look up the maximal α-vector for the current belief to retrieve the action. Hence, value functions represented by a set of α-vectors, each associated with an action, implicitly define a mapping from beliefs to actions.

Optimal value functions also satisfy *Bellman's equation*

$$V^{h+1}(b) = \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^h(b^{ao'}) \quad (3)$$

where $R(b,a) = \sum_s b(s) R(s,a)$, $\Pr(s'|s,a)$ $\Pr(o'|s',a)$, and $b^{ao'}$ is the updated belief after executing a and observing b according to Bayes theorem (Eq. 1). Intuitively, this equation says that the optimal value for $h+1$ steps to go consists of the highest sum of the current reward with the future rewards for the remaining h steps. Since we do not know exactly what rewards will be earned in the future, an expectation (with respect to the observations) is used to estimate future rewards. For discounted infinite horizon problems, the optimal value function V^* is a fixed point of Bellman's equation:

$$V^*(b) = \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a) V^*(b^{ao'})$$

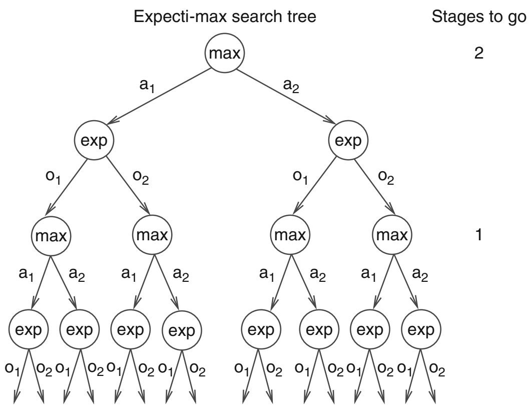
Solution Algorithms

There are two general classes of solution algorithms to optimize a policy. The first class consists of *online* algorithms that plan while executing the policy by growing a search tree. The second class consists of *offline* algorithms that precompute a policy which can be executed with minimal online computation. In practice, it is best to combine online and offline techniques since we may as well obtain the best policy possible in an offline phase and then refine it with an online search at execution time.

Forward Search

Online search techniques generally optimize a conditional plan for the current belief by performing a forward search from that belief. They essentially build an *expecti-max* search tree such that expectations over observations and maximizations over actions are performed in alternation. Figure 5 illustrates such a tree for a two-step horizon (i.e., two alternations of actions and observations). An optimal policy is obtained by computing the beliefs associated with each node in a forward pass, followed by a backward pass that computes the optimal value at each node. A recursive form of this approach is described in Algorithm 1. Beliefs are propagated forward according to Bayes theorem, while rewards are accumulated backward according to Bellman's equation.

Since the expecti-max search tree grows exponentially with the planning horizon h, in practice, the computation can often be simplified by pruning suboptimal actions by branch and bound and sampling a small set of observations instead of doing an exact expectation (Ross et al. 2008). Also, the depth of the search can be reduced by using an approximate value function at the leaves instead of 0.



Partially Observable Markov Decision Processes, Fig. 5 Two-step expecti-max search tree

Algorithm 1 Forward search

```
Inputs: Belief $b$ and horizon $h$
Outputs: Optimal value $V^*$.
if $h = 0$ then
  $V^* \leftarrow 0$
else
  for all $a, o$ do
    $b^{ao'}(s') \leftarrow k \sum_s b(s)\Pr(s'|s,a)\Pr(o'|s',a')\forall s'$
    $V^{ao'} \leftarrow forward\ Search(b^{ao'}, h-1)$
  end for
  $V^* \leftarrow \max_a R(b,a) + \gamma \sum_{o'} \Pr(o'|b,a)V^{ao'}$
end if
```

The value functions computed by offline techniques can often be used for this purpose.

Value Iteration

Value iteration algorithms form an important class of offline algorithms that iteratively estimate the optimal value function according to Bellman's equation (3). Most algorithms exploit the piecewise-linear and convex properties of optimal value functions to obtain a finite representation. In other words, optimal value functions Vh are represented by a set Γ^h of α-vectors that correspond to conditional plans. Algorithm 2 shows how to iteratively compute Γ^t by dynamic programming for an increasing number of time steps t.

Algorithm 2 Value iteration

```
Inputs: horizon $h$
Outputs: Optimal value function $\Gamma^h$.
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to $h$ do
  for all $a \in \mathcal{A}, <\alpha_1, \ldots, \alpha_{|\mathcal{O}|}> \in (\Gamma^{t-1})^{|\mathcal{O}|}$ do
    $\alpha'(s) \leftarrow R(s,a) +$
    $\gamma \sum_{o',s'} \Pr(s'|s,a)\Pr(o'|s',a)\alpha_{o'}(s')\forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
    end for
  end for
```

Algorithm 3 Point based value iteration

```
Inputs: Horizon $h$ and set of beliefs $\mathcal{B}$
Outputs: Value function $\Gamma^h$.
$\Gamma^0 \leftarrow \{0\}$
for $t = 1$ to $h$ do
  for all $b \in \mathcal{B}$ do
    for all $a \in \mathcal{A}, o' \in \mathcal{O}$ do
      $b^{ao'}(s') \leftarrow k \sum_s b(s)\Pr(s'|s,a)\Pr(o'|s',a)\forall s'$
      $\alpha^{ao'} \leftarrow \text{argmax}_{alpha \in \Gamma^{t-1}} \alpha(b^{ao'})$
    end for
    $a^* \leftarrow \text{argmax}_a R(b,a) + \gamma \sum_{o'} \Pr(s'|s,a)\Pr\alpha^{ao'}$
    $\alpha'(s) R(s,a) + \gamma \sum_{o',s'} \Pr(s'|s,a)\Pr(o'|s',a)$
    $\alpha_{o'}(s')\forall s$
    $\Gamma^t \leftarrow \Gamma^t \cup \{\alpha'\}$
  end for
end for
```

Unfortunately, the number of α-vectors in each Γ^t increases exponentially with $\mathcal{O}$ and doubly exponentially with t. While several approaches can be used to prune α-vectors that are not maximal for any belief, the number of α-vectors still grows exponentially for most problems. Instead, many approaches compute a parsimonious set of α-vectors, which defines a lower bound on the optimal value function. The class of *point-based value iteration* (Pineau et al. 2006) algorithms computes the maximal α-vectors only for a set $\mathcal{B}$ of beliefs. Algorithm 3 describes how the parsimonious set Γ^h of α-vectors associated with a given set $\mathcal{B}$ of beliefs can be computed in time linear with h and $|\mathcal{O}|$ by dynamic programming. Most point-based techniques differ in how they choose $\mathcal{B}$ (which may vary at each iteration), but the general rule of thumb is to include beliefs reachable from the initial belief b_0 since these are the beliefs that are likely to be encountered at execution time.

Policy Search

Another important class of offline algorithms consists of policy search techniques. These techniques search for the best policy in a predefined space of policies. For instance, finite state controllers are a popular policy space due to their generality and simplicity. The search for the best (stochastic) controller of N nodes can be formulated as a non-convex quadratically constrained optimization problem (Amato et al. 2007):

$$\max_{x,y,z} \sum_s b_0(s) \underbrace{\alpha_0(s)}_{x}$$

$$\text{s.t. } \underbrace{\alpha_n(s)}_{x} = \sum_a [\underbrace{\Pr(a|n)}_{y} R(S,a)$$

$$+ \gamma \sum_{s',0',n'} \Pr(s'|s,a)$$

$$\Pr(0'|s',a) \underbrace{\Pr(a,n'|n,0')}_{z} \underbrace{\alpha_{n'}(s')}_{x}] \; \forall s,n$$

$$\underbrace{\Pr(a,n'|n,0')}_{x} \geq 0 \forall a,n',n,0'$$

$$\sum_{n'a} \underbrace{\Pr(a,n'|n,0)}_{z} = 1 \forall n,0$$

$$\sum_{n'} \underbrace{\Pr(a,n'|n,0')}_{z} = \underbrace{\Pr(a|n)}_{y} \; \forall a,n,0'$$

The variables of the optimization problem are the α-vectors and the parameters of the controller ($\Pr(a|n)$ and $\Pr(a,\ n'|n,o')$). Here, $\Pr(a|n)$ is the action distribution for each node n and $\Pr(a,\ n'|n,o') = \Pr(a|n)\Pr(n'|a,n,o')$ is the product of the action distribution and successor node distribution for each n, o'-pair. While there does not exist any algorithm that reliably finds the global optimum due to the non-convex nature of the problem, several techniques can be used to find locally optimal policies, including sequential quadratic programming, bounded policy iteration, expectation maximization, stochastic local search, and gradient descent.

Related Work

Although this entry assumes that states, actions, and observations are defined by a single variable, multiple variables can be used to obtain a *factored POMDP* (Boutilier and Poole 1996). As a result, the state, observation, and action spaces often become exponentially large. Aggregation (Shani et al. 2008; Sim et al. 2008) and compression techniques (Poupart and Boutilier 2004; Roy et al. 2005) are then used to speed up computation. POMDPs can also be defined for problems with continuous variables. The piecewise-linear and convex properties of optimal value functions still hold in continuous spaces, which allows value iteration algorithms to be easily extended to continuous POMDPs (Porta et al. 2006). When a planning problem can naturally be thought as a hierarchy of subtasks, *hierarchical POMDPs* (Theocharous and Mahadevan 2002; Pineau et al. 2003; Toussaint et al. 2008) can be used to exploit this structure.

In this article, we also assumed that the transition, observation, and reward functions are known, but in many domains they may be (partially) unknown and therefore the decision maker needs to learn about them while acting. This is a problem of *reinforcement learning*. While several policy search techniques have been adapted to simultaneously learn and act (Meuleau et al. 1999; Aberdeen and Baxter 2002), it turns out that one can treat the unknown parameters of the transition, observation, and reward functions as hidden state variables, which lead to a *Bayes-adaptive POMDP* (Ross et al. 2007; Poupart and Vlassis 2008). We also assumed a single decision maker, however POMDPs have been extended for multiagent systems. In particular, *decentralized POMDPs* (Amato et al. 2009) can model multiple cooperative agents that share a common goal and *interactive POMDPs* (Gmytrasiewicz and Doshi 2005) can model multiple competing agents.

Cross-References

▸ Markov Decision Processes

Recommended Reading

Aberdeen D, Baxter J (2002) Scalable internal-state policygradient methods for POMDPs. In: International conference on machine learning, Sydney, pp 3–10

Amato C, Bernstein DS, Zilberstein S (2009) Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. J Auton Agents Multiagent Syst 21:293–320

Amato C, Bernstein DS, Zilberstein S (2007) Solving POMDPs using quadratically constrained linear programs. In: International joint conferences on artificial intelligence, Hyderabad, pp 2418–2424

Aström KJ (1965) Optimal control of Markov decision processes with incomplete state estimation. J Math Anal Appl 10:174–2005

Boutilier C, Poole D (1996) Computing optimal policies for partially observable decision processes using compact representations. In: Proceedings of the thirteenth national conference on artificial intelligence, Portland, pp 1168–1175

Buede DM (1999) Dynamic decision networks: an approach for solving the dual control problem. Spring INFORMS, Cincinnati

Drake A (1962) Observation of a Markov Process through a noisy channel. PhD thesis, Massachusetts Institute of Technology

Hansen E (1997) An improved policy iteration algorithm for partially observable MDPs. In: Neural information processing systems, Denver, pp 1015–1021

Hauskrecht M, Fraser HSF (2010) Planning treatment of ischemic heart disease with partially observable Markov decision processes. Artif Intell Med 18:221–244

Hoey J, Poupart P, von Bertoldi A, Craig T, Boutilier C, Mihailidis A (2010) Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process. Comput Vis Image Underst 114:503–519

Kaelbling LP, Littman M, Cassandra A (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101:99–134

Meuleau N, Peshkin L, Kim K-E, Kaelbling LP (1999) Learning finite-state controllers for partially observable environments. In: Uncertainty in artificial intelligence, Stockholm, pp 427–436

Pineau J, Gordon G (2005) POMDP planning for robust robot control. In: International symposium on robotics research, San Francisco, pp 69–82

Pineau J, Gordon GJ, Thrun S (2003) Policy-contingent abstraction for robust robot control. In: Uncertainty in artificial intelligence, Acapulco, pp 477–484

Pineau J, Gordon G, Thrun S (2006) Anytime point-based approximations for large POMDPs. J Artif Intell Res 27:335–380

Gmytrasiewicz PJ, Doshi P (2005) A framework for sequential planning in multi-agent settings. J Artif Intell Res 24:49–79

Porta JM, Vlassis NA, Spaan MTJ, Poupart P (2006) Point-based value iteration for continuous POMDPs. J Mach Learn Res 7:2329–2367

Poupart P, Boutilier C (2004) VDCBPI: an approximate scalable algorithm for large POMDPs. In: Neural information processing systems, Vancouver, pp 1081–1088

Poupart P, Vlassis N (2008) Model-based Bayesian reinforcement learning in partially observable domains. In: International symposium on artificial intelligence and mathematics (ISAIM), Fort Lauderdale

Puterman ML (1994) Markov decision processes. Wiley, New York

Rabiner LR (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77:257–286

Ross S, Chaib-Draa B, Pineau J (2007) Bayes-adaptive POMDPs. In: Advances in neural information processing systems (NIPS), Vancouver

Ross S, Pineau J, Paquet S, Chaib-draa B (2008) Online planning algorithms for POMDPs. J Artif Intell Res 32:663–704

Roy N, Gordon GJ, Thrun S (2005) Finding approximate POMDP solutions through belief compression. J Artif Intell Res 23:1–40

Shani G, Meek C (2009) Improving existing fault recovery policies. In: Neural information processing systems, Vancouver

Shani G, Brafman RI, Shimony SE, Poupart P (2008) Efficient ADD operations for point-based algorithms. In: International conference on automated planning and scheduling, Sydney, pp 330–337

Sim HS, Kim K-E, Kim JH, Chang D-S, Koo M-W (2008) Symbolic heuristic search value iteration for factored POMDPs. In: Twenty-third national conference on artificial intelligence (AAAI), Chicago, pp 1088–1093

Smallwood RD, Sondik EJ (1973) The optimal control of partially observable Markov decision processes over a finite horizon. Oper Res 21:1071–1088

Theocharous G, Mahadevan S (2002) Approximate planning with hierarchical partially observable Markov decision process models for robot navigation. In: IEEE international conference on robotics and automation, Washington, DC, pp 1347–1352

Thomson B, Young S (2010) Bayesian update of dialogue state: a POMDP framework for spoken dialogue systems. Comput Speech Lang 24:562–588

Toussaint M, Charlin L, Poupart P (2008) Hierarchical POMDP controller optimization by likelihood maximization. In: Uncertainty in artificial intelligence, Helsinki, pp 562–570

Particle Swarm Optimization

James Kennedy
U.S. Bureau of Labor Statistics, Washington, DC, USA

The Canonical Particle Swarm

The particle swarm is a population-based stochastic algorithm for optimization which is based on social–psychological principles. Unlike ▸ evolutionary algorithms, the particle swarm does not use selection; typically, all population members survive from the beginning of a trial until the end. Their interactions result in iterative improvement of the quality of problem solutions over time.

A numerical vector of D dimensions, usually randomly initialized in a search space, is conceptualized as a point in a high-dimensional Cartesian coordinate system. Because it moves around the space testing new parameter values, the point is well described as a particle. Because a number of them (usually $10 < N < 100$) perform this behavior simultaneously, and because they tend to cluster together in optimal regions of the search space, they are referred to as a *particle swarm*.

Besides moving in a (usually) Euclidean problem space, particles are typically enmeshed in a topological network that defines their communication pattern. Each particle is assigned a number of neighbors to which it is linked bidirectionally.

The most common type of implementation defines the particles' behaviors in two formulas. The first adjusts the velocity or step size of the particle, and the second moves the particle by adding the velocity to its previous position.

On each dimension d:

$$v_{id}^{(t+1)} \leftarrow \alpha v_{id}^{(t)} + U(0,\beta)\left(p_{id} - x_{id}^{(t)}\right) + U(0,\beta)\left(p_{gd} - x_{id}^{(t)}\right) \qquad (1)$$

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + v_{id}^{(t+1)} \qquad (2)$$

where i is the target particle's index, d is the dimension, $\vec{x}_i$ is the particle's position, $\vec{v}_i$ is the velocity, $\vec{p}_i$ is the best position found so far by i, g is the index of i's best neighbor, α and β are constants, and $U(0, \beta)$ is a uniform random number generator.

Though there is variety in the implementations of the particle swarm, the most standard version uses $\alpha = 0.7298$ and $\beta = \psi/2$, where $\psi = 2.9922$, following an analysis published in Clerc and Kennedy (2002). The constant α is called an *inertia weight* or *constriction coefficient*, and β is known as the *acceleration constant*.

The program evaluates the parameter vector of particle i in a function $f(\vec{x})$ and compares the result to the best result attained by i thus far, called $pbest_i$. If the current result is i's best so far, the vector $\vec{p}_i$ is updated with the current position $\vec{x}_i$, and the previous best function result $pbest_i$ is updated with the current result.

When the system is run, each particle cycles around a region centered on the centroid of the previous bests $\vec{p}_i$ and $\vec{p}_g$; as these variables are updated, the particle's trajectory shifts to new regions of the search space, the particles begin to cluster around optima, and improved function results are obtained.

The Social–Psychological Metaphor

Classical social psychology theorists considered the pursuit of *cognitive consistency* to be an important motivation for human behavior (Heider 1958; Festinger 1957; Abelson et al. 1968). Cognitive elements might have emotional or logical aspects to them which could be consistent or inconsistent with one another; several theorists identified frameworks for describing the degree of consistency and described the kinds of processes that an individual might use to increase consistency or balance, or decrease inconsistency or cognitive dissonance.

Contemporary social and cognitive psychologists frequently cast these same concepts in terms of connectionist principles. Cognitive elements are conceptualized as a network with positive and negative vertices among a set of nodes. In some models, the elements are given and the task is to reduce error by adjusting the signs and values of the connections between them, and in other models the connections are given and the goal of optimization is to find activation values that maximize coherence (Thagard 2000), harmony (Smolensky 1986), or some other measure of consistency. Typically, this optimization is performed by gradient-descent programs which psychologically model processes that are private to the individual and are perfectly rational, that is, the individual always decreases error or increases consistency among elements. The particle swarm simulates the optimization of these kinds of structures through social interaction; it is commonly observed, not only in the laboratory but in everyday life, that a person faced with a problem typically solves it by talking with other people.

A direct precursor of the particle swarm is seen in Nowak et al. (1990) cellular automaton simulation of social impact theory's predictions about interaction in human social populations. Social impact theory predicted that an individual was influenced to hold an attitude or belief in proportion to the Strength, Immediacy, and Number of sources of influence holding that position, where Strength was a measure of the persuasiveness or prestige of an individual, Immediacy was their proximity, and Number was literally the number of sources of influence holding a particular attitude or belief. In the simulation, individuals iteratively interacted, taking on the prevalent state of a binary attitude in their neighborhood, until the system reached equilibrium.

The particle swarm extends this model by supposing that various states can be evaluated, for instance, that different patterns of cognitive elements may be more or less dissonant; it assumes that individuals hold more than one attitude or belief, and that they are not necessarily binary; and Strength is replaced with a measure of self-presented success. One feature usually found in particle swarms and not in the paper by Nowak et al. is the phenomenon of persistence or momentum, the tendency of an individual to keep changing or moving in the same direction from one time-step to the next.

Thus, the particle swarm metaphorically represents the interactions of a number of individuals, none knowing what the goal is, each knowing its immediate state and its best performance in the past, each presenting its neighbors with its best success-so-far at solving a problem, each functioning as both source and target of influence in the dynamically evolving system. As individuals emulate the successes of their neighbors, the population begins to cluster in optimal regions of a search space, reliably discovering good solutions to difficult problems featuring, for instance, nonlinearity, high dimension, deceptive gradients, local optima, etc.

The Population Topology

Several kinds of topologies have been most widely used in particle swarm research; the topic is a current focus of much research. In the *gbest* topology, the population is conceptually fully connected; every particle is linked to every other. In practice, with the best neighbor canonical version, this is simpler to implement than it sounds, as it only means that every particle receives influence from the best performing member of the population.

The *lbest* topology of degree K_i comprises a ring lattice, with the particle linked to its K_i nearest neighbors on both sides in the wrapped population array.

Another structure commonly used in particle swarm research is the von Neumann or "square" topology. In this arrangement, the population is laid out in rows and columns, and each individual is connected to the neighbors above, below, and on each side of it in the toroidally wrapped population. Numerous other topologies have been used, including random (Suganthan 1999), hierarchical (Janson and Middendorf 2005), and adaptive ones (Clerc 2006).

The most important effect of the population topology is to control the spread of proposed problem solutions through the population. As a particle finds a good region of the search space, it may become the best neighbor to one of the particles it is connected to. That particle then will tend to explore in the vicinity of the first particle's success, and may eventually find a good solution there, too; it could then become the best neighbor to one of its other neighbors. In this way, information about good regions of the search space migrates through the population.

When connections are parallel, e.g., when the mean degree of particles is relatively high, then information can spread quickly through the population. On unimodal problems this may be acceptable, but where there are local optima there may be a tendency for the population to converge too soon on a suboptimal solution. The *gbest* topology has repeatedly been shown to be vulnerable to the lure of locally optimal attractors.

On the other hand, where the topology is sparse, as in the *lbest* model, problem solutions spread slowly, and subpopulations may search diverse regions of the search space in parallel. This increases the probability that the population will end up near the global optimum. It also means that convergence will be slower.

Vmax and Convergence

The particle swarm has evolved very much since it was first reported by Kennedy and Eberhart (1995) and Eberhart and Kennedy (1995). Early versions required a system constant *Vmax* to limit the velocity. Without this limit, the particles' trajectories would swing wildly out of control.

Following presentation of graphical representations of a deterministic form of the particle swarm by Kennedy (1998), early analyses by Ozcan and Mohan (1999) led to some understanding of the nature of the particle's trajectory. Analytical breakthroughs by Clerc (reported in Clerc and Kennedy (2002)), and empirical discoveries by Shi and Eberhart (1998), resulted in the application of the α constant in concert with appropriate values of the acceleration constant β. These parameters brought the particle under control, allowed convergence under appropriate conditions, and made *Vmax* unnecessary. It is still used sometimes, set to very liberal values such as a half or third of the initialization range of a variable for more efficient swarm behavior, but it is not necessary.

Step Size and Consensus

Step size in the particle swarm is inherently scaled to consensus among the particles. A particle goes in one direction on each dimension until the sign of its velocity is reversed by the accumulation of $(p - x)$ differences; then it turns around and goes the other way. As it searches back and forth, its oscillation on each dimension is centered on the mean of the previous bests $(p_{id} + p_{gd})/2$, and the standard deviation of the distribution of points that are tested is scaled to the difference between them. In fact this function is a very simple one: the standard deviation of a particle's search, when p_{id} and p_{gd} are constants, is approximately $|(p_{id} - p_{gd})|$. This means that when the particles' previous best points are far from one another in the search space, the particles will take big steps, and when they are nearer the particles will take little steps.

Over time, this usually means that exploring behavior is seen in early iterations and exploiting behavior later on as particles come to a state of consensus. If it happens, however, that a particle that has begun to converge in one part of the search space receives information about a good region somewhere else, it can return to the exploratory mode of behaving.

The Fully Informed Particle Swarm (FIPS)

Mendes (2004) reported a version of swarm that featured an alternative to the best neighbor strategy. While the canonical particle is influenced by its own previous success and the previous success of its best neighbor, the fully informed particle swarm (FIPS) allowed influence by all of a particle's neighbors. The acceleration constants were set to $\beta = \psi/\ 2$ in the traditional version; it was defined in this way because what mattered was their sum, which could be distributed among any number of difference terms. In the standard algorithm there were two of them, and thus the sum was divided by 2. In FIPS a particle of K_i degree has coefficients $\beta = \psi/K_i$.

The FIPS particle swarm removed two aspects that were considered standard features of the algorithm. First of all, the particle i no longer influenced itself directly, e.g., there is no $\vec{p}_i$ in the formula. Second, the best neighbor is now averaged in with the others; it was not necessary to compare the successes of all neighbors to find the best one.

Mendes found that the FIPS swarm was more sensitive than the canonical versions to the differences in topology. For instance, while in the standard versions the fully connected *gbest* topology meant influence by the best solution known to the entire population, in FIPS *gbest* meant that the particle was influenced by a stochastic average of the best solutions found by all members of the population; the result tended to be near-random search.

The lesson to be learned is that the *meaning* of a topology depends on the mode of interaction. Topological structure (and Mendes tested more than 1,340 of them) affects performance, but the way it affects the swarm's performance depends on how information is propagated from one particle to another.

Generalizing the Notation

Equation 2 above shows that the position is derived from the previous iteration's position plus the current iteration's velocity. By rearranging the terms, it can be shown that the current iteration's velocity $\vec{v}_i^{(t+1)}$ is the difference between the new position and the previous one: $\vec{v}_i^{(t+1)} = \vec{x}_i^{(t+1)} - \vec{x}_i^{(t)}$. Since this happened on the previous time-step as well, it can be shown that $\vec{v}_i^{(t)} = \vec{x}_i^{(t)} - \vec{x}_i^{(t-1)}$; this fact makes it possible to combine the two formulas into one:

$$x_{id}^{(t+1)} \leftarrow x_{id}^{(t)} + \alpha \left(x_{id}^{(t)} - x_{id}^{(t-1)} \right) + \sum U \left(0, \frac{\Psi}{K_i} \right) \left(p_{kd} - x_{id}^{(t)} \right) \qquad (3)$$

where K_i is the degree of node i, k is the index of i's kth neighbor, and adapting Clerc's (Clerc and Kennedy 2002) scheme $\alpha = 0.7298$ and $\psi = 2.9922$.

In the canonical best neighbor particle swarm, $K_i = 2, \forall i : i = 1, 2, \ldots, N$ and $k \in (i, g)$, that is, k takes the values of the particle's own index and its best neighbor's index. In FIPS, K_i may vary, depending on the topology, and k takes on the indexes of each of i's neighbors. Thus, Eq.3 is a generalized formula for the trajectories of the particles in the particle swarm.

This notation can be interpreted verbally as:

$$\begin{aligned} &NEW\ POSITION \\ &= CURRENT\ POSITION \\ &+ PERSISTENCE \\ &+ SOCIAL INFLUENCE \end{aligned} \qquad (4)$$

That is, on every iteration, every particle on every dimension starts at the point it last arrived at, persists some weighted amount in the direction it was previously going, then makes some adjustments based on the differences between the best previous positions of its sources of influence and its own current position in the search space.

The Evolving Paradigm

The particle swarm paradigm is young, and investigators are still devising new ways to understand, explain, and improve the method. A divergence or bifurcation of approaches is observed: some researchers seek ways to simplify the algorithm (Peña et al. 2006; Owen and Harvey 2007), to find its essence, while others improve performance by adding features to it, e.g., Clerc (2006). The result is a rich unfolding research tradition with innovations appearing on many fronts.

Although the entire algorithm is summarized in one simple formula, it is difficult to understand how it operates or why it works. For instance, while the *Social Influence* terms point the particle in the direction of the mean of the influencers' successes, the *Persistence* term offsets that movement, causing the particle to bypass what seems to be a reasonable target. The result is a spiral-like trajectory that goes past the target and returns to pass it again, with the spiral tightening as the neighbors come to consensus on the location of the optimum.

Further, while authors often talk about the particle's velocity carrying it "toward the previous bests," in fact the velocity counterintuitively carries it *away from* the previous bests as often as toward them. It is more accurate to say the particle "explores around" the previous bests, and it is hard to describe this against-the-grain movement as "gradient descent," as some writers would like.

It is very difficult to visualize the effect of ever-changing sources of influence on a particle. A different neighbor may be best from one iteration to the next; the balance of the random numbers may favor one or another or some compromise of sources; the best neighbor could remain the same one, but may have found a better $\vec{p}_i$ since the last turn; and so on. The result is that the particle is pulled and pushed around in a complex way, with many details changing over time.

The paradoxical finding is that it is best not to give the particle information that is too good, especially early in the search trial. Premature convergence is the result of amplified consensus resulting from too much communication or overreliance on best neighbors, especially the population best. Various researchers have proposed ways to slow the convergence or clustering of particles in the search space, such as occasional reinitialization or randomization of particles, repelling forces among them, etc., and these techniques typically have the desired effect. In many cases, however, implicit methods work as well and more parsimoniously; the effect of topology on convergence rate has been mentioned here, for instance.

Binary Particle Swarms

A binary particle swarm is easily created by treating the velocity as a probability threshold (Kennedy and Eberhart 1997). Velocity vector elements are squashed in a sigmoid or other function, for instance $S(\upsilon) = 1/(1 + exp(-\upsilon))$, producing a result in (0..1). A random number is generated and compared to $S(\upsilon_{id})$ to determine whether x_{id} will be a 0 or a 1. Though discrete systems of higher cardinality have been proposed, it is difficult to define such concepts as distance and direction in a meaningful way within nominal data.

Alternative Probability Distributions

As was noted above, the particle's search is centered around the mean of the previous bests that influence it, and its variance is scaled to the differences among them. This has suggested to several researchers that perhaps the trajectory formula can be replaced, wholly or partly, by some type of random number generator that directly samples the search space in a desirable way.

Kennedy (2003) suggested simple Gaussian sampling, using a random number generator (RNG) $G(mean, s.d.)$ with the mean centered between $\vec{p}_i$ and $\vec{p}_g$, and with the standard deviation defined on each dimension as $s.d. = |(p_{id} - p_{gd})|$. This "bare bones" particle swarm eliminated the velocity component; it performed rather well on a set of test functions, but not as well as the usual version.

Krohling (2004) simply substituted the absolute values of Gaussian-distributed random numbers for the uniformly distributed values in the canonical particle swarm. He and his colleagues have had success on a range of problems using this approach. Richer and Blackwell (2006) replaced the Gaussian distribution of bare bones with a Lévy distribution. The Lévy distribution is bell-shaped like the Gaussian but with fatter tails. It has a parameter α which allows interpolation between the Cauchy distribution ($\alpha = 1$) and Gaussian ($\alpha = 2$) and can be used to control the fatness of the tails. In a series of trials, Richer and Blackwell (2006) were able to emulate the performance of a canonical particle swarm using $\alpha = 1.4$. Kennedy (2005) used a Gaussian RNG for the social influence term of the usual formula, keeping the "persistence" term found in the standard particle swarm. Variations on this format produced results that were competitive with the canonical version.

Numerous other researchers have begun exploring ways to replicate the overall behavior of the particle swarm by replacing the traditional formulas with alternative probability distributions. Such experiments help theorists understand what is essential to the swarm's behavior and how it is able to improve its performance on a test function over time.

Simulation of the canonical trajectory behavior with RNGs is a topic that is receiving a great deal of attention at this time, and it is impossible to predict where the research is leading. As numerous versions have been published showing that the trajectory formulas can be replaced by alternative strategies for selecting a series of points to sample, it becomes apparent that the essence of the paradigm is not to be found in the details of the movements of the particles, but in the nature of their interactions over time, the structure of the social network in which they are embedded, and the function landscape with which they interact, with all these factors working together gives the population the ability to find problem solutions.

Recommended Reading

Abelson RP, Aronson E, McGuire WJ, Newcomb TM, Rosenberg MJ, Tannenbaum RH (eds) (1968) Theories of cognitive consistency: a sourcebook. Rand McNally, Chicago

Clerc M (2006) Particle swarm optimization. Hermes Science Publications, London

Clerc M, Kennedy J (2002) The particle swarm: explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6:58–73

Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the 6th international symposium on micro machine and human science, Nagoya. IEEE Service Center, Piscataway, pp 39–43

Festinger L (1957) A theory of cognitive dissonance. Stanford University Press, Stanford

Heider F (1958) The psychology of interpersonal relations. Wiley, New York

Janson S, Middendorf M (2005) A hierarchical particle swarm optimizer and its adaptive variant. IEEE Trans Syst Man Cybern Part B Cybern 35(6):1272–1282

Kennedy J (1998) The behavior of particles. In: Porto VW, Saravanan N, Waagen D, Eiben AE (eds) Evolutionary programming VII. Proceedings of the 7th annual conference on evolutionary programming, San Diego

Kennedy J (2003) Bare bones particle swarms. In: Proceedings of the IEEE swarm intelligence symposium, Indianapolis, pp 80–87

Kennedy J (2005) Dynamic-probabilistic particle swarms. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2005), Washington, DC, pp 201–207

Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, Perth. IEEE Service Center, Piscataway, pp 1942–1948

Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the 1997 conference on systems, man, and cybernetics. IEEE Service Center, Piscataway, pp 4104–4109

Krohling RA (2004) Gaussian Swarm. A novel particle swarm optimization algorithm. Proc 2004 IEEE Conf Cybern Intell Syst 1:372–376

Mendes R (2004) Population topologies and their influence in particle swarm performance. Doctoral thesis, Escola de Engenharia, Universidade do Minho

Nowak A, Szamrej J, Latané B (1990) From private attitude to public opinion: a dynamic theory of social impact. Psychol Rev 97:362–376

Owen A, Harvey I (2007) Adapting particle swarm optimisation for fitness landscapes with neutrality. In: Proceedings of the 2007 IEEE Swarm intelligence symposium. IEEE Press, Honolulu, pp 258–265

Ozcan E, Mohan CK (1999) Particle swarm optimization: surfing the waves. In: Proceedings of the congress on evolutionary computation, Mayflower hotel, Washington, DC. IEEE Service Center, Piscataway, pp 1939–1944

Peña J, Upegui A, Eduardo Sanchez E (2006) Particle Swarm optimization with discrete recombination: an online optimizer for evolvable hardware. In: Proceedings of the 1st NASA/ESA conference on adaptive hardware and systems (AHS-2006), Istanbul. IEEE Service Center, Piscataway, pp 163–170

Richer TJ, Blackwell TM (2006) The Levy particle Swarm. In: Proceedings of the 2006 congress on evolutionary computation (CEC-2006). IEEE Service Center, Piscataway

Shi Y, Eberhart RC (1998) Parameter selection in particle Swarm optimization. In: Evolutionary programming VII: proceedings EP98. Springer, New York, pp 591–600

Smolensky P (1986) Information processing in dynamical systems: foundations of harmony theory. In: Rumelhart DE, McClelland JL, the PDP Research Group (eds) Parallel distributed processing: explorations in the microstructure of cognition, vol 1, Foundations. MIT Press, Cambridge, pp 194–281

Suganthan PN (1999) Particle Swarm optimisation with a neighbourhood operator. In: Proceedings of congress on evolutionary computation, Washington DC

Thagard P (2000) Coherence in thought and action. MIT Press, Cambridge

Partitional Clustering

Xin Jin[1] and Jiawei Han[2]
[1]PayPal Inc., San Jose, CA, USA
[2]University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

Partitional clustering is a type of clustering algorithms that divide a set of data points into disjoint subsets. Each data point is in exactly one subset.

Synonyms

Objective function

Definition

Partitional clustering (Han et al. 2011) decomposes a data set into a set of disjoint clusters. Given a data set of N points, a partitioning method constructs K ($N \geq K$) partitions of the data with each partition representing a cluster. That is, it classifies the data into K groups by satisfying the following requirements: (1) each group contains at least one point, and (2) each point belongs to exactly one group. For fuzzy partitioning, a point can belong to more than one group. The quality of the solution is measured by clustering criteria.

Some partitional clustering algorithms work by minimizing an objective function. For example, in K-means and K-medoids, the function (also referred as the distortion function) is

$$\sum_{i=1}^{K} \sum_{j=1}^{|C_i|} Dist(x_j, center(i)) \qquad (1)$$

where $|C_i|$ is the number of points in cluster i and $Dist(x_j, center(i))$ is the distance between point x_j and center i. Depending on the need of the applications, different distance functions can be used, such as Euclidean distance and L_1 norm.

Major Algorithms

Many algorithms can be used to perform partitional data clustering; representative technologies include K-means (Lloyd 1957), K-medoids (Kaufman and Rousseeuw 2005), quality threshold (QT) (Heyer et al. 1999), expectation-maximization (EM) (Dempster et al. 1977), mean shift (Comaniciu and Meer 2002), locality-sensitive hashing (LSH) (Gionis et al. 1999), K-way spectral clustering (Luxburg 2007), etc. In the K-means algorithm, each cluster is represented by the mean value of the points in the cluster. For the K-medoids algorithm, each cluster is represented by one of the points located near the center of the cluster. Instead of setting the cluster number K, the QT algorithm uses the maximum cluster diameter as a parameter to find clusters with guaranteed quality. Expectation-maximization clustering performs expectation-maximization analysis based on statistical modeling of the data distribution, and it has more parameters. Mean shift is a nonparameter algorithm to find any shape of clusters using density estimator. Locality-sensitive hashing-based method performs clustering by hashing similar points to the same bin. K-way spectral clustering algorithm represents the data as a graph and performs graph partitioning to find clusters.

Cross-References

- ▸ K-Means Clustering
- ▸ K-Medoids Clustering
- ▸ K-Way Spectral Clustering
- ▸ Quality Threshold Clustering

Recommended Reading

Comaniciu D, Meer P (2002) Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell 24(5):603–619

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Soc Ser B (Methodol) 39(1):1–38

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: proceedings of the 25th international conference on very large data

bases (VLDB'99), San Francisco. Morgan Kaufmann Publishers Inc, pp 518–529
Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann, San Francisco
Heyer L, Kruglyak S, Yooseph S (1999) Exploring expression data: identification and analysis of coexpressed genes. Genome Res 9:1106–1115
Kaufman L, Rousseeuw PJ (2005) Finding groups in data: an introduction to cluster analysis. Wiley series in probability and statistics. Wiley-Interscience, Hoboken
Lloyd SP (1957) Least squares quantization in PCM. Technical report RR-5497, Bell Lab
Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

Passive Learning

A ▸ passive learning system plays no role in the selection of its ▸ training data. Passive learning stands in contrast to ▸ active learning.

PCA

▸ Principal Component Analysis

PCFG

▸ Probabilistic Context-Free Grammars

Phase Transitions in Machine Learning

Lorenza Saitta[1] and Michele Sebag[2]
[1]Università del Piemonte Orientale, Alessandria, Italy
[2]CNRS – INRIA – Université Paris-Sud, Orsay, France

Synonyms

Statistical physics of learning; Threshold phenomena in learning; Typical complexity of learning

Definition

Phase transition (PT) is a term originally used in physics to denote a sudden transformation of a system from one state to another, such as from liquid to solid or to gas state (phase). It is used, by extension, to describe any abrupt change in one of the *order* parameters describing an arbitrary system, when a *control* parameter approaches a *critical* value.

Far from being limited to physical systems, PTs are ubiquitous in sciences, notably including computational science. Typically, hard combinatorial problems display a PT with regard to the probability of existence of a solution. Note that the notion of PT cannot be studied in relation to single-problem instances: it refers to emergent phenomena in an *ensemble* of problem instances, governed by a given probability distribution.

Motivation and Background

Cheeseman et al. (1991) were most influential in starting the study of PTs in artificial intelligence, experimentally showing the presence of a PT containing the most difficult instances for various **NP**-complete problems. Since then, the literature flourished both in breadth and depth, witnessing an increasing transfer of knowledge and results between statistical physics and combinatorics.

As far as machine learning (ML) can be formulated as a combinatorial optimization problem (Mitchell 1982), it is no surprise that PTs emerge in many of its facets. Early results have been obtained in the field of relational learning, either logic (Botta et al. 2003; Giordana and Saitta 2000) or kernel (Gaudel et al. 2008) based. PTs have been studied in neural networks (Demongeot and Sené 2008; Engel and Van den Broeck 2001), grammatical inference (Cornuéjols and Sebag 2008), propositional classification (Baskiotis and Sebag 2004; Rückert and De Raedt 2008), and sparse regression (Donoho and Tanner 2005).

Two main streams of research emerge from the study of PT in computational problems. On the one hand, locating the PT enables very difficult problem instances to be generated, those which are most relevant to benchmarks and comparative assessment of new algorithms. On the other hand, PT studies stimulate the analysis of algorithmic *typical case complexity*, as opposed to the standard worst-case analysis of algorithmic complexity. It is well known that while many algorithms require exponential resources in the worst case, they are effective for a vast majority of problem instances. Studying their typical runtime thus makes sense in a probabilistic perspective. The typical runtime not only reflects the most probable runtime; overall, the probability of deviating from this typical complexity goes to zero as the problem size increases.

Relational Learning

In a seminal paper, Mitchell characterized ML as a search problem (Mitchell 1982). Much attention has ever since been devoted to every component of a search problem: the search space, the search goal, and the search engine.

The search space $\mathcal{H}$ reflects the language $\mathcal{L}$ chosen to express the target knowledge, termed ▸ *hypothesis language*. The reader is referred to other entries of the encyclopedia (▸ *Attribute-value* representation, ▸ *First-order logic*, ▸ *Relational learning*, and ▸ *Inductive Logic Programming*) for a comprehensive presentation of the hypothesis languages and related learning approaches.

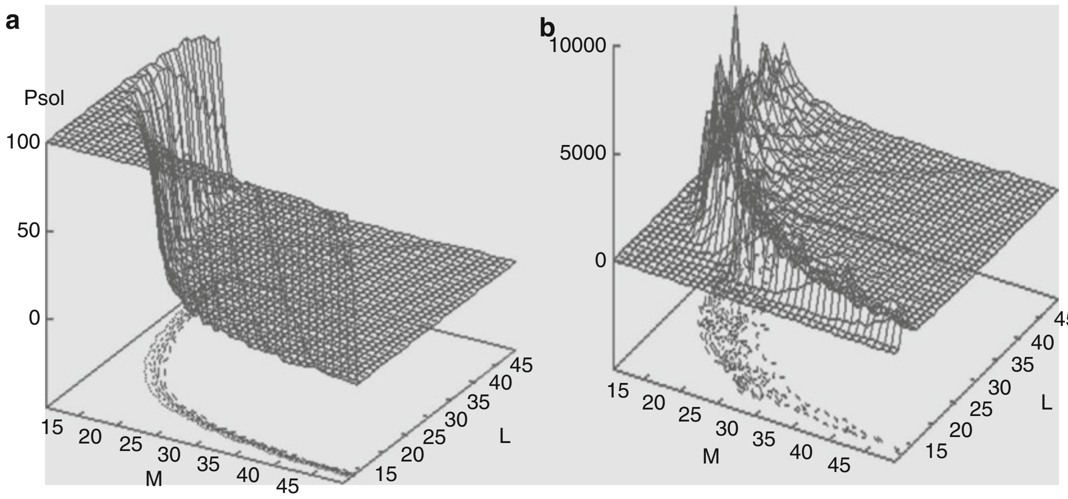
Typically, a learner proceeds iteratively: given a set $\mathcal{E}$ of examples labeled after a target concept ω, the learner maintains a list of candidate hypotheses, assessing their *completeness* (the proportion of positive examples they cover) and their *consistency* (the proportion of negative examples they do not cover) using a ▸ *covering test*. The covering test, checking whether some hypothesis h covers some example e, is thus a key component of the learning process, launched a few hundred thousand times in each learning run on medium-size problems.

While in propositional learning the covering test is straightforward and computationally efficient, in first-order logics, one must distinguish between *learning from interpretation* (h covers a set of facts e iff e is a model for h) and *learning from entailment* (h covers a clause e iff h entails e) (De Raedt 1997). A correct, but incomplete covering test, the ▸ θ-subsumption test defined by Plotkin (1970), is most often used for its decidability properties, and much attention has been paid to optimizing it (Maloberti and Sebag 2004).

As shown by Giordana and Saitta (2000), the θ-subsumption test is equivalent to a constraint satisfaction problem (CSP). A finite CSP is a tuple ($\mathbf{X}$, $\mathbf{R}$, D), where $\mathbf{X} = \{x_1, \ldots x_n\}$ is a set of variables, $\mathbf{R} = \{R_1, \ldots R_c\}$ is a set of constraints (relations), and D is the variable domain. Each relation R_h involves a subset of variables $x_{i_1}, \ldots, x_{i_k}$ in $\mathbf{X}$; it specifies all tuples of values $(a_{i_1}, \ldots, a_{i_k})$ in D^k such that the assignment $([x_{i_1} = a_{i_1}] \wedge \ldots \wedge [x_{i_k} = a_{i_k}])$ satisfies R_h. A CSP is satisfiable if there exists a tuple $(a_1, \ldots, a_n) \in D^n$ such that the assignment $([x_i = a_i], i = 1, \ldots, n)$ satisfies all relations in $\mathbf{R}$. Solving a CSP amounts to finding such a tuple (solution) or showing that none exists.

The probability for a random CSP instance to be satisfiable shows a PT with respect to the constraint density (control parameter $p_1 = \frac{2c}{n(n-1)}$) and constraint tightness ($p_2 = 1 - \frac{N}{L^2}$), where N denotes the cardinality of each constraint (assumed to be equal for all constraints) and L is the number of constants in the example (the universe).

The relational covering test being a CSP, a PT was expected and has been confirmed by empirical evidence (Botta et al. 1999; Giordana and Saitta 2000). The order parameter is the probability of hypothesis h to cover example e; the control parameters are the number m of predicates and the number n of variables in h, on the one hand, and the number N of literals built on each predicate symbol (relation) and the number L of constants in the example e, on the other hand. As shown in Fig. 1a, the covering probability is close to 1 (YES region) when h is general comparatively to e; it abruptly decreases to 0 (NO region) as the number m of predicates in h increases and/or the number L of constants in e decreases. In the PT region, a high peak



Phase Transitions in Machine Learning, Fig. 1 PT of the covering test (h, e) versus the number m of predicates in h and the number L of constants in e. The number n of variables is set to 10, and the number N of literals per predicate is set to 100. (**a**) Percentage of times the covering test succeeds. (**b**) Runtime of the covering test, averaged over 100 pairs (h, e) independently generated for each pair (m, L)

of empirical complexity of the covering test is observed (Fig. 1b).

The PT of the covering test has deep and far-reaching effects on relational learning. By definition, nontrivial hypotheses (covering some examples but not all) mostly belong to the PT region. The learner, searching for hypotheses covering the positive and rejecting the negative examples, must explore this region and thus cannot avoid the associated computational cost. More generally, the PT region acts as an *attractor* for any learner aimed at complete and consistent hypotheses.

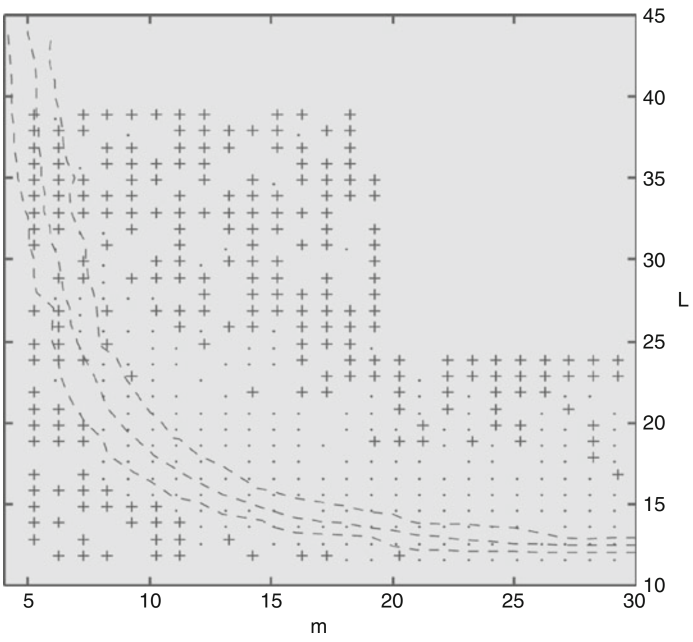
Secondly, top-down learners must traverse the plateau of overly general hypotheses (YES region) before arriving at the PT region. In the YES region, as all hypotheses cover most examples, the learner does not have enough information to make relevant choices; the chance of gradually arriving at an accurate description of the target concept thus becomes very low. Actually, a *blind spot* has been identified close to the PT (Botta et al. 2003): when the target concept lies in this region (relatively to the available examples), every state-of-the-art top-down relational learner tends to build random hypotheses, that is, the learned hypotheses behave like random guessing on the test set (Fig. 2).

This negative result has prompted the design of new relational learners aimed at learning in the PT region and using either prior knowledge about the size of the target concept (Ales Bianchetti et al. 2002) or near-miss examples (Alphonse and Osmani 2008).

Relational Kernels and MIL Problems

Relational learning has been revisited through the so-called kernel trick (Cortes and Vapnik 1995), first pioneered in the context of ▶ Support Vector Machines. Relational kernels, inspired from Haussler's convolutional kernels (Haussler 1999), have been developed for, e.g., strings, trees, or graphs. For instance, $K(\mathbf{x}, \mathbf{x}')$ might count the number of patterns shared by relational structures $\mathbf{x}$ and $\mathbf{x}'$. Relational kernels thus achieve a particular type of ▶ propositionalization (Kramer et al. 2001), mapping every relational example onto a propositional space defined after the training examples.

The question of whether relational kernels enable to avoid the PT faced by relational learning, described in the previous section, was investigated by Gaudel et al. (2007), focusing on the so-called ▶ multi-instance learning



Phase Transitions in Machine Learning, Fig. 2 Competence map of FOIL versus number m of predicates in the target concept and number L of constants in the examples. The target concept involves $n = 4$ variables and each example contains $N = 100$ literals built on each predicate symbol. For each pair (m, L), a target concept ω has been generated independently, balanced 200-example training, and test sets have been generated and labeled after ω. FOIL has been launched on the training set, and the predictive accuracy of the hypothesis has been assessed on the test set. Symbol "–" indicates a predictive accuracy greater than 90 %; symbol "–" indicates a predictive accuracy close to 50 % (akin random guessing)

(MIL) setting. The MIL setting, pioneered by Dietterich et al. (1997), is considered to be the "missing link" between relational and propositional learning (De Raedt 1998).

Multi-instance Learning: Background and Kernels

Formally, an MI example $\mathbf{x}$ is a bag of (propositional) instances noted $x^{(1)}$, …, $x^{(N)}$, where $x^{(j)} \in \mathbf{R}^d$. In the original MI setting (Dietterich et al. 1997), an example is labeled positive iff it includes at least one instance satisfying some target concept C:

$$pos(\mathbf{x}) \text{ iff } \exists i \in 1 \ldots N \ s.t.\ C(x^{(i)}).$$

More generally, in application domains such as image categorization, the example label might depend on the properties of several instances:

$$pos(\mathbf{x}) \text{ iff } \forall j = 1 \ldots m,\ \exists i_j \in 1 \ldots N \ s.t.\ C_j (x^{(i_j)}).$$

In this more general setting, referred to as *presence-based* setting, it has been shown that MIL kernels also have a PT (Gaudel et al. 2007).

Let us consider bag kernels K, built on the top of propositional kernels k on $\mathbf{R}^d$ as follows:

$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}).f(\mathbf{x}') \sum_{k=1}^{N} \sum_{\ell=1}^{N'} k(x^{(k)}, x'^{(\ell)}) \tag{1}$$

where $\mathbf{x} = (x^{(1)}, \ldots, x^{(N)})$ and $\mathbf{x}' = (x'^{(1)}, \ldots, x'^{(N')})$ denote two MI examples and $f(\mathbf{x})$ corresponds to a normalization term, e.g., $f(\mathbf{x}) = 1$ or $1/N$ or $1/\sqrt{K(\mathbf{x}, \mathbf{x})}$.

By construction, such MI-kernels thus consider the average similarity among the example instances, while relational learning is usually concerned with finding existential concepts.

The MI-SVM PT

After Botta et al. (2003) and Giordana and Saitta (2000), the competence of MI-kernels was experimentally assessed using artificial problems. Each problem involves m sub-concept s C_i: a given sub-concept corresponds to a region of the d-dimensional space, and it is satisfied by an MI example **x** if at least one instance in **x** belongs to this region. An instance is said to be relevant if it belongs to some C_i region.

Let n (respectively n') denote the number of relevant instances in positive (respectively negative) examples. Let further τ denote the number of sub-concept s not satisfied by negative examples (by definition, a positive example satisfies all sub-concept s).

Empirical investigations (Gaudel et al. 2007) show that:

- The $n = n'$ region is a failure region, where hypotheses learned by relational MI-SVMs do no better than random guessing (Fig. 3). In other words, while MI-SVMs grasp the notion of relevant instances, they still fail in the "truly relational region" where positive and negative examples only differ in the distribution of the relevant instances.
- The width of the failure region increases as τ increases, i.e., when fewer sub-concept s are satisfied by negative examples. This unexpected result is explained from the variance of the kernel-based propositionalization: the larger τ, the more the distribution of the positive and negative propositionalized examples overlap, hindering the discrimination.

Propositional Learning and Sparse Coding

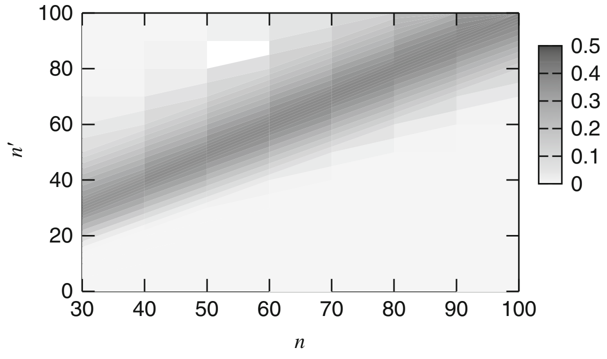
Interestingly, the emergence of a PT is not limited to relational learning. In the case of (context-free) *grammar induction*, for instance (Cornuéjols and Sebag 2008), the coverage of the candidate grammar was found to abruptly go to 1 along (uniform) generalization, as depicted in Fig. 4.

Propositional learning also displays some PTs both in the classification (Baskiotis and Sebag 2004; Rückert and De Raedt 2008) and in the regression (Cands 2008; Donoho and Tanner 2005) context.

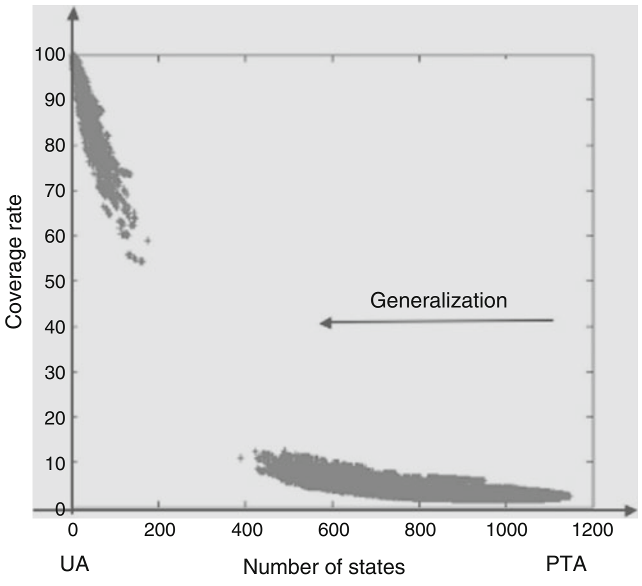
Propositional Classification

Given a target hypothesis language, classification in discrete domains most often aims at the simplest expression complying with the training examples.

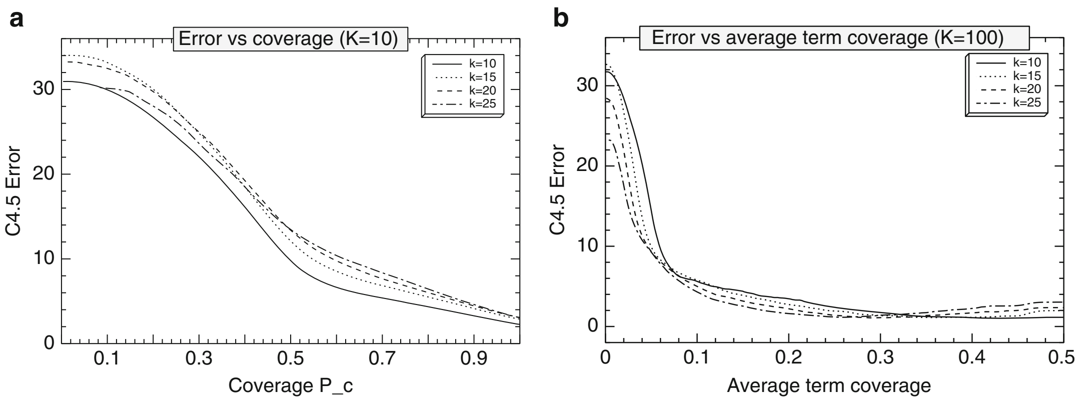
Considering randomly generated positive and negative examples, Rückert and De Raedt (2008) investigated the existence of k-term DNF solutions (disjunction of at most k conjunctions of literals) and showed that the probability of solution abruptly drops as the number of negative examples increases. They proposed a combinatorial optimization algorithm to find a k-term DNF



Phase Transitions in Machine Learning, Fig. 3 MI-SVM failure region in the (n, n') plane. Each (n, n') point reports the test error, averaged on 40 artificial problems



Phase Transitions in Machine Learning, Fig. 4 Gap emerging during learning in the relationship between the number of nodes of the inferred grammar and the coverage rate



Phase Transitions in Machine Learning, Fig. 5 C4.5 error versus concept coverage (**a**) and average term coverage (**b**) in k-term DNF languages. The reported curve is obtained by Gaussian convolution with empirical data (15,000 learning problems, each one involving a 800-example dataset)

complying with the training examples except at most ε % of them (Rückert and De Raedt 2008).

Considering positive and negative examples generated after some k-term DNF target concept ω, Baskiotis and Sebag examined the solutions built by C4.5 Rules (Quinlan 1993), among the oldest and still most used discrete learning algorithms. The observed variable is the generalization error on a test set; the order variables are the coverage of ω and the average coverage of the conjuncts in ω. Interestingly, C4.5 displays a PT behavior (Fig. 5): the error abruptly increases as the coverage and average coverage decrease.

Propositional Regression

▶ Linear regression aims at expressing the target variable as the weighted sum of the N descriptive variables according to some vector $\mathbf{w}$. When the number N of variables is larger than the number n of examples, one is interested in finding the most sparse $\mathbf{w}$ complying with the training examples (s.t. $< \mathbf{w}, \mathbf{x_i} >= \mathbf{y_i}$). The sparsity criterion consists of minimizing the L_0 norm of

w (number of nonzero coefficients in **w**), which defines an NP optimization problem. A more tractable formulation is obtained by minimizing the L_1 norm instead:

$$\text{Find } \arg\min_{\mathbf{w}\in\mathbf{R}^N}\{||\mathbf{w}||_1 \text{ subject to } <\mathbf{w},\mathbf{x_i}> = \mathbf{y_i}, i = 1\ldots n\}. \quad (2)$$

A major result in the field of sparse coding can be stated as: *Let* $\boldsymbol{w}^*$ *be the solution of Eq.* (2)*; if it is sufficiently sparse,* $\boldsymbol{w}^*$ *also is the most sparse vector subject to* $<\boldsymbol{w},\boldsymbol{x_i}> = \boldsymbol{y_i}$ (Donoho and Tanner 2005). In such cases, the L_0 norm minimization can be solved by L_1 norm minimization (an NP optimization problem is solved using linear programming). More generally, the equivalence between L_0 and L_1 norm minimization shows a PT behavior: when the sparsity of the solution is lower than a given threshold w.r.t the problem size (lower curve in Fig. 6), the NP/LP equivalence holds strictly; further, there exists a region (between the upper and lower curves in Fig. 6) where the NP/LP equivalence holds with high probability.

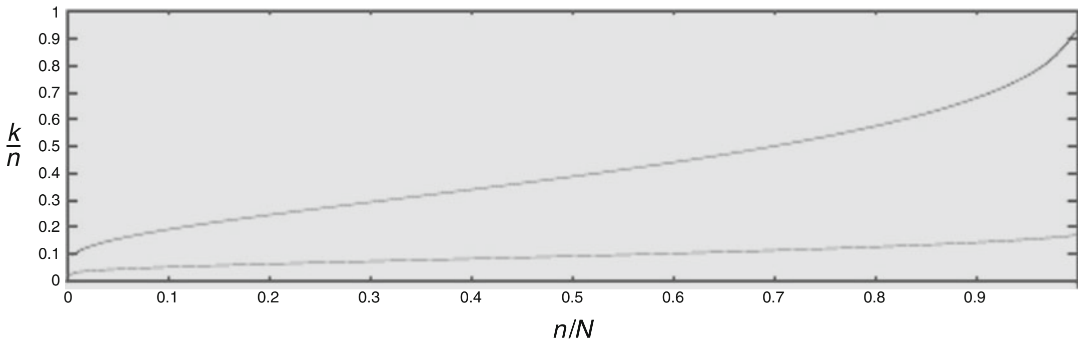
This highly influential result bridges the gap between the statistical and algorithmic objectives. On the statistical side, the importance of sparsity in terms of robust coding (hence learning) is acknowledged since the beginnings of information theory; on the algorithmic side, the sparsity criterion cannot be directly tackled as it boils down to solving a combinatorial optimization problem (minimizing a L_0 norm). The above result reconciles sparsity and tractability by noting that under some conditions, the solution of the L_0 minimization problem can be found by solving the (tractable) L_1 minimization problem: whenever the solution of the latter problem is "sufficiently" sparse, it is also the solution of the former problem.

Perspectives

Since the main two formulations of ML involve constraint satisfaction and constrained optimization, it is no surprise that CSP PTs manifest themselves in ML. The diversity of these manifestations, ranging from relational learning (Botta et al. 2003) to sparse regression (Donoho and Tanner 2005), has been illustrated in this entry, without pretending exhaustivity.

Along this line, the research agenda and methodology of ML can benefit from the lessons learned in the CSP field. Firstly, algorithms must be assessed on problems lying in the PT region; results obtained on problems in the easy regions are likely to be irrelevant (*playing in the sandbox* Hogg et al. 1996).

In order to do so, the PT should be localized through defining control and order parameters, thus delineating several regions in the control parameter space (ML landscape). These regions expectedly correspond to different types of ML difficulty, beyond the classical computational complexity perspective.



Phase Transitions in Machine Learning, Fig. 6 Strong and weak PT in sparse regression (Donoho and Tanner 2005). The x-axis is the ratio between the number n of constraints and the number N of variables; the y-axis is the ratio between the number k of variables involved in the solution and n

Secondly, the response of a given algorithm to these difficulties can be made available through a competence map, depicting its average performance conditionally to the value of the control parameters as shown in Figs. 2 and 3.

Finally, such competence maps can be used to determine whether a given algorithm is a priori relevant in a given region of the control parameter space and support the algorithm selection task (a.k.a. meta-learning; see, e.g., http://www.cs.bris.ac.uk/Research/MachineLearning/metal.html).

Recently, phase transitions have also emerged in learning more complex structures, such as complex networks. For instance, Xhang et al. (Zhang et al. 2014), following previous work, investigated the transitions occurring in the possibility of discovering communities in sparse networks using a semisupervised clustering approach. In their approach, the control parameter is the fraction α of nodes in the network, whose label is known, and they found both a first-order and a second-order phase transition.

Recommended Reading

Ales Bianchetti J, Rouveirol C, Sebag M (2002) Constraint-based learning of long relational concepts. In: Sammut C (ed) Proceedings of international conference on machine learning, ICML'02. Morgan Kauffman, San Francisco, pp 35–42

Alphonse E, Osmani A (2008) On the connection between the phase transition of the covering test and the learning success rate. Mach Learn 70(2–3):135–150

Baskiotis N, Sebag M (2004) C4.5 competence map: a phase transition-inspired approach. In: Proceedings of international conference on machine learning. Morgan Kaufman, Banff, pp 73–80

Botta M, Giordana A, Saitta L (1999) An experimental study of phase transitions in matching. In: Proceedings of the 16th international joint conference on artificial intelligence, Stockholm, pp 1198–1203

Botta M, Giordana A, Saitta L, Sebag M (2003) Relational learning as search in a critical region. J Mach Learn Res 4:431–463

Cands EJ (2008) The restricted isometry property and its implications for compressed sensing. Compte Rendus de l'Academie des Sciences, Paris, Serie I 346:589–592

Cheeseman P, Kanefsky B, Taylor W (1991) Where the really hard problems are. In: Myopoulos R, Reiter J (eds) Proceedings of the 12th international joint conference on artificial intelligence, Sydney. Morgan Kaufmann, San Francisco, pp 331–340

Cornuéjols A, Sebag M (2008) A note on phase transitions and computational pitfalls of learning from sequences. J Intell Inf Syst 31(2):177–189

Cortes C, Vapnik VN (1995) Support-vector networks. Mach Learn 20:273–297

De Raedt L (1997) Logical setting for concept-learning. Artif Intell 95:187–202

De Raedt L (1998) Attribute-value learning versus inductive logic programming: the missing links. In: Proceedings inductive logic programming, ILP. LNCS, vol 2446. Springer, London, pp 1–8

Demongeot J, Sené S (2008) Boundary conditions and phase transitions in neural networks. Simulation results. Neural Netw 21(7):962–970

Dietterich T, Lathrop R, Lozano-Perez T (1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Donoho DL, Tanner J (2005) Sparse nonnegative solution of underdetermined linear equations by linear programming. Proc Natl Acad Sci 102(27):9446–9451

Engel A, Van den Broeck C (2001) Statistical mechanics of learning. Cambridge University Press, Cambridge

Gaudel R, Sebag M, Cornuéjols A (2007) A phase transition-based perspective on multiple instance kernels. In: Proceedings of international conference on inductive logic programming, ILP, Corvallis, pp 112–121

Gaudel R, Sebag M, Cornuéjols A (2008) A phase transition-based perspective on multiple instance kernels. Lect Notes Comput Sci 4894:112–121

Giordana A, Saitta L (2000) Phase transitions in relational learning. Mach Learn 41(2):17–251

Haussler D (1999) Convolutional kernels on discrete structures. Technical report, Computer Science Department, University of California at Santa Cruz

Hogg T, Huberman BA, Williams CP (eds) (1996) Artificial intelligence: special issue on frontiers in problem solving: phase transitions and complexity, vol 81(1–2). Elsevier

Kramer S, Lavrac N, Flach P (2001) Propositionalization approaches to relational data mining. In: Dzeroski S, Lavrac N (eds) Relational data mining. Springer, New York, pp 262–291

Maloberti J, Sebag M (2004) Fast theta-subsumption with constraint satisfaction algorithms. Mach Learn J 55:137–174

Mitchell TM (1982) Generalization as search. Artif Intell 18:203–226

Plotkin G (1970) A note on inductive generalization. In: Machine intelligence, vol 5. Edinburgh University Press, Edinburgh

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Francisco

Rückert U, De Raedt L (2008) An experimental evaluation of simplicity in rule learning. Artif Intell 172(1):19–28
Zhang P, Moore C, Zdeborova L (2014) Phase transitions in semisupervised clustering of sparse networks. CoRR vol abs/1404.7789

Piecewise Constant Models

▶ Regression Trees

Piecewise Linear Models

▶ Model Trees

Plan Recognition

▶ Inverse Reinforcement Learning

Polarity Learning on a Stream

▶ Opinion Stream Mining

Policy Gradient Methods

Jan Peters[1,2,4] and J. Andrew Bagnell[3]
[1]Department of Empirical Inference, Max-Planck Institute for Intelligent Systems, Tübingen, Germany
[2]Intelligent Autonomous Systems, Computer Science Department, Technische Universität Darmstadt, Darmstadt, Hessen, Germany
[3]Carnegie Mellon University, Pittsburgh, PA, USA
[4]Max Planck Institute for Biological Cybernetics, Tübingen, Germany

Abstract

Already Richard Bellman suggested that searching in policy space is fundamentally different from value function-based reinforcement learning — and frequently advantageous, especially in robotics and other systems with continuous actions. Policy gradient methods optimize in policy space by maximizing the expected reward using a direct gradient ascent. We discuss their basics and the most prominent approaches to policy gradient estimation.

Definition

A policy gradient method is a ▶ reinforcement learning approach that directly optimizes a parametrized control policy by a variant of gradient descent. These methods belong to the class of ▶ policy search techniques that maximize the expected return of a policy from a fixed class, in contrast with ▶ value function approximation approaches that derive policies indirectly from an estimated value function. Policy gradient approaches have various advantages: they enable the straightforward incorporation of domain knowledge in policy parametrization; often an optimal policy is more compactly represented than the corresponding value function; many such methods guarantee to convergence to at least a locally optimal policy; the methods naturally handle continuous states and actions and often even imperfect state information. The countervailing drawbacks include difficulties in off-policy settings, the potential for very slow convergence and high sample complexity, as well as identifying local optima that are not globally optimal.

Structure of the Learning System

Policy gradient methods center around a parametrized policy π_θ, also known as a ▶ direct controller, with parameters θ that defines the selection of actions a given the state s. Such a policy may either be deterministic $a = \pi_\theta(s)$ or stochastic $a \sim \pi_\theta(a|s)$. This choice also affects the class of policy gradient algorithms applicable (stochastic policies often lead to smooth differentiable objective with gradients that can be estimated via likelihood ratio methods (Williams 1992), where a deterministic policy may lead to a non-smooth optimization problem), influences how the exploration-exploitation dilemma is addressed (e.g., a stochastic policy naturally chooses novel actions while a deterministic policy requires the perturbation of policy parameters or sufficient stochasticity in the system to achieve exploration), and may affect the quality of optimal solution (e.g., for a time-invariant or stationary policy, the optimal policy can be stochastic Sutton et al. 2000). Frequently used policy classes include Gibbs distributions $\pi_\theta(a|s) = \exp(\phi(s,a)^T\theta)/\sum_b \exp(\phi(s,b)^T\theta)$ for discrete problems (Sutton et al. 2000; Bagnell 2004) and, for continuous problems, Gaussian policies $\pi_\theta(a|s) = \mathcal{N}(\phi(s,a)^T\theta_1, \theta_2)$ with an exploration parameter θ_2 (Williams 1992; Peters and Schaal 2008).

Expected Return

Policy gradient methods seek to optimize the expected return of a policy π_θ,

$$J(\theta) = Z_\gamma E\left\{\sum_{k=0}^{H} \gamma^k r_k\right\},$$

where $\gamma \in [0,1]$ denotes a discount factor, a normalization constant Z_γ, and H the planning horizon. For finite H, we have an episodic reinforcement learning scenario where the truly optimal policy is nonstationary and the normalization does not matter. For an infinite horizon $H = \infty$, we choose the normalization to be $Z_\gamma \equiv (1-\gamma)$ for $\gamma < 1$ and $Z_1 \equiv \lim_{\gamma\to 1}(1-\gamma) = 1/H$ for ▶ average reward reinforcement learning problem where $\gamma = 1$.

Gradient Descent in Policy Space

Policy gradient methods follow an estimate the gradient of the expected return

$$\theta_{k+1} = \theta_k + \alpha_k g(\theta_k),$$

where $g(\theta_k) \approx \nabla_\theta J(\theta)|_{\theta=\theta_k}$ is a gradient estimate for the policy with parameters $\theta = \theta_k$ after update k (with an initial policy θ_0) and α_k denotes a learning rate. If the gradient estimator is unbiased, $\sum_{k=0}^{\infty}\alpha_k \to \infty$ while $\sum_{k=0}^{\infty}\alpha_k^2$ remains bounded, convergence to a local minimum can be guaranteed. In optimal control, model-based gradient methods have been used for optimizing policies since the 1960s (Pontryagin et al. 1962). While these are used machine learning community (e.g., differential dynamic programming with learned models), they may be numerically brittle and must rely on accurate, deterministic models. Hence, they may suffer significantly from optimization biases (i.e., if possible, they will reach a higher average return on the approximate model than possible on the real system by exploiting the shortcomings of the model) and are not generally applicable as learning problems often include discrete elements and maybe very difficult to learn effective predictive models.

Several model-free alternatives can be found in the simulation-based optimization literature (Fu 2006), including, e.g., finite-difference gradients, likelihood ratio approaches, response-surface methods, and mean-valued, weak derivatives. The advantages and disadvantages of these different approaches remain a fiercely debated topic (Fu 2006). In machine learning, the first two approaches have largely dominated gradient-based approaches to ▶ policy search, although response surface methods are arriving especially in the context of Bayesian optimization for policy search.

Finite Difference Gradients

The simplest policy gradient approaches with perhaps the most practical applications (see Bagnell (2004) and Peters and Schaal (2008) for robotics application of this method) estimate the gradient by perturbing the policy parameters. For a current policy θ_k with expected return $J(\theta_k)$, this approach will create perturbed policies $\hat{\theta}_i = \theta_k + \delta\theta_i$ with the approximated expected returns given by $J(\hat{\theta}_i) \approx J(\theta_k) + \delta\theta_i^T g$ where $g = \nabla_\theta J(\pi_\theta)|_{\theta=\theta_k}$. Such returns are typically estimated by simulation. The gradient can then be estimated by linear regression; i.e., we obtain

$$g = (\Delta\Theta^T\Delta\Theta)^{-1}\Delta\Theta^T\Delta J,$$

with parameter perturbations $\Delta\Theta = [\delta\theta_1, \ldots, \delta\theta_n]$ and mean-subtracted roll-out returns $\delta J_n = J(\hat{\theta}_i) - \overline{J(\theta_k)}$ form $\Delta J = [\delta J_1, \ldots, \delta J_n]$. The choice of the parameter perturbation largely determines the performance of the approach (Spall 2003). Limitations particular to this approach include the need for many exploratory samples, the sensitivity of the system with respect to each parameter may differs by orders of magnitude, small changes in a single parameter may render a system unstable, and stochasticity requires particular care in optimization (e.g., multiple samples, fixed random seeds, etc.), see Glynn (1990) and Spall (2003). This method is additionally referred to as the *naive Monte-Carlo policy gradient.*

Likelihood Ratio Gradients

The likelihood ratio method relies upon the stochasticity of either the policy for model-free approaches, or the system in the model-based case. Hence, it requires no explicit parameter exploration and may cope better with noise as well as parameter perturbation sensitivity problems. Moreover, in the model-free setting, in contrast with naive Monte-Carlo estimation, it potentially benefits from more assumptions on the policy parameterization. Denoting a time-indexed sequence of states, actions, and rewards of the joint system composed of the policy and environment as a *path*, a parameter setting induces a path distribution $p_\theta(\tau)$ and rewards $R(\tau) = Z_\gamma \sum_{k=0}^{H} \gamma^k r_k$ along a path τ. Thus, we may write the gradient of the expected return as

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau) d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau \\ &= E\{\nabla_\theta \log p_\theta(\tau) R(\tau)\}.\end{aligned}$$

If our system $p(s'|s,a)$ is Markovian, we may use $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, a_k) \pi_\theta(a_k|s_k)$ for a stochastic policy $a \sim \pi_\theta(a|s)$ to obtain the model-free policy gradient estimator known as Episodic REINFORCE (Williams 1992)

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \sum_{k=h}^{H} \gamma^{k-h} r_k\right\},$$

and for the deterministic policy $a = \pi_\theta(s)$, the model-based policy gradient

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \left(\nabla_a \log p(s_{k+1}|s_k, a_k)^T \nabla_\theta \pi_\theta(s)\right) \sum_{k=h}^{H} \gamma^{k-h} r_k\right\},$$

follows from $p_\theta(\tau) = p(s_0) \prod_{h=0}^{H} p(s_{k+1}|s_k, \pi_\theta(s_k))$.

Note that all rewards preceding an action may be omitted as the cancel out in expectation. Using a state-action value function $Q^{\pi_\theta}(s,a,h) = E\left\{\sum_{k=h}^{H} \gamma^{k-h} r_k \middle| s, a, \pi_\theta\right\}$ (see ▸ value function approximation), we can rewrite REINFORCE in its modern form

$$\nabla_\theta J(\theta) = Z_\gamma E\left\{\sum_{h=0}^{H} \gamma^k \nabla_\theta \log \pi_\theta(a_k|s_k) \left(Q^{\pi_\theta}(s,a,h) - b(s,h)\right)\right\},$$

known as the policy gradient theorem where the baseline $b(s,h)$ is an arbitrary function that may be used to reduce the variance, and $Q^{\pi_\theta}(s,a,h)$ represents the action-▸ value function.

While likelihood ratio gradients have been known since the late 1980s, they have recently experienced an upsurge of interest due to their demonstrated effectiveness in applications; see, e.g., Peters and Schaal (2008)), progress toward variance reduction using optimal baselines (Lawrence et al. 2003), rigorous understanding of the relationships between value functions and policy gradients (Sutton et al. 2000), policy gradients in reproducing kernel Hilbert space (Bagnell 2004), as well as faster, more robust convergence using natural policy gradients (Bagnell 2004; Peters and Schaal 2008)

A recent major development (Silver et al. 2014) demonstrates that many of the key results from model-free stochastic policy search can be transferred to deterministic policy classes by considering the limiting case of a likelihood ratio method. Importantly, this estimation of a deterministic policy gradient can be much more sample efficient than existing techniques; the caveat remains that the total return may indeed fail to be differentiable and both practical performance and theory in such settings are poorly understood.

Cross-References

▶ Policy Search
▶ Reinforcement Learning
▶ Value Function Approximation

Recommended Reading

Bagnell JA (2004) Learning decisions: robustness, uncertainty, and approximation. Doctoral dissertation, Robotics institute, Carnegie Mellon University, Pittsburgh

Fu MC (2006) Stochastic gradient estimation. In: Henderson SG, Nelson BL (eds) Handbook on operations research and management science: simulation, vol 19. Elsevier, Burlington, pp 575–616

Glynn P (1990) Likelihood ratio gradient estimation for stochastic systems. Commun ACM 33(10):75–84

Lawrence G, Cowan N, Russell S (2003) Efficient gradient estimation for motor control learning. In: Proceedings of the international conference on uncertainty in artificial intelligence (UAI), Acapulco

Peters J, Schaal S (2008) Reinforcement learning of motor skills with policy gradients. Neural Netw 21(4):682–697

Pontryagin LS, Boltyanskii VG, Gamkrelidze RV, Mishchenko E (1962) The mathematical theory of optimal processes. International series of monographs in pure and applied mathematics. Interscience publishers, New York

Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st international conference on Machine learning (ICML), Bejing

Spall JC (2003) Introduction to stochastic search and optimization: estimation, simulation, and control. Wiley, Hoboken

Sutton RS, McAllester D, Singh S, Mansour Y (2000) Policy gradient methods for reinforcement learning with function approximation. In: Solla SA, Leen TK, Mueller KR (eds) Advances in neural information processing systems (NIPS). MIT, Denver

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8:229–256

Policy Search

▶ Markov Decision Processes

POMDPs

▶ Partially Observable Markov Decision Processes

POS Tagging

Walter Daelemans
CLIPS University of Antwerp, Antwerpen, Belgium

Synonyms

Grammatical tagging; Morphosyntactic disambiguation; Part of speech tagging; Tagging

Definition

Part-of-speech tagging (POS tagging) is a process in which each word in a text is assigned its appropriate morphosyntactic category (for example *noun-singular*, *verb-past*, *adjective*, *pronoun-personal*, and the like). It therefore provides information about both morphology (structure of words) and syntax (structure of sentences). This disambiguation process is determined both by constraints from the lexicon (what are the possible categories for a word?) and by constraints from the context in which the word occurs (which of the possible categories is the right one in this context?). For example, a word like *table* can be a noun-singular, but also a verb-present (as in *I table this motion*). This is lexical knowledge. It is the context of the word that should be used to decide which of the possible categories is the correct one. In a sentence like *Put it on the table,* the fact that *table* is preceded by the determiner *the*, is a good indication that it is used as a noun here. Systems that automatically assign parts of speech to words in text should take into account both lexical and contextual constraints, and they are typically found in implementations as a lookup module and a disambiguation module.

Motivation and Background

In most natural language processing (NLP) applications, POS tagging is one of the first steps to allow abstracting away from individual words. It is not to be confused with *lemmatization*, a process that reduces morphological variants of words to a canonical form (the citation form, for example, infinitive for verbs and singular for nouns). Whereas lemmatization allows abstraction over different forms of the same word, POS tagging abstracts over sets of different words that have the same function in a sentence. It should also not be confused with *tokenization*, a process that detects word forms in text, stripping off punctuation, handling abbreviations, and so on. For example, the string *don't* could be converted to *do not*. Normally, a POS tagging system would take tokenized text as input. More advanced tokenizers may even handle multiword items, for example treating *in order to* not as three separate words but as a single lexical item.

Applications. A POS tagger is the first disambiguation module in text analysis systems. In order to determine the syntactic structure of a sentence (and its semantics), we have to know the parts of speech of each word. In earlier approaches to syntactic analysis (parsing), POS tagging was part of the parsing process. However, individually trained and optimized POS taggers have increasingly become a separate module in shallow or deep syntactic analysis systems. By extension, POS tagging is also a foundational module in text mining applications ranging from information extraction and terminology/ontology extraction to summarization and question answering.

Apart from being one of the first modules in any text analysis system, POS tagging is also useful in linguistic studies (corpus linguistics) – for example for computing frequencies of disambiguated words and of superficial syntactic structures. In speech technology, knowing the part of speech of a word can help in speech synthesis (the verb "subJECT" is pronounced differently from the noun "SUBject"), and in speech recognition, POS taggers are used in some approaches to language modeling. In spelling and grammar checking, POS tagging plays a role in increasing the precision of such systems.

Part-of-speech tag sets. The inventory of POS tags can vary from tens to hundreds depending on the richness of morphology and syntax that is represented and on the inherent morphological complexity of a language. For English, the tag sets most used are those of the Penn Treebank (45 tags; Marcus et al. 1993), and the CLAWS C7 tag set (146 tags; Garside and Smith 1997). Tag sets are most often developed in the context of the construction of annotated corpora. There have been efforts to standardize the construction of tag sets to increase translatability between different tag sets, such as Eagles. (http://www.ilc.cnr.it/EAGLES96/browse.html) and ISO/TC 37/SC 4. (http://www.tc37sc4.org/)

The following example shows both tag sets. By convention, a tagged word is represented by attaching the POS tag to it, separated by a slash.

Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ./. [Penn Treebank]

Pierre/NP1 Vinken/NP1 ,/, 61/MC years/NNT2 old/JJ ,/, will/VM join/VVI the/AT Board/NN1 as/II a/AT1 nonexecutive/JJ director/NN1 Nov./NPM1 29/MC ./. [CLAWS C7]

As can be seen, the tag sets differ in level of detail. For example, NNT2 in the C7 tag set indicates a plural temporal noun (as a specialization of the word class noun), whereas the Penn

Treebank tag set only specializes to plural noun (NNS).

Like most tasks in NLP, POS tagging is a disambiguation task, and both linguistic knowledge-based handcrafting methods and corpus-based learning methods have been proposed for this task. We will restrict our discussion here to the statistical and machine learning approaches to the problem, which have become mainstream because of the availability of large POS tagged corpora and because of better accuracy in general than handcrafted systems. A state of the art system using a knowledge-based approach is described in Karlsson et al. (1995).

A decade old now, but still a complete and informative book-length introduction to the field of POS tagging is van Halteren (1999). It discusses many important issues that are not covered in this article (performance evaluation, history, handcrafting approaches, tag set development issues, handling unknown words, and more.). A more recent introductory overview is Chap. 5 in Jurafsky and Martin (2008).

Statistical and Machine Learning Approaches to Tagging

In the late 1970s, statistical approaches based on n-gram probabilities (probabilities that sequences of n tags occur in a corpus) computed on frequencies in tagged corpora have already been proposed by the UCREL team at the University of Lancaster (Garside and Smith 1997). These early models lacked a precise mathematical framework and a principled solution to working with zero- or low probability frequencies. It was realized that Hidden Markov Models (HMM) in use in speech recognition were applicable to the tagging problem as well.

HMMs

HMMs are probabilistic finite state automata that are flexible enough to combine n-gram information with other relevant information to a limited extent. They allow supervised learning by computing the probabilities of n-grams from tagged corpora, and unsupervised learning using the Baum-Welch algorithm. Finding the most probable tag sequence given a sequence of words (decoding) is done using the Viterbi search. In combination with smoothing methods for low-frequency events and special solutions for handling unknown words, this approach results in a state-of-the-art tagging performance. A good implementation is TnT (Trigrams'n Tags Brants 2000).

Transformation-Based Error-Driven Learning (Brill-Tagging)

Transformation-based learning is an eager learning method in which the learner extracts a series of rules, each of which transforms a tag into another tag given a specific context. Learning starts with an initial annotation (e.g., tag each word in a text by the POS tag it is most frequently associated with in a training corpus), and compares this annotation with a gold standard annotation (annotated by humans). Discrepancies trigger the generation of rules (constrained by templates), and in each cycle, the best rule is chosen. The best rule is the one that most often leads to a correct transformation in the whole training corpus (Brill 1995a). An unsupervised learning variant (using a lexicon with word-tag probabilities) is described in Brill (1995b). Fully unsupervised POS tagging can also be achieved using distributional clustering techniques, as pioneered by Schutze (1995). However, these methods are hard to evaluate and compare to supervised approaches. The best way to evaluate them is indirectly, in an application-oriented way, as in Ushioda (1996).

Other Supervised Learning Methods

As a supervised learning task, POS tagging has been handled mostly as in a *sliding window* representation. Instances are created by making each word in each sentence a focus feature of an instance, and adding the left and right context as additional features. The class to be predicted is the POS tag of the focus word. Instead of using the words themselves as features, information about them can be used as features as well (e.g., capitalized or not, hyphenated or not, the POS tag of the word for left context words as predicted by the tagger previously, a symbol representing the possible lexical categories of the focus word and right context words, first and last letters of the word in each position, and so on.).

The following table lists the structure of instance representations for part of the sentence shown earlier. In this case the words themselves are feature values, but most often other derived features would replace these because of sparseness problems.

		Focus		Class	
=	=	Pierre	Vinken		NNP
=	Pierre	Vinken		61	NNP
Pierre	Vinken		61	years	
Vinken		61	years	old	CD

Most classification-based, supervised machine learning methods can be, and have been applied to this problem, including decision tree learning (Schmid 1994b), memory-based learning (Daelemans et al. 1996), maximum entropy models (Ratnaparkhi 1996), neural networks (Schmid 1994a), ensemble methods (van Halteren et al. 2001), and many others. All these methods seem to converge to a 96–97 % accuracy rate on the Wall Street Journal corpus using the Penn Treebank tag set. In a systematic comparison of some of the methods listed here, van Halteren et al. (2001) found that TnT outperforms maximum entropy and memory-based learning methods, which in turn outperform Brill tagging. Non-propositional supervised learning methods have been applied to the task as well (Cussens 1997) with grammatical structure as background knowledge with similar results. The best results reported on the WSJ corpus so far is bidirectional perceptron learning (Shen et al. 2007) with a 97.33 % accuracy.

Because of these high scores, POS tagging (at least for English) is considered by many a solved problem. However, as for most machine-learning based NLP systems, domain adaptation is still a serious problem for POS tagging. A tagger trained to high accuracy on newspaper language will fail miserably on other types of text, such as medical language.

Cross-References

▶ Classification
▶ Clustering
▶ Decision Tree
▶ Document Categorization
▶ Inductive Logic Programming
▶ Information Retrieval
▶ Lazy Learning
▶ Maxent Models
▶ Text Mining

Recommended Reading

Brants T (2000) TnT – a statistical part-of-speech tagger. In: Proceedings of the sixth applied natural language processing conference ANLP-2000, Seattle

Brill E (1995a) Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. Comput Linguist 21(4):543–565

Brill E (1995b) Unsupervised learning of disambiguation rules for part of speech tagging. In: Proceedings of the third workshop on very large corpora. Ohio State University, Ohio, pp 1–13

Cussens J (1997) Part-of-speech tagging using progol. In: Lavrac N, Dzeroski S (eds) Proceedings of the seventh international workshop on inductive logic programming. Lecture notes in computer science, vol 1297. Springer, London, pp 93–108

Daelemans W, Zavrel J, Berck P, Gillis S (1996) MBT: a memory-based part of speech tagger generator. In: Proceedings of the fourth workshop on very large corpora, Copenhagen, pp 14–27

Garside R, Smith N (1997) A hybrid grammatical tagger: CLAWS4. In: Garside R, Leech G, McEnery A (eds) Corpus annotation: linguistic information from computer text corpora. Longman, London, pp 102–121

Jurafsky D, Martin J (2008) Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd edn. Prentice Hall, Upper Saddle River

Karlsson F, Voutilainen A, Heikkilä J, Anttila A (1995) Constraint grammar. A language-independent system for parsing unrestricted text. Mouton de Gruyter, Berlin/New York, p 430
Marcus M, Santorini B, Marcinkiewicz M (1993) Building a large annotated corpus of English: the Penn Treebank. Comput Linguist 19(2):313–330
Ratnaparkhi A (1996) A maximum entropy part of speech tagger. In: Proceedings of the ACL-SIGDAT conference on empirical methods in natural language processing, Philadelphia, pp 17–18
Schmid H (1994a) Part-of-speech tagging with neural networks. In: Proceedings of COLING-94, Kyoto, pp 172–176
Schmid H (1994b) Probabilistic part-of-speech tagging using decision trees. In: Proceedings of the international conference on new methods in language processing (NeMLaP), Manchester, pp 44–49
Schutze H (1995) Distributional part-of-speech tagging. In: Proceedings of EACL 7, Dublin, pp 141–148
Shen L, Satta G, Joshi A (2007) Guided learning for bidirectional sequence classification. In: Proceedings of the 45th annual meetings of the association of computational linguistics (ACL 2007), Prague, pp 760–767
Ushioda A (1996) Hierarchical clustering of words and applications to NLP tasks. In: Proceedings of the fourth workshop on very large corpora, Somerset, pp 28–41
van Halteren H (ed) (1999) Syntactic wordclass tagging. Kluwer Academic Publishers, Boston
van Halteren H, Zavrel J, Daelemans W (2001) Improving accuracy in NLP through combination of machine learning systems. Comput Linguist 27(2):199–229

Positive Definite

▶ Positive Semidefinite

Positive Predictive Value

▶ Precision

Positive Semidefinite

Synonyms

Positive definite

Definition

A symmetric $m \times m$ matrix K satisfying $\forall x \in c^m : x^* Kx \geq 0$ is called positive semidefinite. If the equality only holds for $x = \vec{0}$ the matrix is positive definite.

A function $k : X \times X \to c, X \neq \emptyset$, is positive (semi-) definite if for all $m \in n$ and all $x_1, \ldots, x_m \in X$ the $m \times m$ matrix $\vec{K}$ with elements $K_{ij} := k(x_i, x_j)$ is positive (semi-) definite.

Sometimes the term strictly positive definite is used instead of positive definite, and positive definite refers then to positive semidefiniteness.

Posterior

▶ Posterior Probability

Posterior Probability

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Posterior

Definition

In Bayesian inference, a *posterior probability* of a value x of a random variable X given a context a value y of a random variable Y, $P(X = x|Y = y)$, is the probability of X assuming the value x in the context of $Y = y$. It contrasts with the ▶ prior probability, $P(X = x)$, the probability of X assuming the value x in the absence of additional information.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1 %, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50 % of people who have skin discolorations of greater than 1 cm width (sd > 1 cm) have exoma. It follows that the posterior probability of exoma given sd >1 cm, P(exoma = true | sd >1 cm = true), is 0.500.

Cross-References

▶ Bayesian Methods

Post-pruning

Definition

Post-pruning is a ▶ Pruning mechanism that first learns a possibly ▶ Overfitting hypothesis and then tries to simplify it in a separate learning phase.

Cross-References

▶ Overfitting
▶ Pre-pruning
▶ Pruning

Postsynaptic Neuron

The neuron that receives signals via a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Precision

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Synonyms

Positive predictive value

Precision, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Definition

Precision is defined as the ratio of true positives (TP) and the total number of positives predicted by a model. This is defined with reference to a special case of the ▶ confusion matrix, with two classes: one designated the *positive* class and the other the *negative* class, as indicated in Table 1.

Precision can then be defined in terms of true positives and false positives (FP) as follows.

Precision = TP/(TP + FP)

Cross-References

▶ Precision and Recall

Precision and Recall

Kai Ming Ting
Federation University, Mount Helen, VIC, Australia

Definition

▶ Precision and recall are the measures used in the information retrieval domain to measure how well an information retrieval system retrieves the relevant documents requested by a user. The measures are defined as follows:

Precision = Total number of documents retrieved that are relevant/total number of documents that are retrieved

Recall = Total number of documents retrieved that are relevant/total number of relevant documents in the database

Precision and Recall, Table 1 The outcomes of classification into positive and negative classes

		Assigned class	
		Positive	Negative
Actual class	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

We can employ the same terminology used in a ▶ confusion matrix to define these two measures. Let relevant documents be positive examples and irrelevant documents, negative examples. The two measures can be redefined with reference to a special case of the confusion matrix, with two classes: one designated the *positive* class and the other the *negative* class, as indicated in Table 1.

Precision = True positives/total number of positives predicted = TP/(TP + FP)

Recall = True positives/total number of actual positives = TP/(TP + FN)

Instead of two measures, they are often combined to provide a single measure of retrieval performance called the ▶ F-measure as follows:

F-measure = 2 * recall * precision/(recall + precision)

Cross-References

▶ Confusion Matrix

Predicate

A *predicate* or predicate symbol is used in logic to denote properties and relationships. Formally, if P is a predicate with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms (i.e., constants, variables, or compound terms built from function symbols), then $P(t_1, \ldots, t_n)$ is an atomic formula or *atom*. Such an atom represents a statement that can be either true or false. Using logical connectives, atoms can be combined to build well-formed formulae in ▶ first-order logic or ▶ clauses in ▶ logic programs.

Cross-References

▶ Clause
▶ First-Order Logic
▶ Logic Program

Predicate Calculus

▶ First-Order Logic

Predicate Invention

Definition

Predicate invention is used in ▶ inductive logic programming to refer to the automatic introduction of new relations or predicates in the hypothesis language. Inventing relevant new predicates is one of the hardest tasks in machine learning, because there are so many possible ways to introduce such predicates and because it is hard to judge their quality. As an example, consider a situation where in the predicates `fatherof` and `motherof` are known. Then it would make sense to introduce a new predicate that is true whenever `fatherof` or `motherof` is true. The new predicate that would be introduced this way corresponds to the `parentof` predicate. Predicate invention has been introduced in the context of inverse resolution.

Cross-References

▶ Inductive Logic Programming
▶ Logic of Generality

Predicate Logic

▶ First-Order Logic

Prediction with Expert Advice

▶ Online Learning

Predictive Software Models

▶ Predictive Techniques in Software Engineering

Predictive Techniques in Software Engineering

Jelber Sayyad Shirabad
University of Ottawa, Ottawa, ON, Canada

Synonyms

Predictive software models

Introduction

Software engineering (SE) is a knowledge- and decision-intensive activity. From the initial stages of the software life cycle (i.e., requirement analysis), to the later stage of testing the system, and finally maintaining the software through its operational life, decisions need to be made which impact both its success and failure. For instance, during project planning one needs to be able to forecast or predict the required resources to build the system. At the later stages such as testing or maintenance it is desirable to know which parts of the system may be impacted by a change, or are more risky or will require more intensive testing.

The process of developing software can potentially create a large amount of data and domain knowledge. The nature of the data, of course, depends on the phase in which the data were generated. During the requirement analysis, this data most times is manifested in the form of documentations. As the process moves forward, other types of artifacts such as code and test cases are generated. However, what, when, how accurately, and how much is recorded varies from one organization to the next. More mature organizations have a tendency to maintain larger amount of data about the software systems they develop.

The data generated as part of the software engineering process captures a wide range of latent knowledge about the system. Having such a source of information, the question one needs to ask is that whether there is any technology that can leverage this potentially vast amount of data to:

- Better understand a system
- Make more informative decisions as needed through the life of an existing system
- Apply lessons learned from building other systems to the creation of a new system

As this chapter will show, machine learning (ML), which provides us with a host of algorithms and techniques to learn from data, is such a technology. In preparing this entry we have drawn from over two decades of research in applying ML to various software engineering problems. The number of potential uses of ML in SE is practically enormous and the list of applications is expanding over time. The focus of this chapter is a subset of these applications, namely the ones that aim to create models for the purpose of making a prediction regarding some aspect of a software system. One could dedicate a separate article for some of these prediction tasks, as there is a large body of research covering different aspects of interest, such as algorithms, estimation methods, features used, and the like. However, due to space constraints, we will only mention a few representative research examples. The more general topic of the application of ML in SE can be studied from different points of view. A good discussion of many such aspects and applications can be found in Zhang and Tsai (2003).

Traditionally, regression-based techniques have been used in software engineering for building predictive models. However, this requires making a decision as to what kind of regression method should be used (e.g., linear or quadratic), or alternatively what kind of curve should be fit to the data. This means that the general shape of the function is determined first, and then the model is built. Some researcher, have used ML as a way to delegate such decisions to the algorithm. In other words, it is the algorithm that would produce the best fit to the data. Some of the most common replacements in the case of regression problems have been neural networks (NN) and genetic programming (GP). However, obviously the use of such methods still requires other types of decisions, such as the topology of the network, the number of generations, or the probability of mutations to be made by humans. Sometimes, a combination of different methods such as genetic algorithms and neural networks are used, where one method explores possible parameters for the actual method used to build the model.

Software engineering-related datasets, similar to many other real world datasets, are known to contain noise. Another justification for the use of ML in software engineering applications is that it provides algorithms that are less sensitive to noise.

The Process of Applying ML to SE

To apply ML to SE, similar to other applications, one needs to follow certain steps, which include:
Understanding the problem. This is an essential step that heavily influences the decisions to follow. Examples of typical problems in the software engineering domain are the need to be able to estimate the cost or effort involved in developing a software, or to be able to characterize the quality of a software system, or to be able to predict what modules in a system are more likely to have a defect.

Casting the original problem as a learning problem. To use ML technology, one needs to decide on how to formulate the problem as a learning task. For instance, the problem of finding modules that are likely to be faulty can be cast as a classification problem, (e.g., is the module faulty or not) or a numeric prediction problem (e.g., what the estimated fault density of a module is). This mapping is not always straightforward, and may require further refinement of the original problem statement or breaking down the original problem into sub-problems, for some of them ML may provide an appropriate solution.

Collection of data and relevant background knowledge. Once the ML problem for a particular SE application is identified, one needs to collect the necessary data and background knowledge in support of the learning task. In many SE applications data is much more abundant or easier to collect than the domain theory or background knowledge relevant to a particular application. For instance, collecting data regarding faults discovered in a software system and changes applied to the source to correct a fault is a common practice in software projects. On the other hand, there is no comprehensive and agreed upon domain theory describing software systems. Having said that, in the case of some applications, if we limit ourselves to incomplete background knowledge, then it can be automatically generated by choosing a subset that is considered to be relevant. For instance, in Cohen and Devanbu (1999), the authors apply inductive logic programming to the task of predicting faulty modules in a software system. They describe the software system in terms of cohesion and coupling-based relations between classes, which are generated by parsing the source code.

Data preprocessing and encoding. Preprocessing the data includes activities such as reducing the noise, selecting appropriate subsets of the collected data, and determining a proper subset of features that describe the concept to be learned. This cleaner data will be input to a specific algorithm and implementation. Therefore, the data and background knowledge, if any, may need to be described and formatted in a manner that complies with the requirements of the algorithm used.

Applying machine learning and evaluating the results. Running a specific ML algorithm is fairly straightforward. However, one needs to measure the goodness of what is learned. For instance, in the case of classification problems, models are frequently assessed in terms of their accuracy by using methods such as holdout and cross-validation. In case of numeric prediction, other standard measures such as mean magnitude of relative error (MMRE) are commonly used. Additionally, software engineering researchers have sometimes adopted other measures for certain applications. For instance PRED(x), which is percentage of the examples (or samples) with magnitude of relative error (MRE)$\leq x$. According to Pfleeger and Atlee (2003), most managers use PRED(25) to assess cost, effort, and schedule models, and consider the model to function well if the value of PRED(25) is greater than 75 %. As for MMRE, a value of less than 25 % is considered to be good; however, other researchers, such as Boehm, would recommend a value of 10 % or less. Assessing the usefulness of what is learned sometimes requires feedback from domain experts or from end users. If what is learned is determined to be inadequate, one may need to either retry this step by adjusting the parameters of the algorithms used, or reconsider the decisions made in earlier stages and proceed accordingly.

Field testing and deployment. Once what is learned is assessed to be of value, it needs to actually be used by the intended users (e.g., project managers and software engineers). Unfortunately, despite the very large body of research in software engineering in general and use of ML in specific applications in SE, the number of articles discussing the actual use and impact of the research in industry is relatively very small. Very often, the reason for this is the lack of desire to share what the industry considers to be confidential information. However, there are numerous research articles that are based on industrial data, which is an indication of the practical benefits of ML in real-world SE.

Applications of Predictive Models in SE

The development of predictive models is probably the most common application of ML in software engineering. This observation is consistent with findings of previous research (Zhang and Tsai 2003). In this section, we mention some of the predictive models one can learn from software engineering data. Our goal is to provide examples of both well established and newer applications. It should be noted that the terminology used by researchers in the field is not always consistent. As such, one may argue that some of these examples belong to more than one category. For instance, in Fenton and Neil (1999) the authors consider predicting faults as a way of estimating software quality and maintenance effort. The paper could potentially belong to any of the categories of fault, quality, or maintenance effort prediction.

Software Size Prediction

Software size estimation is the process of predicting the size of a software system. As software size is usually an input to models that estimate project cost schedule and planning, an accurate estimation of software size is essential to proper estimation of these dependent factors. Software size can be measured in different ways, most common of which is the number of lines of code (LOC); however, other alternatives, such as function points, which are primarily for effort estimation, also provide means to convert the measure to LOC. There are different methods for software sizing, one of which is the component-based method (CBM). In a study to validate the CBM method, Dolado (2000) compared models generated by multiple ▸ linear regression (MLR) with the ones obtained by neural networks and genetic programming. He concluded that both NN- and GP-based models perform as well or better than the MLR models. One of the cited benefits of NN was its ability to capture non-linear relations, which is one of the weaknesses of MLR, while GP was able to generate models that were interpretable. Regolin et al. (2003) also used NN- and GP-based models to predict software size in terms of LOC. They use both function points and number of components metrics for this task. Pendharkar (2004) uses decision tree regression to predict the size of OO components. The total size of the system can be calculated after the size of its components is determined.

Software Quality Prediction

The ISO 9126 quality standard decomposes quality to functionality, reliability, efficiency, usability, maintainability, and portability factors. Other models such as McCall's, also define quality in terms of factors that are themselves composed of quality criteria. These quality criteria are further associated with measurable attributes called quality metrics, for instance fault or change counts (Fenton and Pfleeger 1998) However, as stated in Fenton and Pfleeger (1998), many software engineers have a narrower view of quality as the lack of software defects. A de facto standard for software quality is fault density. Consequently, it is not surprising to see that in many published articles the problem of predicting the quality of a system is formulated as prediction of faults. To that end, there has been a large body of work over the years that has applied various ML techniques to build models to assess the quality of a system. For instance, Evett and Khoshgoftar (1998) used genetic programming to build models that predict the number of faults expected in each module. Neural networks have appeared in a number of software quality modeling applications such as Khoshgoftaar et al. (1997), which applied the technique to a large industrial system to classify modules as fault-prone or not fault-prone, or Quah and Thwin (2003) who used object-oriented design metrics as features in developing the model. In El Emam et al. (2001) the authors developed fault prediction models for the purpose of identifying high-risk modules. In this study, the authors investigated the effect of various parameter settings on the accuracy of these models. The models were developed using data from a large real-time system. More recently, Xing et al. (2005) used SVMs and Seliya and Khoshgoftaar (2007) used an EM semi-supervised learning algorithm to develop software quality models. Both these works cite the ability of these algorithms to generate models with good performance in the presence of a small amount of labeled data.

Software Cost Prediction

Software cost prediction typically refers to the process of estimating the amount of effort needed to develop a software system. As this definition suggests, cost and effort estimations are often used interchangeably. Various kinds of cost estimations are needed throughout the software life cycle. Early estimation allows one to determine the feasibility of a project. More detailed estimation allows managers to better plan for the project. As there is less information available in the early stages of the project, early predictions have a tendency to be the least accurate. Software cost and effort estimation models are among some of the oldest software process prediction models. There are different methods of estimating costs including:

(1) Expert opinion; (2) analogy based on similarity to other projects; (3) decomposition of the project in terms of components to deliver or tasks to accomplish, and to generate a total estimate from the estimates of the cost of individual components or activities; and (4) the use of estimation models (Fenton and Pfleeger 1998).

In general, organization-specific cost estimation datasets tend to be small, as many organizations deal with a limited number of projects and do not systematically collect process level data, including the actual time and effort expenditure for completion of a project. As cost estimation models are numeric predictors, many of the original modeling techniques were based on regression methods.

The study in Briand et al. (1999) aims to identify methods that generate more accurate cost models, as well as to investigate the effects of the use of organization-specific versus multi-organization datasets. The authors compared the accuracy of models generated by using ordinary least squares regression, stepwise ANOVA, CART, and analogy. The measures used were MMRE, median of MRE (MdMRE), and PRED(25). While their results did not show a statistical difference between models obtained from these methods, they suggest that CART models are of particular interest due to their simplicity of use and interpretation.

Shepperd and Schofield (1997) describes the use of analogies for effort prediction. In this method, projects are characterized in terms of attributes such as the number of interfaces, the development method, or the size of the functional requirements document. The prediction for a specific project is made based on the characteristics of projects most similar to it. The similarity measure used in Shepperd and Schofield (1997) is Euclidean distance in n-dimensional space of project features. The proposed method was validated on nine different industrial datasets, covering a total of 275 projects. In all cases, the analogy-based method outperforms algorithmic models based upon stepwise regression when measured in terms of MMRE. When results are compared using PRED(25) the analogy-based method generates more accurate models in seven out of nine datasets. Decision tree and neural network-based models are also used in a number of studies on effort estimation models.

In a more recent paper, (Oliveira 2006), a comparative study of support vector regression (SVR), radial basis function ▶ neural networks (RBFNs), and ▶ linear regression-based models for estimation of a software project effort is presented. Both linear as well as RBF kernels were used in the construction of SVR models. Experiments using a dataset of software projects from NASA showed that SVR significantly outperforms RBFNs and linear regression in this task.

Software Defect Prediction

In research literature one comes across different definitions for what constitutes a defect: fault and failure. According to Fenton and Pfleeger (1998) a fault is a mistake in some software product due to a human error. Failure, on the other hand, is the departure of the system from its required behavior. Very often, defects refer to faults and failures collectively. In their study of defect prediction models, Fenton and Neil observed that, depending on the study, defect count could refer to a post-release defect, the total number of known defects, or defects that are discovered after some arbitrary point in the life cycle. Additionally, they note that defect rate, defect density, and failure rate are used almost interchangeably in the literature (Fenton and Neil 1999). The lack of an agreed-upon definition for such a fundamental measure makes comparison of the models or published results in the literature difficult. Two major reasons cited in research literature for developing defect detection models are assessing software quality and focusing testing or other needed resources on modules that are more likely to be defective. As a result, we frequently find ourselves in a situation where a model could be considered both a quality prediction model and a defect prediction model. Therefore, most of the publications we have mentioned under software quality prediction could also be referred to in this subsection. Fenton and Neil suggest using Bayesian Belief Networks as an alternative to other existing methods (Fenton and Neil 1999).

Software Reliability Prediction

The ANSI Software Reliability Standard defines software reliability as:

> "the probability of failure-free operation of a computer program for a specified time in a specified environment."

Software reliability is an important attribute of software quality. There are a number of publications on the use of various neural network-based reliability prediction models, including Sitte (1999) where NN-based software reliability growth models are compared with models obtained through recalibration of parametric models. Results show that neural networks are not only much simpler to use than the recalibration method, but that they are equal or better trend predictors. In Pai and Hong (2006) the authors use SVMs to predict software reliability. They use simulated annealing to select the parameters of the SVM model. Results show that an SVM-based model with simulated annealing performs better than existing Bayesian models.

Software Reusability Prediction

The use of existing software artifacts or software knowledge is known as software reuse. The aim of software reuse is to increase the productivity of software developers, and increase the quality of end product, both of which contribute to overall reduction in software development costs. While the importance of software reuse was recognized as early as 1968 by Douglas McIlroy, applications of ML in predicting reusable components are relatively few and far between. The typical approach is to label the reusable piece of code (i.e., a module or a class) as one of reusable or non-reusable, and to then use software metrics to describe the example of interest. An early work by Esteva (1990) used ID3 to classify Pascal modules from different application domains as either reusable or not-reusable. These modules contained different number of procedures. Later work in Mao et al. (1998) uses models built using C4.5 as a means to verify three hypothesis of correlation between reusability and the quantitative attributes of a piece of software: inheritance, coupling, and complexity. For each hypothesis, a set of relevant metrics (e.g., complexity metrics for a hypothesis on the relation between complexity and reuse) is used to describe examples. Each example is labeled as one of four classes of reusability, ranging from "totally reusable" to "not reusable at all." If the learned model performs well then this is interpreted as the existence of a hypothesized relation between reuse and one of the abovementioned quantitative attributes.

Other Applications

In this section, we discuss some of the more recent uses of ML techniques in building predictive models for software engineering applications that do not fall into one the above widely researched areas.

In Padberg et al. (2004) models are learned to predict the defect content of documents after software inspection. Being able to estimate how many defects are in a software document (e.g., specifications, designs) after the inspection, allows managers to decide whether to re-inspect the document to find more defects or pass it on to the next development step. To capture the non-linear relation between the inspection process metrics, such as total number of defects found by the inspection team and the number of defects in the document, the authors train a neural network. They conclude that these models yield much more accurate estimates than standard estimation methods such as capture-recapture and detection profile.

Predicting the stability of object-oriented software, defined as the ease by which a software system or component can be changed while maintaining its design, is the subject of research in Grosser et al. (2002). More specifically, stability is defined as preservation of the class interfaces through evolution of the software. To accomplish the above task, the authors use Cased-Base Reasoning. A class is considered stable if its public interface in revision J is included in revision $J + 1$. Each program class or case is represented by structural software metrics, which belong to one of the four categories of coupling, cohesion, inheritance, and complexity.

Models that predict which defects will be escalated are developed in Ling et al. (2006). Escalated defects are the ones that were not addressed prior to release of the software due to factors such as deadlines and limited resources. However, after the release, these defects are escalated by the customer and must be immediately resolved by the vendor at a very high cost. Therefore, the ability to predict the risk of escalation for existing defect reports will prevent many escalations, and result in large savings for the vendor. The authors in this paper show how the problem of maximizing net profit (the difference in cost of following predictions made by the escalation prediction model versus the existing alternative policy) can be converted to cost-sensitive learning. The assumption here is that net profit can be represented as a linear combination of true positive, false positive, true negative, and false negative prediction counts, as is done for cost-sensitive learning that attempts to minimize the weighted cost of the abovementioned four factors. The results of the experiments performed by the authors show that an improved version of the CSTree algorithm can produce comprehensible models that generate a large positive unit net profit.

Most predictive models developed for software engineering applications, including the ones cited in this article, make prediction regarding a single entity – for instance, whether a module is defective, how much effort is needed to develop a system, is a piece of code reusable, and so on. Sayyad Shirabad et al. (2007) introduced the notion of relevance relations among multiple entities in software systems. As an example of such relations, the authors applied historic problem report and software change data to learned models for the Co-update relation among files in a large industrial telecom system. These models predict whether changing one source file may require a change in another file. Different sets of attributes, including syntax-based software metrics as well as textual attributes such as source file comments and problem reports, are used to describe examples of the Co-update relation. The C5.0 decision tree induction algorithm was used to learn these predictive models. The authors concluded that text-based attributes outperform syntactic attributes in this model-building task. The best results are obtained for text-based attributes extracted from problem reports. Additionally, when these attributes are combined with syntactic attributes, the resulting models perform slightly better.

Future Directions

As we mentioned earlier due to its decision-intensive nature, there is potential for learning a large number of predictive models for software engineering tasks. A very rich area of research for future applications of predictive models in software engineering is in *Autonomic Computing*. Autonomic computing systems, as was put forward in Ganek and Corbi (2003), should be:

- *Self-configuring*: able to adapt to changes in the system in a dynamic fashion.
- *Self-optimizing*: able to improve performance and maximize resource allocation and utilization to meet end users' needs while minimizing human intervention.
- *Self-healing*: able to recover from mistakes by detecting improper operations proactively or reactively and then initiate actions to remedy the problem without disrupting system applications.
- *Self-protecting*: able to anticipate and take actions against intrusive behaviors as they occur, so as to make the systems less vulnerable to unauthorized access.

Execution of actions in support of the capabilities mentioned above follows the detection of a triggering change of state in the environment. In some scenarios, this may entail a prediction about the current state of the system; in other scenarios, the prediction may be about the future state of the system. In a two-state scenario, the system needs to know whether it is in a normal or abnormal (undesired) state. Examples of undesired states are *needs optimization* or *needs healing*. The detection of the state of a system can be cast as a classification problem. The decision as to what attributes should be used to represent each example of a normal or an abnormal state depends on the specific prediction model that we would like to build and on the monitoring capabilities of the system. Selecting the best attributes among a set of potential attributes will require empirical analysis. However, the process can be further aided by:

- *Expert knowledge*: Based on their past experience, hardware and software experts typically have evidence or reasons to believe that some attributes are better indicators of desired or undesired states of the system.
- *Documentation*: System specification and other documents sometimes include the range of acceptable values for certain parameters of the system. These parameters could be used as attributes.
- *Feature selection*: This aims to find a subset of available features or attributes that result in improving a predefined measure of goodness, such as the accuracy of the model. Reducing the number of features may also result in a simpler model. One of the benefits of such simpler models is the higher prediction speed, which is essential for timely responses by the autonomic system to changes in the environment.

Obviously, given enough examples of different system states, one can build multi-class models, which can make finer predictions regarding the state of the system.

In the context of autonomic computing, besides classification models, numeric predictors can also be used for resource estimation (e.g., what is the appropriate database cache size considering the current state of the system). Furthermore, an autonomic system can leverage the ability to predict the future value of a variable of interest, such as the use of a particular resource based on its past values. This can be accomplished through ▶ time series predictions. Although researchers have used neural networks and support vector machines for time series prediction in various domains, we are not aware of an example of the usage of such algorithms in autonomic computing.

Recommended Reading

Briand L, El Emam K, Surmann D, Wieczorek I (1999) An assessment and comparison of common software cost estimation modeling techniques. In: Proceedings of 21st international conference on software engineering, Los Angeles, pp 313–322

Cohen W, Devanbu P (1999) Automatically exploring hypotheses about fault prediction: a comparative study of inductive logic programming methods. Int J Softw Eng Knowl Eng 9(5):519–546

Dolado JJ (2000) A validation of the component-based method for software size estimation. IEEE Trans Softw Eng 26(10):1006–1021

El Emam K, Benlarbi S, Goel N, Rai S (2001) Comparing case-based reasoning classifiers for predicting high risk software components. J Syst Softw 55(3): 301–320

Esteva JC (1990) Learning to recognize reusable software modules using an inductive classification system. In: Proceedings of the fifth Jerusalem conference on information technology, Jerusalem, pp 278–285

Evett M, Khoshgoftar T (1998) GP-based software quality prediction. In: Proceedings of the third annual conference on genetic programming, pp 60–65

Fenton N, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689

Fenton NE, Pfleeger SL (1998) Software metrics: a rigorous and practical approach, 2nd edn. PWS, Boston

Ganek AG, Corbi TA (2003) The dawning of autonomic computing era. IBM Syst J 42(1):5–18

Grosser D, Sahraoui HA, Valtchev P (2002) Predicting software stability using case-based reasoning. In: Proceedings of 17th IEEE international conference on automated software engineering (ASE), Edinburgh, pp 295–298

Khoshgoftaar T, Allen E, Hudepohl J, Aud S (1997) Applications of neural networks to software quality modeling of a very large telecommunications system. IEEE Trans Neural Netw 8(4):902–909

Ling C, Sheng V, Bruckhaus T, Madhavji N (2006) Maximum profit mining and its application in software development. In: Proceedings of the 12th ACM international conference on knowledge discovery and data mining (SIGKDD), Philadelphia, pp 929–934

Mao Y, Sahraoui H, Lounis H (1998) Reusability hypothesis verification using machine learning techniques: a case study. In: Proceedings of the 13th IEEE international conference on automated software engineering, Honolulu, pp 84–93

Oliveira A (2006) Estimation of software project effort with support vector regression. Neurocomputing 69(13–15):1749–1753

Padberg F, Ragg T, Schoknecht R (2004) Using machine learning for estimating the defect content after an inspection. IEEE Trans Softw Eng 30(1):17–28

Pai PF, Hong WC (2006) Software reliability forecasting by support vector machines with simulated annealing algorithms. J Syst Softw 79(6):747–755

Pendharkar PC (2004) An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models. Inf Manag 42(1):61–73

Pfleeger SL, Atlee JM (2003) Software engineering: theory and practice. Prentice-Hall, Upper Saddle River

Quah TS, Thwin MMT (2003) Application of neural networks for software quality prediction using object-oriented metrics. In: Proceedings of international conference on software maintenance, Amsterdam, pp 22–26

Regolin EN, de Souza GA, Pozo ART, Vergilio SR (2003) Exploring machine learning techniques for software size estimation. In: Proceedings of the 23rd international conference of the Chilean computer science society (SCCC), Chillan, pp 130–136

Sayyad Shirabad J, Lethbridge TC, Matwin S (2007) Modeling relevance relations using machine learning techniques. In: Tsai J, Zhang D (eds) Advances in machine learning applications in software engineering. IGI, pp 168–207

Seliya N, Khoshgoftaar TM (2007) Software quality estimation with limited fault data: a semi-supervised learning perspective. Softw Qual J 15(3):327–344

Shepperd M, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng 23(11):736–743

Sitte R (1999) Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration. IEEE Trans Reliab 48(3):285–291

Xing F, Guo P, Lyu MR (2005) A novel method for early software quality prediction based on support vector machine. In: Proceedings of IEEE international conference on software reliability engineering, Chicago, pp 213–222
Zhang Du, Tsai JP (2003) Machine learning and software engineering. Softw Qual J 11(2):87–119

Preference Learning

Johannes Fürnkranz[1,3] and Eyke Hüllermeier[2]
[1]Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
[2]Department of Computer Science, Paderborn University, Paderborn, Germany
[3]Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Preference learning refers to the task of learning to predict (contextualized) preferences on a collection of alternatives, which are often represented in the form of an order relation, on the basis of observed or revealed preference information. Supervision in preference learning is typically weak, in the sense that only partial information about sought structures or indirect information about an underlying value function are provided; a common example is feedback in the form of pairwise comparisons between alternatives. Especially important in preference learning are ranking problems, in which preferences are represented in terms of total or partial order relations. Such problems can be approached in two fundamentally different ways, either by learning binary preferences on pairs of alternatives or by inducing an underlying (latent) value function on single alternatives.

Synonyms

Comparison training; Constraint classification; Learning from preferences

Motivation and Background

Preference information plays a key role in automated decision-making and appears in various guises in artificial intelligence (AI) research (Domshlak et al. 2011). In particular, the formal modeling of preferences can be considered an essential aspect of autonomous agent design. Yet, in spite of the existence of formalisms for representing preferences in a compact way, such as CP-networks (Boutilier et al. 2004), modeling preferences by hand is a difficult task. This is an important motivation for preference *learning*, which is meant to support and partly automatize the design of preference models. Roughly speaking, preference learning is concerned with the automated acquisition of preference models from observed or revealed preference information, that is, data from which (possibly uncertain) preference representations can be deduced in a direct or indirect way.

Computerized methods for revealing the preferences of individuals (users) are useful not only in AI but also in many other fields showing a tendency for *personalization* of products and services, such as computational advertising, e-commerce, and information retrieval, where such techniques are also known as ▸ learning to rank (Liu 2011). Correspondingly, a number of methods and tools have been proposed with the goal of leveraging the manifold information that users provide about their preferences, either explicitly via ratings, written reviews, etc. or implicitly via their behavior (shopping decisions, websites visited, and so on). Typical examples include ▸ recommender systems and ▸ collaborative filtering, which can be viewed as special cases of preference learning. A first attempt at setting a common framework for this emerging subfield of machine learning was made by Fürnkranz and Hüllermeier (2010).

Ranking is one of the key tasks in the realm of preference learning. One can distinguish between two important types of ranking problems, namely, learning from object and learning from label preferences. A ranking is a special type of preference structure, namely, a *strict total order*, that is, a binary relation $\succ$ on a set $\mathcal{A}$ of alternatives that is total, irreflexive, and transitive. In agreement with our preference semantics, $a \succ b$ suggests that alternative a is preferred to alternative b. However, in a wider sense, the term "preference" can simply be interpreted as any kind of order relation. For example, $a \succ b$ can also mean that a is an algorithm that outperforms b on a certain problem or that a is a student finishing her studies before another student b.

Structure of the Learning System

An important difference between object and label ranking concerns the formal representation of the preference context and the alternatives to be ordered. In object ranking, the alternatives themselves are characterized by properties, typically in terms of a feature vector (attribute-value representation). Thus, the learner has the possibility to generalize via properties of the alternatives, whence a ranking model can be applied to arbitrary sets of such alternatives. In label ranking, the alternatives to be ranked are labels as in classification learning, i.e., mere identifiers without associated properties. Instead, the ranking context is characterized in terms of an instance from a given instance space, and the task of the model is to rank alternatives depending on properties of the context. Thus, the context may change (as opposed to object ranking, where it is implicitly fixed), but the objects to be ranked remain the same. Stated differently, object ranking is the problem to rank varying sets of objects under invariant preferences, whereas label ranking is the problem to rank an invariant set of objects under varying preferences.

Both problems can be approached in two principal ways, either by learning a *value function* that induces the sought ranking by *evaluating individual alternatives* or by comparing pairs of alternatives, that is, learning a *binary preference relation*. Note that the first approach implicitly assumes an underlying total order relation, since numerical (or at least totally ordered) utility scores enforce the comparability of alternatives. The second approach is more general in this regard, as it also allows for partial order relations. On the other hand, this approach may lead to additional complications, since a set of *hypothetical* binary preferences induced from empirical data may exhibit inconsistencies in the form of preferential cycles.

Learning from Object Preferences

The most frequently studied problem in learning from preferences is to induce a *ranking function* $r(\cdot)$ that is able to order any (finite) subset $\mathcal{O}$ of an underlying (possibly infinite) class $\mathcal{X}$ of objects. That is, $r(\cdot)$ assumes as input a subset $\mathcal{O} \subseteq \mathcal{X}$ of objects and returns as output a permutation τ of $\{1, \ldots, |\mathcal{O}|\}$. The interpretation of this permutation is that, for objects $x_i, x_j \in \mathcal{O}$, the former is preferred to the latter whenever $\tau(i) < \tau(j)$. The objects themselves are typically characterized by a finite set of features as in conventional attribute-value learning. The training data consists of a set of exemplary pairwise preferences $x \succ x'$ with $x, x' \in \mathcal{X}$. A survey of object ranking approaches is given by Kamishima et al. (2010).

Note that, in order to evaluate the predictive performance of a ranking algorithm, an accuracy measure (or loss function) is needed that compares a predicted ranking with a given reference ranking. To this end, one can refer, for example, to statistical measures of ▸ rank correlation. Expected or empirical loss minimization is a difficult problem for measures of that kind, especially because they are not (instance-wise) decomposable.

Many ▸ learning to rank problems may be viewed as object ranking problems. For example, Joachims (2002) studies a scenario where the training information could be provided implicitly by the user who clicks on some of the links in a query result and not on others. This information can be turned into binary preferences by assuming a preference of the selected pages over those nearby pages that are not clicked on. Applications in information retrieval typically suggest loss functions that put more emphasis on the top and less on the bottom of a ranking; for this purpose, specific measures have been proposed, such as the (normalized) discounted cumulative gain (Liu 2011).

Learning from Label Preferences

In label ranking, preferences are contextualized by elements x of an instance space $\mathcal{X}$, and the goal is to learn a ranking function $\mathcal{X} \longrightarrow \mathcal{S}_m$ for a fixed $m \geq 2$. Thus, for any instance $x \in \mathcal{X}$ (e.g., a person), a prediction in the form of an associated ranking $\succ_x$ of a finite set $\mathcal{L} = \{\lambda_1, \ldots, \lambda_m\}$ of labels or alternatives is sought, where $\lambda_i \succ_x \lambda_j$ means that instance x prefers λ_i to λ_j. Again, the quality of a prediction of that kind is typically captured in terms of a rank correlation measure (or an associated loss function). The training information consists of a set of instances for which (partial) knowledge about the associated preference relation is available. More precisely, each training instance x is associated with a subset of all pairwise preferences. Thus, despite the assumption of an underlying ("true") target ranking, the training data is not expected to provide full information about such rankings (and may even contain inconsistencies, such as pairwise preferences that are conflicting due to observation errors).

The above formulation essentially follows Fürnkranz and Hüllermeier (2010), though similar formalizations have been proposed independently by several authors; for an overview, see the survey papers by Vembu and Gärtner (2010) and Zhou et al. (2014). Label ranking contributes to the general trend of extending machine learning methods to complex and structured output spaces (Tsochantaridis et al. 2005). Moreover, label ranking can be viewed as a generalization of several standard learning problems. In particular, the following well-known problems are special cases of learning label preferences: (i) ▸ classification, where a single class label λ is assigned to each instance x; this is equivalent to the set of preferences $\{\lambda \succ_x \lambda_j \,|\, \lambda_j \in \mathcal{L} \setminus \{\lambda\}\}$, and (ii) ▸ multi-label classification, where each training example x is associated with a subset $L \subseteq \mathcal{L}$ of possible labels. This is equivalent to the set of preferences $\{\lambda_i \succ_x \lambda_j \,|\, \lambda_i \in L, \lambda_j \in \mathcal{L} \setminus L\}$. In each of the former scenarios, the sought prediction can be obtained by post-processing the output of a ranking model $f : \mathcal{X} \longrightarrow \mathcal{S}_m$ in a suitable way. For example, in multi-class classification, where only a single label is requested, it suffices to project a label ranking to the top-ranked label.

Applications of this general framework can be found in various fields, for example, in marketing research; here, one might be interested in discovering dependencies between properties of clients and their preferences for products. Another application scenario is ▸ meta-learning, where the task is to rank learning algorithms according to their suitability for a new dataset, based on the characteristics of this dataset (Schäfer and Hüllermeier 2015). Moreover, every preference statement in the well-known CP-nets approach (Boutilier et al. 2004), a qualitative graphical representation that reflects conditional dependence and independence of preferences under a ceteris paribus interpretation, formally corresponds to a label ranking function that orders the values of a certain attribute depending on the values of the parents of this attribute (predecessors in the graph representation).

Other Settings

A number of variants of the above ranking problems have been proposed and studied in the literature. For example, a setting referred to as *instance ranking* is very similar to object ranking. However, instead of relative (pairwise) comparisons, training data consists of absolute ratings of alternatives; typically these ratings are taken from an ordinal scale, such as 1 to 5 stars. Moreover, a predicted ranking is not compared with another (ground-truth) ranking but with the multi-partition induced by the rating of the alternatives (Fürnkranz et al. 2009).

Attempts have also been made at combining object and label ranking, that is, to exploit feature representations of both the preference context and the alternatives to be ranked. One approach is to combine both pieces of information by means of a *joint feature map* $\phi : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathcal{Z}$ and to learn a value function $f : \mathcal{Z} \longrightarrow \mathbb{R}$; here, $\mathcal{Y}$ is a parametric or structured space of alternatives and $\mathcal{Z} \subseteq \mathbb{R}^d$ a joint feature space (Tsochantaridis et al. 2005; Schäfer and Hüllermeier 2015).

Learning Utility Functions

Evaluating alternatives in terms of a value or utility function is a very natural way of representing preferences, which has a long tradition in economics and decision theory (Fishburn 1969). In the object preferences scenario, such a function is a mapping $f : \mathcal{X} \longrightarrow \mathbb{R}$ that assigns a utility degree $f(x)$ to each object x and, thereby, induces a linear order on $\mathcal{X}$. In the label preferences scenario, a utility function $f_i : \mathcal{X} \longrightarrow \mathcal{U}$ is needed for every label λ_i, $i = 1, \ldots, m$. Here, $f_i(x)$ is the utility assigned to alternative λ_i in the context x. To obtain a ranking for x, the alternatives are ordered according to their utility scores, i.e., a ranking $\succ_x$ is derived such that $\lambda_i \succ_x \lambda_j$ implies $f_i(x) \geq f_j(x)$.

If the training data offers the utility scores directly, preference learning essentially reduces to a standard ▸ regression or an ordinal regression problem, depending on the underlying utility scale. This information can rarely be assumed, however. Instead, usually only constraints derived from comparative preference information of the form "this alternative should have a higher utility score than that alternative" are given. Thus, the challenge for the learner is to find a value function that is as much as possible in agreement with a set of such constraints.

For object ranking approaches, this idea has first been formalized by Tesauro (1989) under the name *comparison training*. He proposed a symmetric neural-network architecture that can be trained with representations of two states and a training signal that indicates which of the two states is preferable. The elegance of this approach comes from the property that one can replace the two symmetric components of the network with a single network, which can subsequently provide a real-valued evaluation of single states. Similar ideas have also been investigated for training other types of classifiers, in particular support vector machines. We already mentioned Joachims (2002) who analyzed "click-through data" in order to rank documents retrieved by a search engine according to their relevance. Earlier, Herbrich et al. (1998) proposed an algorithm for training SVMs from pairwise preference relations between objects.

For the case of label ranking, a corresponding method for learning the functions $f_i(\cdot)$, $i = 1, \ldots, m$, from training data has been proposed in the framework of *constraint classification*, which allows for reducing a label ranking to a single binary classification problem (Har-Peled et al. 2002). The learning method proposed in this work constructs two training examples, a positive and a negative one, for each given preference $\lambda_i \succ_x \lambda_j$, where the original N-dimensional training example (feature vector) x is mapped into an $(m \times N)$-dimensional space. In this space, the learner finds a linear model (hyperplane) f that separates the positive from the negative examples. Finally, the model f is "split" into m linear value functions $f_1, \ldots, f_m$, one for each label.

Learning Preference Relations

An alternative to learning latent utility functions consists of learning binary preference relations, which essentially amounts to reducing preference learning to binary classification. For object ranking, the pairwise approach has been pursued in Cohen et al. (1999). The authors propose to solve object ranking problems by learning a binary preference predicate $Q(x, x')$, which predicts whether x is preferred to x' or vice versa. A final ordering is found in a second phase by deriving a ranking that is maximally consistent with these (possibly conflicting) predictions.

For label ranking, the pairwise approach has been introduced in Hüllermeier et al. (2008) as a natural extension of *pairwise classification*, a well-known ▸ class binarization technique. The idea is to train a separate model (base learner) $\mathcal{M}_{i,j}$ for each pair of labels $(\lambda_i, \lambda_j) \in \mathcal{L}$, $1 \leq i < j \leq m$; thus, a total number of $m(m-1)/2$ models are needed. For training, a preference information of the form $\lambda_i \succ_x \lambda_j$ is turned into a (classification) example (x, y) for the learner $\mathcal{M}_{a,b}$, where $a = \min(i, j)$ and $b = \max(i, j)$. Moreover, $y = 1$ if $i < j$ and $y = 0$ otherwise. Thus, $\mathcal{M}_{a,b}$ is intended to learn the mapping that outputs 1 if $\lambda_a \succ_x \lambda_b$ and 0 if $\lambda_b \succ_x \lambda_a$. This mapping can be realized by any binary classifier. Instead of a $\{0, 1\}$-valued classifier, one can of course also employ a scoring classifier. For example, the output of a probabilistic classifier would be a number in the unit interval $[0, 1]$ that can be interpreted as a probability of the preference $\lambda_a \succ_x \lambda_b$.

At classification time, a query $x_0 \in \mathcal{X}$ is submitted to the complete ensemble of binary learners. Thus, a collection of predicted pairwise preference degrees $\mathcal{M}_{i,j}(x)$, $1 \le i, j \le m$, is obtained. The problem, then, is to turn these pairwise preferences into a ranking of the label set $\mathcal{L}$. To this end, different ranking procedures can be used. The simplest approach is to extend the (weighted) voting procedure that is often applied in pairwise classification: For each label λ_i, a score

$$S_i = \sum_{1 \le j \ne i \le m} \mathcal{M}_{i,j}(x_0)$$

is derived (where $\mathcal{M}_{i,j}(x_0) = 1 - \mathcal{M}_{j,i}(x_0)$ for $i > j$), and then the labels are ordered according to these scores. Despite its simplicity, this ranking procedure has several appealing properties. Apart from its computational efficiency, it turned out to be relatively robust in practice, and, moreover, it possesses some provable optimality properties in the case where Spearman's rank correlation is used as an underlying accuracy measure. Roughly speaking, if the binary learners are unbiased probabilistic classifiers, the simple "ranking by weighted voting" procedure yields a label ranking that maximizes the expected Spearman rank correlation (Hüllermeier and Fürnkranz 2010). Finally, it is worth mentioning that, by changing the ranking procedure, the pairwise approach can also be adjusted to accuracy measures other than Spearman's rank correlation.

Other Approaches

Referring to the type of training data and the loss function to be minimized on this data, learning value functions and learning preference relations are sometimes called the "pointwise" and "pairwise" approach to preference learning, respectively. This is distinguished from the "listwise" approach, in which a loss is defined on a predicted ranking directly. This can be done, for example, on the basis of probabilistic models of ranking data, such as the Plackett-Luce model. The idea, then, is to learn the parameters of a probabilistic model using statistical methods such as maximum likelihood estimation (or, equivalently, minimizing logarithmic loss). Methods of this kind have been proposed both for object ranking (Cao et al. 2007) and label ranking (Cheng et al. 2010).

Yet another alternative is to resort to the idea of local estimation techniques as prominently represented, for example, by the ▸ nearest neighbor estimation principle: Considering the rankings observed in similar situations as representative, a ranking for the current situation is estimated on the basis of these neighbor rankings, namely, by finding a suitable consensus among them; essentially, this is a problem of rank aggregation (Cheng et al. 2009).

Future Directions

As already said, preference learning is an emerging branch of machine learning and still developing quite dynamically. In particular, new settings or variants of existing frameworks will certainly be proposed and studied in the future. As for ranking problems, for example, an obvious idea and reasonable extension is to go beyond strict total order relations and instead allow for *incomparability* or *indifference* between alternatives and for representing uncertainty about predicted relations (Cheng et al. 2012). Another interesting direction is to combine preference learning with ▸ online learning, i.e., to predict preferences in an online setting. First steps in the direction of online preference learning have recently been made with a preference-based variant of the ▸ multiarmed bandit problem (Busa-Fekete and Hüllermeier 2014).

Cross-References

▸ Class Binarization
▸ Classification
▸ Metalearning

▶ Multi-armed bandit
▶ Online Learning
▶ Rank Correlation
▶ Regression

Recommended Reading

Boutilier C, Brafman R, Domshlak C, Hoos H, Poole D (2004) CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. J AI Res 21:135–191

Busa-Fekete R, Hüllermeier E (2014) A survey of preference-based online learning with bandit algorithms. In: Proceedings of ALT, 25th international conference on algorithmic learning theory, Bled. Springer, pp 18–39

Cao Z, Qin T, Liu TY, Tsai MF, Li H (2007) Learning to rank: from pairwise approach to listwise approach. In: Proceedings of ICML, 24th international conference on machine learning, pp 129–136

Cheng W, Hühn J, Hüllermeier E (2009) Decision tree and instance-based learning for label ranking. In: Proceedings of ICML–2009, 26th international conference on machine learning, Montreal, pp 161–168

Cheng W, Dembczynski K, Hüllermeier E (2010) Label ranking based on the Plackett-Luce model. In: Proceedings of ICML–2010, international conference on machine learning, Haifa, pp 215–222

Cheng W, Hüllermeier E, Waegeman W, Welker V (2012) Label ranking with partial abstention based on thresholded probabilistic models. In: Proceedings of NIPS–2012, 26th annual conference on neural information processing systems, Lake Tahoe

Cohen WW, Schapire RE, Singer Y (1999) Learning to order things. J Artif Intell Res 10(1):243–270

Domshlak C, Hüllermeier E, Kaci S, Prade H (2011) Preferences in AI: an overview. Artif Intell 175(7–8):1037–1052

Fishburn PC (1969) Utility-theory for decision making. Wiley, New York

Fürnkranz J, Hüllermeier E (eds) (2010) Preference learning. Springer, Heidelberg/New York

Fürnkranz J, Hüllermeier E (2010) Preference learning: an introduction. In: Preference learning. Springer, Heidelberg/New York, pp 1–18

Fürnkranz J, Hüllermeier E, Vanderlooy S (2009) Binary decomposition methods for multipartite ranking. In: Proceedings of ECML/PKDD–2009, European conference on machine learning and knowledge discovery in databases, Bled

Har-Peled S, Roth D, Zimak D (2002) Constraint classification: a new approach to multiclass classification. In: Proceedings of 13th international conference on algorithmic learning theory, Lübeck. Springer, pp 365–379

Herbrich R, Graepel T, Bollmann-Sdorra P, Obermayer K (1998) Supervised learning of preference relations. In: Proceedings des Fachgruppentreffens Maschinelles Lernen (FGML-98), pp 43–47

Hüllermeier E, Fürnkranz J (2010) On predictive accuracy and risk minimization in pairwise label ranking. J Comput Syst Sci 76(1):49–62

Hüllermeier E, Fürnkranz J, Cheng W, Brinker K (2008) Label ranking by learning pairwise preferences. Artif Intell 172:1897–1917

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of KDD–02, 8th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, pp 133–142

Kamishima T, Kazawa H, Akaho S (2010) A survey and empirical comparison of object ranking methods. In: Fürnkranz J, Hüllermeier E (eds) Preference learning. Springer, Heidelberg/New York, pp 181–202

Liu TY (2011) Learning to rank for information retrieval. Springer, Berlin/Heidelberg/New York

Schäfer D, Hüllermeier E (2015) Dyad ranking using a bilinear Plackett-Luce model. In: Proceedings of ECML/PKDD–2015, European conference on machine learning and knowledge discovery in databases, Porto

Tesauro G (1989) Connectionist learning of expert preferences by comparison training. In: Advances in neural information processing systems 1 (NIPS-88). Morgan Kaufmann, pp 99–106

Tsochantaridis I, Joachims T, Hofmann T, Altun Y (2005) Large margin methods for structured and interdependent output variables. J Mach Learn Res 6:1453–1484

Vembu S, Gärtner T (2010) Label ranking: a survey. In: Fürnkranz J, Hüllermeier E (eds) Preference learning. Springer, Heidelberg/New York

Zhou Y, Lui Y, Yang J, He X, Liu L (2014) A taxonomy of label ranking algorithms. J Comput 9(3):557

Pre-pruning

Synonyms

Stopping criteria

Definition

Pre-pruning is a ▶ Pruning mechanism that monitors the learning process and prevents further refinements if the current hypothesis becomes too complex.

Cross-References

▶ Overfitting
▶ Post-pruning
▶ Pruning

Presynaptic Neuron

The neuron that sends signals across a synaptic connection. A chemical synaptic connection between two neurons allows to transmit signals from a presynaptic neuron to a postsynaptic neuron.

Principal Component Analysis

Synonyms

PCA

Definition

Principal Component Analysis (PCA) is a ▶ dimensionality reduction technique. It is described in ▶ covariance matrix.

Prior

▶ Prior Probability

Prior Probability

Geoffrey I. Webb
Faculty of Information Technology, Monash University, Victoria, Australia

Synonyms

Prior

Definition

In Bayesian inference, a *prior probability* of a value x of a random variable X, $\text{P}(X = x)$, is the probability of X assuming the value x in the absence of (or before obtaining) any additional information. It contrasts with the ▶ posterior probability, $\text{P}(X = x|Y = y)$, the probability of X assuming the value x in the context of $Y = y$.

For example, it may be that the prevalence of a particular form of cancer, *exoma*, in the population is 0.1 %, so the prior probability of exoma, P(exoma = true), is 0.001. However, assume 50 % of people who have skin discolorations of greater than 1 cm width (sd > 1 cm) have exoma. It follows that the posterior probability of exoma given sd > 1 cm, P(exoma = true | sd > 1 cm = true), is 0.500.

Cross-References

▶ Bayesian Methods

Privacy-Preserving Data Mining

▶ Privacy-Related Aspects and Techniques

Privacy-Related Aspects and Techniques

Stan Matwin
University of Ottawa, Ottawa, ON, Canada
Polish Academy of Sciences, Warsaw, Poland

Synonyms

Privacy-preserving data mining

Definition

The privacy-preserving aspects and techniques of machine learning cover the family of methods and architectures developed to protect the privacy of people whose data are used by machine learning (ML) algorithms. This field, also known as privacy-preserving data mining (PPDM), addresses the issues of data privacy in ML and data mining. Most existing methods and approaches are intended to hide the original data from the learning algorithm, while there is emerging interest in methods ensuring that the learned model does not reveal private information. Another research direction contemplates methods in which several parties bring their data into the model-building process without mutually revealing their own data.

Motivation and Background

The key concept for any discussion of the privacy aspects of data mining is the definition of privacy. After Alan Westin, we understand privacy as the ability "of individuals ... to determine for themselves when, how, and to what extent information about them is communicated to others" (Westin 1967). One of the main societal concerns about modern computing is that the storing, keeping, and processing of massive amounts of data may jeopardize the privacy of individuals whom the data represent. In particular, ML and its power to find patterns and infer new facts from existing data makes it difficult for people to control information about themselves. Moreover, the infrastructure normally put together to conduct large-scale model building (e.g., large data repositories and data warehouses), is conducive to misuse of data. Personal data, amassed in large collections that are easily accessed through databases and often available online to the entire world, become – as phrased by Moor in an apt metaphor (Moor 2004) – "greased." It is difficult for people to control the use of this data.

Theory/Solutions

Basic Dimensions of Privacy Techniques

Privacy-related techniques can be characterized by: (1) the kind of source data modification they perform, e.g., data perturbation, randomization, generalization, and hiding; (2) the ML algorithm that works on the data and how is it modified to meet the privacy requirements imposed on it; and (3) whether the data are centralized or distributed among several parties, and – in the latter case – on what the distribution is based. But even at a more basic level, it is useful to view privacy-related techniques along just two fundamental dimensions.

The first dimension defines what is protected as private – is it the data itself, or the model (the results of data mining)? As we show below, the knowledge of the latter can also lead to identifying and revealing information about individuals. The second dimension defines the protocol of the use of the data: are the data centralized and owned by a single owner, or are the data distributed among multiple parties? In the former case, the owner needs to protect the data from revealing information about individuals represented in the data when that data is being used to build a model by someone else. In the latter case, we assume that the parties have limited trust in each other: they are interested in the results of data mining performed on the union of the data of all the parties, while not trusting the other parties to see their own data without first protecting it against disclosure of information about individuals to other parties.

Moreover, work in PPDM has to apply a framework that is broader than the standard ML methodology. When privacy is an important goal, what matters in performance evaluation is not only the standard ML performance measures, but also some measure of the privacy achieved, as well as some analysis of the robustness of the approach to attacks.

In this article, we structure our discussion of the current work on PPDM in terms of the taxonomy proposed above. This leads to the following bird's-eye view of the field.

Protecting Centralized Data

This subfield emerged in 2000 with the seminal paper by Agrawal and Srikant (2000). They stated the problem as follows: given data in the standard ▶ attribute-value representation, how can an accurate ▸ decision tree be built so that, instead of using original attribute values x_i, the decision tree induction algorithm takes input values $x_i + r$, where r belongs to a certain distribution (Gaussian or uniform). This is a data perturbation technique: the original values are changed beyond recognition, while the distributional properties of the entire data set that decision tree ▸ induction uses remain the same, at least up to a small (empirically, less than 5 %) degradation in accuracy. There is a clear trade-off between the privacy assured by this approach and the quality of the model compared to the model obtained from the original data. This line of research has been continued in Evfimievski et al. (2002) where the approach is extended to association rule mining. As a note of caution about these results, Kargupta et al. (2003) have shown, in 2003, how the randomization approaches are sensitive to attack. They demonstrate how the noise that randomly perturbs the data can be viewed as a random matrix, and that the original data can be accurately estimated from the perturbed data using a spectral filter that exploits some theoretical properties of random matrices.

The simplest and most widely used privacy preservation technique is anonymization of data (also called de-identification). In the context of de-identification, it is useful to distinguish three types of attributes.

Explicit identifiers allow direct linking of an instance to a person (e.g., a cellular phone number or a driver's license number to its holder).

Quasi-identifiers, possibly combined with other attributes, may lead to other data sources capable of unique identification. For instance, Sweeney (2001) shows that the quasi-identifier triplet <date of birth, 5 digit postal code, gender>, combined with the voters' list (publicly available in the USA) uniquely identifies 87 % of the population of the country. As a convincing application of this observation, using quasi-identifiers, Sweeney was able to obtain health records of the governor of Massachusetts from a published dataset of health records of all state employees in which only explicit identifiers have been removed.

Finally, non-identifying attributes are those for which there is no known inference linking to an explicit identifier. Usually performed as part of data preparation, anonymization removes all explicit identifiers from the data.

While anonymization is by far the most common privacy-preserving technique used in practice, it is also the most fallible one. In August 2006, for the benefit of the Web Mining Research community, AOL published 20 million search records (queries and URLs the members had visited) from 658,000 of its members. AOL had performed what it believed was anonymization, in the sense that it removed the names of the members. However, based on the queries – which often contained information that would identify a small set of members or a unique person – it was easy, in many cases, to manually re-identify the AOL member using secondary public knowledge sources. An inquisitive New York Times journalist identified one member and interviewed her.

L. Sweeney is to be credited with sensitizing the privacy community to the fallacy of anonymization: "Shockingly, there remains a common incorrect belief that if the data look anonymous, it is anonymous" (Sweeney 2001). Even if information is de-identified today, future data sources may make re-identification possible. As anonymization is very commonly used prior to model building from medical data, it is interesting that this type of data is prone to specific kinds of re-identification, and therefore anonymization of medical data should be done with particular skill and understanding of the data. Malin (2005) shows how the four main de-identification techniques used in anonymization of genomic data are prone to known, published attacks that can re-identify the data. Moreover, he points out that there will never be certainty about de-identification for quasi-identifiers, as new attributes and data sources that can lead to a linkage to explicitly identifying attributes are constantly being engineered as part of genetics research.

Other perturbation approaches targeting binary data involve changing (flipping) values of selected attributes with a given probability (Du and Zhan 2003; Zhan and Matwin 2004), or replacing the original attribute with a value that is more general in some pre-agreed taxonomy (Iyengar 2002). Generalization approaches often use the concept of k-anonymity: any instance in the database is indistinguishable from other $k-1$ instances (for every row in the database there are $k-1$ identical rows). Finding the least general k-anonymous generalization of a database (i.e., moving the least number of edges upward in a given taxonomy) is an optimization task, known to be *NP*-complete. There are heuristic solutions proposed for it; e.g., Iyengar (2002) uses a ▸ genetic algorithm for this task. Friedman et al. (2006) shows how to build k-anonymity into the decision tree induction. Lately, PPDM researchers have pointed out some weaknesses of the k-anonymity approach. In particular, attacks on data with some properties (e.g., skewed distribution of values of a sensitive attribute, or specific background knowledge) have been described, and techniques to prevent such attacks have been proposed. The notion of p-sensitivity or l-diversity proposed in Machanavajjhala et al. (2007) addresses these weaknesses of k-anonymity by modifying k-anonymity techniques so that the abovementioned attacks do not apply. Furthermore, t-closeness (Ninghui et al. 2007) shows certain shortcomings of these techniques and the resulting potential attacks, and proposes a data perturbation technique which ensures that the distribution of the values of the sensitive attribute in any group resulting from anonymization is close to its distribution in the original table. Some authors, e.g., Domingo-Ferrer et al. (2008), propose the integration of several techniques addressing shortcomings of k-anonymity into a single perturbation technique. The drawback of these solutions is that they decrease the utility of the data more than the standard k-anonymity approaches.

Protecting the Model (Centralized Data)

Is it true that when the data are private, there will be no violation of privacy? The answer is no. In some circumstances, the model may reveal private information about individuals. Atzori et al. (2005) gives an example of such a situation for association rules: suppose the ▸ association rule $a_1 \wedge a_2 \wedge a_3 \Rightarrow a_4$ has support sup = 80, confidence conf = 98.7 %. This rule is 80-anonymous, but considering that

$$\mathrm{sup}(\{a_1, a_2, a_3\}) = \frac{\mathrm{sup}(\{a_1, a_2, a_3, a_4\})}{\mathrm{conf}} = \frac{80}{0.0987} \approx 81.05$$

and given that the pattern $a_1 \wedge a_2 \wedge a_3 \wedge a_4$ holds for 80 individuals, and the pattern $a_1 \wedge a_2 \wedge a_3$ holds for 81 individuals, clearly the pattern $a_1 \wedge a_2 \wedge a_3 \wedge \neg a_4$ holds for just one person. Therefore, the rule unexpectedly reveals private information about a specific person. Atzori et al. (2005) proposes to apply k-anonymity to patterns instead of data, as in the previous section. The authors define inference channels as ▸ itemsets from which it is possible to infer other itemsets that are not k-anonymous, as in the above example. They then show an efficient way to represent and compute inference channels, which, once known, can be blocked from the output of an association rule finder. The inference channel problem is also discussed in Oliveira et al. (2004), where itemset "sanitization" removes itemsets that lead to sensitive (non-k-anonymous) rules.

This approach is an interesting continuation of Sweeney's classical work (Sweeney 2001), and it addresses an important threat to privacy ignored by most other approaches based on data perturbation or cryptographic protection of the data.

Distributed Data

Most of the work mentioned above addresses the case of centralized data. The distributed situation, however, is often encountered and has important applications. Consider, for example, several hospitals involved in a multi-site medical trial that want to mine the data describing the union of their patients. This increases the size of the population subject to data analysis, thereby increasing the scope and the importance of the trial. In another example, a car manufacturer performing data analysis on the set of vehicles exhibiting a given problem wants to represent data about different components of the vehicle originating in databases of the suppliers of these components. In general, if we abstractly represent the database as a table, there are two collaborative frameworks in which data is distributed. Horizontally partitioned data is distributed by rows (all parties have the same attributes, but different instances – as in the medical study example). Vertically partitioned data is distributed by columns (all parties have the same instances; some attributes belong to specific parties, and some, such as the class, are shared among all parties – as in the vehicle data analysis example).

An important branch of research on learning from distributed data while parties do not reveal their data to each other is based on results from computer security, specifically from cryptography and from the secure multiparty computation (SMC). Particularly interesting is the case when there is no trusted external party – all the computation is distributed among parties that collectively hold the partitioned data. SMC has produced constructive results showing how any Boolean function can be computed from inputs belonging to different parties, so that the parties never get to know input values that do not belong to them. These results are based on the idea of splitting a single data value between two parties into "shares," so that none of them knows the value but they can still do computation on the shares using a gate such as *exclusive or* Yao (1986). In particular, there is an SMC result known as secure sum: k parties have private values x_i and they want to compute $\sigma_I x_i$ without disclosing their x_i to any other party. This result, and similar results for value comparison and other simple functions, are the building blocks of many privacy-preserving ML algorithms. On that basis, a number of standard ▶ classifier induction algorithms, in their horizontal and vertical partitioning versions, have been published, including decision tree (ID3) induction (Friedman et al. 2006), Naïve Bayes, the ▶ Apriori association rule mining algorithm (Kantarcioglu and Clifton 2004; Vaidya and Clifton 2002), and many others.

We can observe that data privacy issues extend to the use of the learned model. For horizontal partitioning, each party can be given the model and apply it to the new data. For vertical partitioning, however, the situation is more difficult: the parties, all knowing the model, have to compute their part of the decision that the model delivers, and have to communicate with selected other parties after this is done. For instance, for decision trees, a node n applies its test and contacts the party holding the attribute in the child c chosen by the test, giving c the test to perform. In this manner, a single party n only knows the result of its test (the corresponding attribute value) and the tests of its children (but not their outcomes). This is repeated recursively until the leaf node is reached and the decision is communicated to all parties.

A different approach involving cryptographic tools other than Yao's circuits is based on the concept of homomorphic encryption (Paillier 1999). Encryption e is homomorphic with respect to some operation $*$ in the message space if there is a corresponding operation $*'$ in the ciphertext space, such that for any messages m1, m2, $e(\mathrm{m1})*' e(\mathrm{m2}) = e(\mathrm{m1}*\mathrm{m2})$. The standard RSA encryption is homomorphic with $*'$ being logical multiplication and $*$ logical addition on sequences of bytes. To give a flavor of the use of homomorphic encryption, let us see in detail how this kind of encryption is used in computing the scalar product of two binary vectors.

Assume just two parties, Alice and Bob. They both have their private binary vectors $\mathrm{A}_{1,\ldots N}$, $\mathrm{B}_{1,\ldots,N}$. In association rule mining, A_i and B_i represent A's and B's transactions projected on the set of items whose frequency is being computed. In our protocol, one of the parties is randomly chosen as a key generator. Assume Alice is selected as the key generator. Alice generates an encryption key (e) and a decryption key (d). She applies the encryption key to the sum of each value of A and a digital envelope $R_i{}^*X$ of A_i (i.e., $e(A_i i + R_i{}^*X)$), where R_i is a random integer and X is an integer that is greater than N. She then sends $e(A_i + R_i{}^*X)$s to Bob. Bob computes the multiplication $M = \prod_{j=1}^{N} [e(A_j + R_i{}^*X) \times B_j]$ when $B_j = 1$ (as when $B_j = 0$, the result of multiplication does not contribute to the frequency count). Now, $M = e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1)^*X)$ due to the property of homomorphic encryption. Bob sends

Privacy-Related Aspects and Techniques, Table 1 Classification taxonomy to systematize the discussion of the current work in PPDM

	Data centralized	Data distributed
Protecting the data	Agrawal and Srikant (2000), Evfimievski et al. (2002), Du and Zhan (2003), and Iyengar (2002)	Vaidya and Clifton (2002), Vaidya et al. (2008), and Kantarcioglu and Clifton (2004)
Protecting the model	Oliveira et al. (2004), Atzori et al. (2005), Felty and Matwin (2002), and Friedman et al. (2006)	Jiang and Atzori (2006)

the result of this multiplication to Alice, who computes $[d(e(A_1 + A_2 + \cdots + A_j + (R_1 + R_2 + \cdots + R_1)^* X))] \bmod X = (A_1 + A_2 + \cdots + A_1 + (R_1 + R_2 + \cdots + R_j)^* X) \bmod X$ and obtains the scalar product. This scalar product is directly used in computing the frequency count of an itemset, where N is the number of items in the itemset, and A_i, B_i are Alice's and Bob's transactions projected on the itemset whose frequency is computed.

While more efficient than the SMC-based approaches, homomorphic encryption methods are more prone to attack, as their security is based on a weaker security concept (Paillier 1999) than Yao's approach. In general, cryptographic solutions have the advantage of protecting the source data while leaving it unchanged: unlike data modification methods, they have no negative impact on the quality of the learned model. However, they have a considerable cost impact in terms of complexity of the algorithms, computation cost of the cryptographic processes involved, and the communication cost for the transmission of partial computational results between the parties (Subramaniam et al. 2004). Their practical applicability on real-life-sized datasets still needs to be demonstrated.

The discussion above focuses on protecting the data. In terms of our diagram in Table 1, we have to address its right column. Here, methods have been proposed to mainly address mainly the north-east entry of the diagram. In particular, in Vaidya and Clifton (2002) propose a method to compute association rules in an environment where data is distributed. In particular, their method addresses the case of vertically partitioned data, where different parties hold different attribute sets for the same instances. The problem is solved without the existence of a trusted third party, using SMC. Independently, we have obtained a different solution to this task using homomorphic encryption techniques (Zhan et al. 2007). Many papers have presented solutions for both vertically and horizontally partitioned data, and for different data mining tasks, e.g., Friedman et al. (2006) and Vaidya et al. (2006).

Moreover, Jiang and Atzori (2006) have obtained a solution for the model-protection case in a distributed setting (south-east quadrant in Table 1). Their work is based on a cryptographic technique, and addresses the case of vertical partitioning of the data among parties.

Evaluation

The evaluation of privacy-related techniques must be broader than standard ML evaluation. Besides evaluating the performance of the ML component using the appropriate tool (e.g., ▶ accuracy, ▶ ROC, support/confidence), one also needs to evaluate the various privacy aspects of a learned model. This is difficult, as there is no commonly accepted definition of privacy. Even if there were one, it would not be in quantitative, operational terms that can be objectively measured, but most certainly with references to moral and social values. For instance, Clifton (2005) points out that a definition of privacy as the "freedom from unauthorized intrusion" implies that we need to understand what constitutes an intrusion and that we can measure its extent. For these reasons, most definitions in current privacy-preserving data mining research are method-specific, without any comparison between different methods. For example, the classic work of Agrawal and Srikant (2000) measures privacy after data perturbation as the size of the interval to which the original value can be estimated. If we know that the original value was 0.5, and following a perturbation its best estimate is, with 95 % confidence, within the interval [0.3, 0.7], then the amount of privacy is the size of this interval, (i.e., 0.4, with a confidence of 95 %). Later, Agrawal and Aggarwal (2001) proposed a more general measure of data privacy measuring this property of a dataset that has been subject to one of the data perturbation techniques. The idea is that if noise from a random variable A is added to the data, we can measure the uncertainty of the perturbed values using differential entropy inherent in A. Specifically, if we add noise from a random variable A, the privacy is

$$\prod(A) = 2^{-f_{\Omega A}^{fA(a)\log_2 f_A(a)da}},$$

where Ω_A is the domain of A. Privacy is 0 if the exact value is known (the entropy is ∞); if it is known that the data is in the interval of length $a, \prod(A) = a$.

Clifton (2005) argues that if disclosure is only possible to a group of people rather than a single person, then the size of the group is a natural measure of privacy. This is the case for k-anonymity methods. He further argues that a good evaluation measure should not only capture the likelihood of linking an ML result to an individual, but should also capture how intrusive this linking is. For instance, an association rule with a support value of 50 and a confidence level of 100 % is 50-anonymous, but it also reveals the consequent of the rule to all 50 participants.

Finally, the style of evaluation needs to take into account attack analysis, as in Malin (2005).

Future Directions

One of the most pressing challenges for the community is to work out a quantifiable and socially comprehensible definition of privacy for the purpose of privacy-preserving techniques. This is clearly a difficult problem, likely not solvable by ML or even computer science alone. As privacy has basic social and economic dimensions, economics may contribute to an acceptable definition, as already explored in Rossi (2004).

Another important question is the ability to analyze data privacy, including inference from data using ML, in the context of specific rules and regulations, e.g., HIPAA (Health and Services 2003) or the European Privacy Directive (1995). First forays in this direction using formal methods have already been made, e.g., Barth et al. (2006) and Felty and Matwin (2002).

Finally, the increasing abundance and availability of data tracking mobile devices will bring new challenges to the field. People will become potentially identifiable by knowing the trajectories their mobile devices leave in fixed times and time intervals. Clearly such data, already collected, present an important asset from the public security point of view, but also a very considerable threat from a privacy perspective. There is early work in this area (Gianotti and Pedreschi 2008). Such data are already being collected. This is an important asset for public security, but also a considerable threat for privacy.

Recommended Reading

Agrawal D, Aggarwal CC (2001) On the design and quantification of privacy preserving data mining algorithms. In: Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, Santa Barbara

Agrawal R, Srikant R (2000) Privacy-preserving data mining. ACM SIGMOD Rec. 29(Part 2):439–450

Atzori M, Bonchi F, Giannotti F, Pedreschi D (2005) k-Anonymous patterns. In: Proceedings of the ninth European conference on principles and practice of knowledge discovery in databases (PKDD 05), Porto

Barth A, Datta A, Mitchell JC, Nissenbaum H (2006) Privacy and contextual integrity: framework and applications. IEEE Symp Secur Priv 184–198

Clifton CW (2005) What is privacy? Critical steps for privacy-preserving data mining, workshop on privacy and security aspects of data mining

Directive (1995) Directive 95/46/EC of the European Parliament on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Off J Eur Commun 38(L281):0031–0050

Domingo-Ferrer J, Sebé F, Solanas A (2008) An anonymity model achievable via microaggregation. In: VLDB workshop on secure data management, Auckland. Springer, pp 209–218
Du W, Zhan Z (2003) Using randomized response techniques for privacy-preserving data mining. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, vol 510
Evfimievski A, Srikant R, Agrawal R, Gehrke J (2002) Privacy preserving mining of association rules. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, pp 217–228
Felty A, Matwin S (2002) Privacy-oriented data mining by proof checking. In: Sixth European conference on principles of data mining and knowledge discovery, Helsink, vol 2431, pp 138–149
Friedman A, Schuster A, Wolff R (2006) k-anonymous decision tree induction. In: PKDD 2006, Berlin, pp 151–162
Health UDo, Services H (eds) (2003) Summary of HIPAA privacy rule. US Department of Health and Human Services, Washington, DC
Gianotti F, Pedreschi D (2008) Mobility, data mining and privacy: geographic knowledge discovery. Springer, Berlin
Iyengar VS (2002) Transforming data to satisfy privacy constraints. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, pp 279–288
Jiang W, Atzori M (2006) Secure distributed k-Anonymous pattern mining. In: Proceedings of the sixth international conference on data mining, Hong Kong. IEEE Computer Society
Kantarcioglu M, Clifton C (2004) Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans Knowl Data Eng 16:1026–1037
Kargupta H, Datta S, Wang Q (2003) On the privacy preserving properties of random data perturbation techniques. In: Third IEEE international conference on data mining (ICDM 2003), Melbourne, pp 99–106
Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) L-diversity: privacy beyond k-anonymity. ACM Trans Knowl Discov Data 1:3
Malin BA (2005) An evaluation of the current state of genomic data privacy protection technology and a roadmap for the future. J Am Med Inf Assoc 12:28
Moor J (2004) Towards a theory of privacy in the information age. In: Bynum T, Rodgerson S (eds) Computer ethics and professional responsibility. Blackwell, Malden
Ninghui L, Tiancheng L, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: IEEE 23rd international conference on data engineering (ICDE 2007), Istanbul, pp 106–115
Oliveira SRM, Zaïane OR, Saygin Y (2004) Secure association rule sharing. In: Proceedings of the eighth PAKDD and advances in knowledge discovery and data mining, Sydney, pp 74–850
Paillier P (1999) The 26th international conference on privacy and personal data protection, advances in cryptography (EUROCRYPT'99), Prague, pp 23–38
Rossi G (2004) Privacy as quality in modern economy. In: The 26th international conference on privacy and personal data protection, Wroclaw
Subramaniam H, Wright RN, Yang Z (2004) Experimental analysis of privacy-preserving statistics computation. In: Proceedings of the VLDB workshop on secure data management, Toronto, pp 55–66
Sweeney L (2001) Computational disclosure control: a primer on data privacy protection. Massachusetts Institute of Technology, Deptartment of Electrical Engineering and Computer Science, Cambridge
Vaidya J, Clifton C (2002) Privacy preserving association rule mining in vertically partitioned data. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Edmonton, pp 639–644
Vaidya J, Clifton C, Kantarcioglu M, Patterson AS (2008) Privacy-preserving decision trees over vertically partitioned data. ACM Trans Knowl Discov Data 2:1–27
Vaidya J, Zhu YM, Clifton CW (2006) Privacy preserving data mining. Springer, New York
Website of the GeoPKDD Project (2006)
Westin A (1967) Privacy and freedom. Atheneum, New York
Yao A (1986) How to generate and exchange secrets. In: 27th FOCS, Toronto
Zhan J, Matwin S, Chang L (2007) Privacy-preserving collaborative association rule mining. J Netw Comput Appl 30:1216–1227
Zhan JZ, Matwin S (2004) Privacy-prteserving data mining in electronic surveys. In: ICEB 2004, Beijing, pp 1179–1185

Probabilistic Context-Free Grammars

Yasubumi Sakakibara
Keio University, Hiyoshi, Kohoku-ku, Japan

Synonyms

PCFG

Definition

In formal language theory, formal grammar (phrase-structure grammar) is developed to capture the generative process of languages (Hopcroft and Ullman 1979). A formal grammar is a set of productions (rewriting rules) that are used to generate a set of strings, that is, a *language*. The productions are applied iteratively to generate a string, a process called *derivation*. The simplest kind of formal grammar is a *regular* grammar.

Context-free grammars (CFG) form a more powerful class of formal grammars than regular grammars and are often used to define the syntax of programming languages. Formally, a CFG consists of a set of nonterminal symbols N, a terminal alphabet Σ, a set P of productions (rewriting rules), and a special nonterminal S called the start symbol. For a nonempty set X of symbols, let X^* denote the set of all finite strings of symbols in X. Every CFG production has the form $S \rightarrow \alpha$, where $S \in N$ and $\alpha \in (N \cup \Sigma)^*$. That is, the left-hand side consists of one nonterminal and there is no restriction on the number or placement of nonterminals and terminals on the right-hand side. The *language generated* by a CFG G is denoted $L(G)$.

A *probabilistic context-free grammar* (PCFG) is obtained by specifying a probability for each production for a nonterminal A in a CFG, such that a probability distribution exists over the set of productions for A.

A CFG $G = (N, \Sigma, P, S)$ is in *Chomsky normal form* if each production rule is of the form $A \rightarrow BC$ or $A \rightarrow a$, where $A, B, C \in N$ and $a \in \Sigma$.

Given a PCFG G and a string $w = a_1 \ldots a_m$, there are three basic problems:

1. Calculating the probability $\Pr(w|G)$ that the grammar G assigns to w
2. Finding the most likely derivation (parse tree) of w by G
3. Estimating the parameters of G to maximize $\Pr(w|G)$

The first two problems, calculating the probability $\Pr(w|G)$ of a given string w assigned by a PCFG G and finding the most likely derivation of w by G, can be solved using dynamic programming methods analogous to the Cocke-Younger-Kasami or Early parsing methods. A polynomial-time algorithm for solving the second problem is known as *Viterbi* algorithm, and a polynomial-time algorithm for the third problem is known as the *inside-outside* algorithm (Lari and Young 1990).

Derivation Process

A *derivation* is a rewriting of a string in $(N \cup \Sigma)^*$ using the production rules of a CFG G. In each step of the derivation, a nonterminal from the current string is chosen and replaced with the right-hand side of a production rule for that nonterminal. This replacement process is repeated until the string consists of terminal symbols only. If a derivation begins with a nonterminal A and derives a string $\alpha \in (N \cup \Sigma)^*$, we write $A \Rightarrow \alpha$.

For example, the grammar in Fig. 1 generates an RNA sequence AGAAACUUGCUGGCCU by the following derivation: Beginning with the start symbol S, any production with S left of the arrow can be chosen to replace S. If the production $S \rightarrow \mathrm{A}X_1\mathrm{U}$ is selected (in this case, this is the only production available), the effect is to replace S with $\mathrm{A}X_1\mathrm{U}$. This one derivation step is written $S \Rightarrow \mathrm{A}X_1\mathrm{U}$, where the double arrow signifies application of a production. Next, if the production $X_1 \rightarrow \mathrm{G}X_2\mathrm{C}$ is selected, the derivation step is $\mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU}$. Continuing with similar derivation operations, each time choosing a nonterminal symbol and replacing it with the right-hand side of an appropriate production, we obtain the following derivation terminating with the desired sequence:

$$\begin{aligned} S &\Rightarrow \mathrm{A}X_1\mathrm{U} \Rightarrow \mathrm{AG}X_2\mathrm{CU} \Rightarrow \mathrm{AG}X_3X_4\mathrm{CU} \\ &\Rightarrow \mathrm{AGA}X_5\mathrm{U}X_4\mathrm{CU} \Rightarrow \mathrm{AGAA}X_6\mathrm{UU}X_4\mathrm{CU} \\ &\Rightarrow \mathrm{AGAAACUU}X_4\mathrm{CU} \end{aligned}$$

[illegible]

Probabilistic Context-Free Grammars, Fig. 1 This set of productions P generates RNA sequences with a certain restricted structure. $S, X_1, \ldots, X_{16}$ are nonterminals; A, U, G, and C are terminals representing the four nucleotides. Note that only for X_6 is there a choice of productions

$$\Rightarrow \text{AGAAACUUG}X_{15}\text{CCU}$$
$$\Rightarrow \text{AGAAACUUGC}X_{16}\text{GCCU}$$
$$\Rightarrow \text{AGAAACUUGCUGGCCU}.$$

Such a derivation can be arranged in a tree structure called a *parse tree*.

The *language generated* by a CFG G is denoted $L(G)$, that is, $L(G) = \{w | S \Rightarrow w, w \in \Sigma^*\}$. Two CFGs G and G' are said to be *equivalent* if and only if $L(G) = L(G')$.

Probability Distribution

A PCFG assigns a probability to each string which it derives and hence defines a probability distribution on the set of strings. The probability of a derivation can be calculated as the product of the probabilities of the productions used to generate the string. The probability of a string w is the sum of probabilities over all possible derivations that could generate w, written as follows:

$$\Pr(w|G) = \sum_{\text{all derivations } d} \Pr(S \overset{d}{\Rightarrow} w|G)$$
$$= \sum_{\alpha_1,\ldots,\alpha_n} \Pr(S \Rightarrow \alpha_1|G) \cdot \Pr(\alpha_1 \Rightarrow \alpha_2|G)$$
$$\ldots \Pr(\alpha_n \Rightarrow w|G).$$

Parsing Algorithm

Efficiently computing the probability of a string w, $\Pr(s|G)$, presents a problem because the number of possible derivations for w is exponential in the length of the string. However, a dynamic programming technique analogous to the Cocke-Kasami-Young or Earley methods for nonprobabilistic CFGs can accomplish this task efficiently (in time proportional to the cube of the length of w).

The CYK algorithm is a polynomial time algorithm for solving the parsing (membership) problem of CFGs using dynamic programming. The CYK algorithm assumes Chomsky normal form of CFGs, and the essence of the algorithm is the construction of a triangular *parse table* T. Given a CFG $G = (N, \Sigma, P, S)$ and an input string $w = a_1 a_2 \ldots a_n$ in Σ^* to be parsed according to G, each element of T, denoted $t_{i,j}$, for $1 \leq i \leq n$ and $1 \leq j \leq n - i + 1$, has a value which is a subset of N. The interpretation of T is that a nonterminal A is in $t_{i,j}$ if and only if $A \Rightarrow a_i a_{i+1} \ldots a_{i+j-1}$, that is, A derives the substring of w beginning at position i and of length j. To determine whether the string w is in $L(G)$, the algorithm computes the parse table T and look to see whether S is in entry $t_{1,n}$.

In the first step of constructing the parse table, the CYK algorithm sets $t_{i,1} = \{ A | A \to a_i$ is in $P\}$. In the jth step, the algorithm assumes that $t_{i,j'}$ has been computed for $1 \leq i \leq n$ and $1 \leq j' < j$, and it computes $t_{i,j}$ by examining the nonterminals in the following pairs of entries:

$$(t_{i,1}, t_{i+1,j-1}), (t_{i,2}, t_{i+2,j-2}), \ldots,$$
$$(t_{i,j-1}, t_{i+j-1,1}),$$

5	S, A				
4	S, A	S, A			
3	S, A	S	S, A		
2	S	A	S	S	
$j = 1$	A	S	A	A	A
	$i = 1$	2	3	4	5
	a	b	a	a	a

Probabilistic Context-Free Grammars, Fig. 2 The parse table T of G for "*abaaa*"

and if B is in $t_{i,k}$ and C is in $t_{i+k,j-k}$ for some k ($1 \leq k < j$) and the production $A \to BC$ is in P, A is added to $t_{i,j}$.

For example, we consider a simple CFG $G = (N, \Sigma, P, S)$ of Chomsky normal form where $N = \{S, A\}$, $\Sigma = \{a, b\}$ and

$$P = \{S \to AA, S \to AS, S \to b,$$
$$A \to SA, A \to a\}.$$

This CFG generates a string "*abaaa*," that is, $S \Rightarrow abaaa$, and the parse table T for *abaaa* is shown in Fig. 2. The parse table can efficiently store all possible parse trees of G for *abaaa*.

Learning

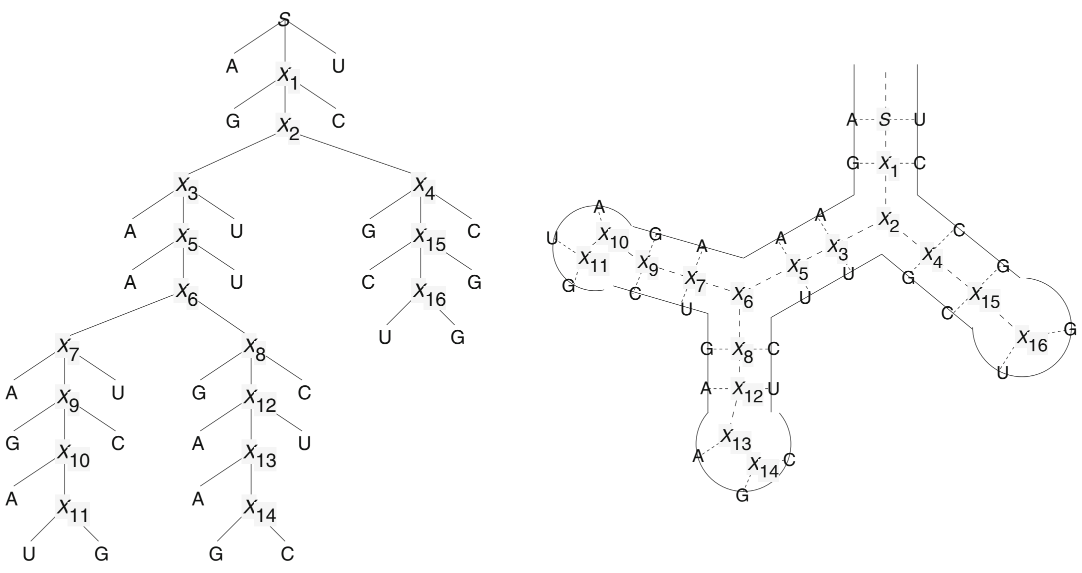
The problem of learning PCFGs from example strings has two aspects: determining a discrete structure (topology) of the target grammar and estimating probabilistic parameters in the grammar (Sakakibara 1997). Based on the maximum likelihood criterion, an efficient estimation algorithm for probabilistic parameters has been proposed: the inside-outside algorithm for PCFGs. On the other hand, finding an appropriate discrete structure of a grammar is a harder problem.

The procedure to estimate the probabilistic parameters of a PCFG is known as the *inside-outside* algorithm. Just like the forward-backward algorithm for HMMs, this procedure is an expectation-maximization (EM) method for obtaining maximum likelihood of the grammar's parameters. However, it requires the grammar to be in Chomsky normal form, which is inconvenient to handle in many practical problems (and requires more nonterminals). Further, it takes time at least proportional to n^3, whereas the forward-backward procedure for HMMs takes time proportional to n^2, where n is the length of the string w. There are also many local maxima in which the method can get caught. Therefore, the initialization of the iterative process is crucial since it affects the speed of convergence and the goodness of the results.

Application to Bioinformatics

An effective method for learning and building PCFGs has been applied to modeling a family of RNA sequences (Durbin et al. 1998; Sakakibara 2005). In RNA, the nucleotides adenine (A), cytosine (C), guanine (G), and uracil (U) interact in specific ways to form characteristic secondary-structure motifs such as helices, loops, and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular A-U and G-C Watson–Crick pairs. Such base pairs constitute the so-called biological palindromes in a genome and can be clearly described by a CFG. In particular, productions of the forms $X \to$ A Y U, $X \to$ U Y A, $X \to$ G Y C, and $X \to$ C Y G describe a structure in RNA due to Watson–Crick base pairing. Using productions of this type, a CFG can specify a language of biological palindromes.

For example, the application of productions in the grammar shown in Fig. 1 generates the RNA sequence CAUCAGGGAAGAUCUCUUG and the derivation can be arranged in a tree structure of a *parse tree* (Fig. 3, left). A parse tree represents the syntactic structure of a sequence produced by a grammar. For the RNA sequence, this syntactic structure corresponds to the physical secondary structure (Fig. 3, right). PCFGs are applied to perform three tasks in RNA sequence analysis: to discriminate RNA-family sequences from nonfamily sequences, to produce multiple alignments, and to ascertain the secondary structure of new sequences.



Probabilistic Context-Free Grammars, Fig. 3 A parse tree (*left*) generated by a simple context-free grammar (CFG) for RNA molecules and the physical secondary structure (*right*) of the RNA sequence which is a reflection of the parse tree

Recommended Reading

Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis. Cambridge University Press, Cambridge

Hopcroft JE, Ullman JD (1979) Introduction to automata theory, languages and computation. Addison-Wesley, Reading

Lari K, Young SJ (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. Comput Speech Lang 4:35–56

Sakakibara Y (1997) Recent advances of grammatical inference. Theor Comput Sci 185:15–45

Sakakibara Y (2005) Grammatical inference in bioinformatics. IEEE Trans Pattern Anal Mach Intell 27:1051–1062

Probability Calibration

▶ Classifier Calibration

Probably Approximately Correct Learning

▶ PAC Learning

Process-Based Modeling

▶ Inductive Process Modeling

Program Synthesis from Examples

▶ Inductive Programming

Programming by Demonstration

Pierre Flener[1] and Ute Schmid[2]

[1]Department of Information Technology, Uppsala University, Uppsala, Sweden

[2]Faculty of Information Systems and Applied Computer Science, University of Bamberg, Bamberg, Germany

P

Abstract

Programming by demonstration (PBD) is introduced as family of approaches to teach a computer system new behavior by demonstrating it in the context of a concrete example.

Most of the work by this author was done while on leave of absence in 2006/2007 as a Visiting Faculty Member and Erasmus Exchange Teacher at Sabancı University, Turkey.

References to classical and current PBD systems are given.

Synonyms

Programming by example (PBE)

Definition

Programming by demonstration (PBD) describes a collection of approaches for the support of end-user programming with the goal of making the power of computers fully accessible to all users. The general objective is to *teach* computer systems new behavior by demonstrating (repetitive) actions on concrete examples. A user provides examples of how a program should operate, either by demonstrating trace steps or by showing examples of the inputs and outputs, and the system infers a generalized program that achieves those examples and can be applied to new examples. Typical areas of application are macro generation (e.g., for text editing), simple arithmetic functions in spreadsheets, simple shell programs, XML transformations, or query-replace commands, as well as the generation of helper programs for web agents, geographic information systems, or computer-aided design. The most challenging approach to PBD is to obtain generalizable examples by minimal intrusion, where the user's ongoing actions are recorded without an explicit signal for the start of an example and without explicit confirmation or rejection of hypotheses. An early example of such a system is EAGER (Cypher 1993a).

Current PBD approaches incorporate some simple forms of **generalization** learning, but typically no or only highly problem-dependent methods for the induction of loops or recursion from examples or traces of repetitive commands. Introducing **inductive programming** or **trace-based programming** methods into PBD applications could significantly increase the possibilities of end-user programming support. This is demonstrated impressively with the Microsoft Excel plug-in Flash Fill (Gulwani et al. 2012).

Cross-References

▶ Inductive Programming
▶ Trace-Based Programming

Recommended Reading

Cypher A (1993a) Programming repetitive tasks by demonstration. In: Cypher A (ed) Watch what I do: programming by demonstration. MIT, Cambridge, pp 205–217

Cypher A (ed) (1993b) Watch what I do: programming by demonstration. MIT, Cambridge

Gulwani S, Harris WR, Singh R (2012) Spreadsheet data manipulation using examples. Commun ACM 55(8):97–105

Lieberman H (ed) (2001) Your wish is my command: programming by example. Morgan Kaufmann, San Francisco

Programming by Example (PBE)

▶ Programming by Demonstration

Programming by Examples

▶ Inductive Programming

Programming from Traces

▶ Trace-Based Programming

Projective Clustering

Cecilia M. Procopiuc
AT&T Labs, NJ, USA

Synonyms

Local feature selection; Subspace clustering

Definition

Projective clustering is a class of problems in which the input consists of high-dimensional data, and the goal is to discover those subsets of the input that are strongly correlated in subspaces of the original space. Each subset of correlated points, together with its associated subspace, defines a *projective cluster*. Thus, although all cluster points are close to each other when projected on the associated subspace, they may be spread out in the full-dimensional space. This makes projective clustering algorithms particularly useful when mining or indexing datasets for which full-dimensional clustering is inadequate (as is the case for most high-dimensional inputs). Moreover, such algorithms compute projective clusters that exist in different subspaces, making them more general than global dimensionality-reduction techniques.

Motivation and Background

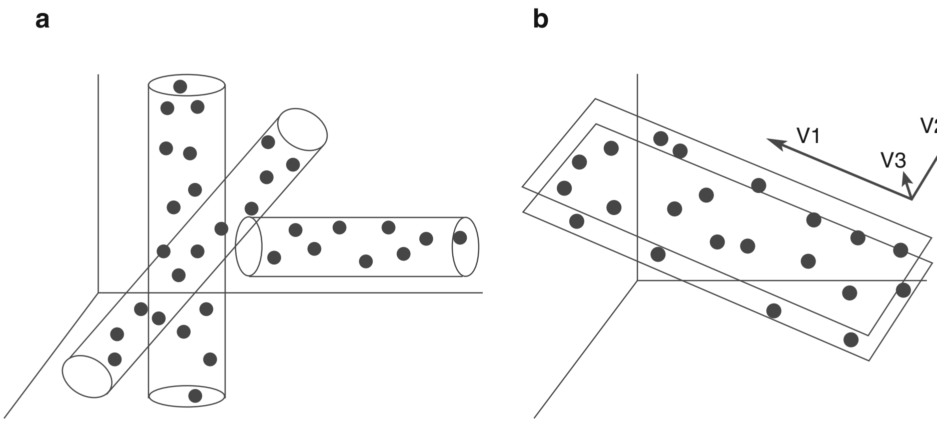
Projective clustering is a type of data mining whose main motivation is to discover correlations in the input data that exist in subspaces of the original space. This is an extension of traditional full-dimensional clustering, in which one tries to discover point subsets that are strongly correlated in all dimensions. Figure 1a shows an example of input data for which full-dimensional clustering cannot discover the three underlying patterns. Each pattern is a projective cluster.

It is well known (Beyer et al. 1999) that for a broad class of data distributions, as the dimensionality increases, the distance to the nearest neighbor of a point approaches the distance to its farthest neighbor. This implies that full-dimensional clustering will fail to discover significantly correlated subsets on such data, since the diameter of a cluster is almost the same as the diameter of the entire dataset. In practice, many applications from text and image processing generate data with hundreds or thousands of dimensions, which makes them extremely bad candidates for full-dimensional clustering methods.

One popular technique to classify high-dimensional data is to first project it onto a much lower-dimensional subspace, and then employ a full-dimensional clustering algorithm in that space. The projection subspace is the same for all points, and is computed so that it best "fits" the data. A widely used dimensionality-reduction technique, called ▶ principal component analysis (PCA), defines the best projection subspace to be the one that minimizes least-square error. While this approach has been proven successful in certain areas such as text mining, its effectiveness depends largely on the characteristics of the data. The reason is that there may be no way to choose a single projection subspace without encountering a significant error; or alternatively, setting a maximum bound on the error results in a subspace with high dimensionality. Figure 1b shows the result of PCA on a good candidate set. The points are projected on the subspace spanned by vectors V_1 and V_2, along which they have greatest variance. However, for the example in Fig. 1a, no plane or line fits the data well enough. Projective clustering can thus be viewed as a generalized dimensionality-reduction method, in which different subsets of the data are projected on different subspaces.

There are many variants of projective clustering, depending on what quality measure one tries to optimize for the clustering. Most such measures, however, are expressed as a function of the distances between points in the clusters. The distance between two cluster points is computed with respect to the subspace associated with that cluster. Alternative quality measures consider the density of cluster points inside the associated subspace.

Megiddo and Tamir (1982) showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines. This early result implies not only that most projective clustering problems are NP-Complete even in the planar case, but also that approximating the objective function within a constant factor is NP-Complete. Nevertheless, several approximation algorithms have been proposed, with running time polyno-



Projective Clustering, Fig. 1 Dimensionality reduction via (**a**) projective clustering and (**b**) principal component analysis

mial in the number of points n and exponential in the number of clusters k. Agrawal et al. (1998) proposed a subspace clustering method based on density measure that computes clusters in a bottom-up approach (from lower to higher dimensions). Aggarwal et al. (1999) designed a partitioning-style algorithm.

Theory

Many variants of projective clustering problems use a distance-based objective function and thus have a natural geometric interpretation. In general, the optimization problem is stated with respect to one or more parameters that constrain the kind of projective clusters one needs to investigate. Examples of such parameters are: the number of clusters, the dimensionality (or average dimensionality) of the clusters, the maximum size of the cluster in its associated subspace, the minimum density of cluster points, etc. Below we present the most frequently studied variants for this problem.

Distance-Based Projective Clustering

Given a set S of n points in $\mathbb{R}^d$ and two integers $k < n$ and $q \leq d$, find kq-dimensional flats $h_1, \ldots, h_k$ and partition S into k subsets $C_1, \ldots, C_k$ so that one of the following objective functions is minimized:

$$\max_{1 \leq i \leq k} \max_{p \in C_i} d(p, h_i) \qquad (k - \text{center})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d(p, h_i) \qquad (k - \text{median})$$

$$\sum_{1 \leq i \leq k} \sum_{p \in C_i} d(p, h_i) \qquad (k - \text{means})$$

These types of problems are also referred to as *geometric clustering problems*. They require all cluster subspaces to have the same dimensionality, i.e., $d - q$ (the subspace associated with C_i is orthogonal to h_i). The number of clusters is also fixed, and the clustering must be a partitioning of the original points.

Further variants are defined by introducing slight modifications in the above framework. For example, one can allow the existence of outliers, i.e., points that do not belong to any projective cluster. This is generally done by providing an additional parameter, which is the maximum percentage of outliers. The problems can also be changed to a dual formulation, in which a maximum value for the objective function is specified, and the goal is to minimize the number of clusters k.

Special cases for the k-center objective function are $q = d - 1$ and $q = 1$. In the first case, the problem is equivalent to finding k hyper-strips that contain S so that the maximum width of a hyper-strip is minimized. If $q = 1$, then the problem is to cover S by k congruent hyper-cylinders of smallest radius. Since this is equivalent to finding the k lines that are the axes of the hyper-cylinders, this problem is also referred to as *k-line-center*. Figure 1a is an example of 3-line-center.

In addition, k-median problems have also been studied when cluster subspaces have different dimensionalities. In that case, distances computed in each cluster are normalized by the dimensionality of the corresponding subspace.

Density-Based Projective Clustering

A convex region in a subspace is called dense if the number of data points that project inside it is larger than some user-defined threshold. For a fixed subspace, the convex regions of interest in that subspace are defined in one of several ways, as detailed below. Projective clusters are then defined to be connected unions of dense regions of interest. The different variants for defining regions of interest can be broadly classified in three classes:

(*ε-Neighborhoods*) Regions of interest are L_p-balls of radius ε centered at the data points. In general, L_p is either L_2 (hyper-spheres) or L_∞ (hyper-cubes).

(Regular Grid Cells) Regions of interest are cells defined by an axis-parallel grid in the subspace. The grid hyper-planes are equidistant along each dimension.

(Irregular Grid Cells) Regions of interest are cells defined by an irregular grid in the subspace. Parallel grid hyper-planes are not necessarily equidistant, and they may also be arbitrarily oriented.

Another variant of projective clustering defines a so-called *quality measure* for a projective cluster, which depends both on the number of cluster points and the number of dimensions in the associated subspace. The goal is to compute the clusters that maximize this measure. Projective clusters are required to be L_p-balls of fixed radius in their associated subspace, which means that clusters in higher dimensions tend to have fewer points, and vice-versa. Hence, the quality measure provides a way to compare clusters that exist in different number of dimensions. It is related to the notion of dense ε-neighborhoods.

Many other projective clustering problems are application driven and do not easily fit in the above classification. While they follow the general framework of finding correlations among data in subspaces of the original space, the notion of projective cluster is specific to the application. One such example is presented later in this section.

Algorithms

Distance-based projective clustering problems are NP-Complete when the number of clusters k is an input parameter. Moreover, k-center problems cannot be approximated within a constant factor, unless $P = NP$. This follows from the result of Meggido and Tamir (1982), who showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines.

Agarwal and Procopiuc (2003) first proposed approximation algorithms for k-center projective clustering in two and three dimensions. The algorithms achieve constant factor approximation by generating more clusters than required.

Subsequent work by several other authors led to the development of a general framework in which $(1 + \varepsilon)$-approximate solutions can be designed for several types of full-dimensional and projective clustering. In particular, k-center and k-means projective clustering can be approximated in any number of dimensions. The idea is to compute a so-called *coreset*, which is a small subset of the points, such that the optimal projective clusters for the coreset closely approximate the projective clusters for the original set. Computing the optimal solution for the coreset has (super) exponential dependence on the number of clusters k, but it is significantly faster than computing the optimal solution for the original set of points. The survey by Agarwal et al. (2005) gives a comprehensive overview of these results.

While the above algorithms have approximation guarantees, they are not practical even for moderate values of n, k, and d. As a result, heuristic methods have also been developed for these problems. The general approach is to iteratively refine a current set of clusters, either by re-assigning points among them, or by merging nearby clusters. When the set of points in a cluster changes, the new subspace associated with the cluster is also recomputed, in a way that tries to optimize the objective function for the new clustering. Aggarwal et al. (1999) proposed the PROCLUS algorithm for k-median projective clustering with outliers. The cluster subspaces can have different dimensionalities, but they must be orthogonal to coordinate axes. Aggarwal and Yu (2000) subsequently extended the algorithm to arbitrarily oriented clusters, but with the same number of dimensions. Agarwal and Mustafa (2004) proposed a heuristic approach for k-means projective clustering with arbitrary orientation and different dimensionalities.

The first widely used method for density-based projective clustering was proposed by Agrawal et al. (1998). The algorithm, called CLIQUE, computes projective clusters based on regular grid cells in orthogonal subspaces, starting from the lowest-dimensional subspaces (i.e., the coordinate axes) and iterating to higher dimensions. Pruning techniques are used to skip subspaces in which a large fraction of points lie outside dense regions. Subsequent strategies improved the running time and accuracy by imposing irregular grids and using different pruning criteria.

Böhm et al. (2004) designed an algorithm called 4C for computing density-connected ε-neighborhoods in arbitrarily oriented subspaces. The method is agglomerative: It computes the local dimensionality around each point p by using PCA on all points inside the (full-dimensional) ε-neighborhood of p. If the dimensionality is small enough and the neighborhood is dense, then p and its neighbors form a projective cluster. Connected projective clusters with similarly oriented subspaces are then repeatedly merged.

The OptiGrid algorithm by Hinneburg and Keim (1999) was the first method to propose irregular grid cells of arbitrary (but fixed) orientation. Along each grid direction, grid hyper-planes are defined to pass through the local minima of a probability density function. This significantly reduces the number of cells compared with a regular grid that achieves similar overall accuracy. The probability density function is defined using the kernel-density estimation framework. Input points are projected on the grid direction, and their distribution is extrapolated to the entire line by the density function

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - s_i}{h}\right),$$

where $s_1, \ldots, s_n$ denote the projections of the input points, and h is a parameter. The function $K(x)$, called the *kernel*, is usually the Gaussian function, although other kernels can also be used.

The DOC algorithm proposed by Procopiuc et al. (2002) approximates optimal clusters for a class of quality measures. Orthogonal projective clusters are computed iteratively via random sampling. If a sample is fully contained in a cluster then it can be used to determine the subspace of that cluster, as well as (a superset of) the other cluster points. Such a sample is called a *discriminating set*. Using the properties of the quality measure, the authors show that a discriminating set is found with high probability after a polynomial number of trials.

An overview of most of these practical methods, as well as of subsequent work expanding their results, can be found in the survey by Parsons et al. (2004).

Applications

Similar to full-dimensional clustering, projective clustering methods provide a way to efficiently organize databases for searching, as well as for pattern discovery and data compression. In a broad sense, they can be used in any application that handles high-dimensional data, and which can benefit from indexing or mining capabilities. In practice, additional domain-specific information is often necessary. We present an overview of the generic database usage first, and then discuss several domain-specific applications.

Data Indexing

An index tree is a hierarchical structure defined on top of a data set as follows. The root corresponds to the entire data set. For each internal node, the data corresponding to that node is partitioned in some pre-defined manner, and there is a child of the node corresponding to each subset in the partition. Often, the partitioning method is a distance-based clustering algorithm. In addition, each node stores the boundary of a geometric region that contains its points, to make searching the structure more efficient. For many popular indexes, the geometric region is the minimum axis-parallel bounding box. Index trees built with full-dimensional clustering methods become inefficient for dimensionality about 10 or higher, due to the large overlap in the geometric regions of sibling nodes. Chakrabarti and Mehrotra (2000) first proposed an index tree that uses projective clustering as a partitioning method. In that case, each node also stores the subspace associated with the cluster.

Pattern Discovery

A projective cluster, by definition, is a pattern in the data, so any of the above algorithms can be used in a pattern discovery application. However, most applications restrict the projective clusters to be orthogonal to coordinate axes, since the axes have special interpretations. For example, in a database of employees, one axis may represent salary, another the length of employment, and the third one the employees' age. A projective cluster in the subspace spanned by salary and employment length has the following interpretation: there is a correlation between salaries in range A and years of employment in range B, which is independent of employees' age.

Data Compression

As discussed in the introduction, projective clusters can be used as a dimensionality-reduction technique, by replacing each point with its projection on a lower dimensional subspace. The projection subspace is orthogonal to the subspace of the cluster that contains the point. In general, this method achieves smaller information loss and higher compression ratio than a global technique such as PCA.

Image Processing

A picture can be represented as a high-dimensional data point, where each pixel represents one dimension, and its value is equal to the RGB color value of the pixel. Since this representation loses pixel adjacency information, it is generally used in connection with a smoothing technique, which replaces the value of a pixel with a function that depends both on the old pixel value, and the values of its neighbors. A projective cluster groups images that share some similar features, while they differ significantly on others. The DOC algorithm has been applied to the face detection problem as follows: Projective clusters were computed on a set of (pre-labeled) human faces, then used in a classifier to determine whether a new image contained a human face.

Document Processing

Text documents are often represented as sparse high-dimensional vectors, with each dimension corresponding to a distinct word in the document collection. Several methods are used to reduce the dimensionality, e.g., by eliminating so-called stop words such as "and," "the," and "of." A non zero entry in a vector is usually a function of the corresponding word's frequency in the document. Because of the inherent sparsity of the vectors, density-based clustering, as well as k-center methods, are poor choices for such data. However, k-means projective clustering has been successfully applied to several document corpora (Li et al. 2004).

DNA Microarray Analysis

A gene-condition expression matrix, generated by a DNA microarray, is a real-valued matrix, such that each row corresponds to a gene, and each column corresponds to a different condition. An entry in a row is a function of the relative abundance of the mRNA of the gene under that specific condition. An orthogonal projective cluster thus represents several genes that have similar expression levels under a subset of conditions. Genetics researchers can infer connections between a disease and the genes in a cluster. Due to the particularities of the data, different notions of similarity are often required. For example, order preserving clusters group genes that have the same tendency on a subset of attributes, i.e., an attribute has the same rank (rather than similar value) in each projected gene. See the results of Liu and Wang (2003).

Principal Component Analysis

PCA also referred to as the Karhunen-Loève Transform, is a global ▸ dimensionality reduction technique, as opposed to projective clustering, which is a local dimensionality reduction method. PCA is defined as an orthogonal linear transformation with the property that it transforms the data into a new coordinate system, such that the projection of the data on the first coordinate has the greatest variance among all projections on a line, the projection of the data on the second coordinate has the second greatest variance, and so on. Let X denote the data matrix, with each point written as a column vector in X, and modified so that X has empirical mean zero (i.e., the mean vector is subtracted from each data point). Then the eigenvectors of the matrix $XX\ T$ are the coordinates of the new system. To reduce the dimensionality, keep only the eigenvectors corresponding to the largest few eigenvalues.

Coresets

Let $P \subseteq \mathbb{R}^d$ be a set of points, and μ be a measure function defined on subsets of $\mathbb{R}^d$, such that μ is monotone (i.e., for $P_1 \subseteq P_2$, $\mu(P_1) \leq \mu(P_2)$). A subset $Q \subseteq P$ is an *ε-coreset* with respect to μ if $(1 - \varepsilon)\mu(P) \leq \mu(Q)$. The objective functions for k-center, k-median, and k-means projective clustering are all examples of measure functions μ.

Cross-References

▸ Clustering
▸ Curse of Dimensionality
▸ Dimensionality Reduction
▸ Kernel Methods
▸ K-Means Clustering
▸ Principal Component Analysis

Recommended Reading

Agarwal PK, Mustafa N (2004) k-means projective clustering. In: Proceeding of ACM SIGMOD-SIGACT-SIGART symposium principles of database systems, pp 155–165

Agarwal PK, Procopiuc CM (2003) Approximation algorithms for projective clustering. J Algorithms 46(2):115–139

Agarwal PK, Har-Peled S, Varadarajan KR (2005) Geometric approximation via coresets. In: Goodman JE, Pach J, Welzl E (eds) Combinatorial and computational geometry. Cambridge University Press, Cambridge/New York, pp 1–30

Aggarwal CC, Yu PS (2000) Finding generalized projected clusters in high dimensional spaces. In: Proceeding of ACM SIGMOD international conference management of data, pp 70–81

Aggarwal CC, Procopiuc CM, Wolf JL, Yu PS, Park JS (1999) Fast algorithms for projected clustering. In: Proceeding of ACM SIGMOD international conference management of data, pp 61–72

Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: Proceeding of ACM SIGMOD international conference management of data, pp 94–105

Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbour" meaningful? In: Proceeding of 7th international conference data theory, vol 1540, pp 217–235

Böhm C, Kailing K, Kröger P, Zimek A (2004) Computing clusters of correlation connected objects. In: Proceeding of ACM SIGMOD international conference management of data, pp 455–466

Chakrabarti K, Mehrotra S (2000) Local dimensionality reduction: a new approach to indexing high dimensional spaces. In: Proceeding of 26th international conference very large data bases, pp 89–100

Hinneburg A, Keim DA (1999) Optimal grid-clustering: towards breaking the curse of dimensionality in high-dimensional clustering. In: Proceeding of 25th international conference very large data bases, pp 506–517

Li T, Ma S, Ogihara M (2004) Document clustering via adaptive subspace iteration. In: Proceeding of 27th international ACM SIGIR conference research and development in information retrieval, pp 218–225

Liu J, Wang W (2003) Op-cluster: clustering by tendency in high dimensional space. In: Proceeding of international conference on data mining, pp 187–194

Megiddo N, Tamir A (1982) On the complexity of locating linear facilities in the plane. Oper Res Lett 1:194–197

Parsons L, Haque E, Liu H (2004) Subspace clustering for high dimensional data: a review. ACM SIGKDD Explor Newslett 6(1):90–105

Procopiuc CM, Jones M, Agarwal PK, Murali TM (2002) A Monte Carlo algorithm for fast projective clustering. In: Proceeding of ACM SIGMOD international conference management of data, pp 418–427

Prolog

Prolog is a declarative programming language based on logic. It was conceived by French and British computer scientists in the early 1970s. A considerable number of public-domain and commercial Prolog interpreters are available today. Prolog is particularly suited for applications requiring pattern matching or search. Prolog programs are also referred to as ▸ logic programs.

In machine learning, classification rules for structured individuals can be expressed using a subset of Prolog. Learning Prolog programs from examples is called ▸ inductive logic programming (ILP). ILP systems are sometimes – but not always – implemented in Prolog. This has the advantage that classification rules can be executed directly by the Prolog interpreter.

Cross-References

▸ Clause
▸ First-Order Logic
▸ Inductive Logic Programming
▸ Logic Program

Recommended Reading

Colmerauer A, Kanoui H, Pasero R, Roussel P (1973) Un système de communication homme-machine an Français. Report, Groupè d'Intelligence Artificielle, University d'Aix Marseille II, Luminy

Kowalski RA (1972) The predicate calculus as a programming language. In: Proceedings of the international symposium and summer school on mathematical foundations of computer science, Jablonna

Roussel P (1975) Prolog: Manual de reference et d'utilization. Technical report, Groupe d'Intelligence Artificielle, Marseille-Luminy

Property

▸ Attribute

Propositional Logic

Propositional logic is the logic of propositions, i.e., expressions that are either true or false. Complex propositions are built from propositional atoms using logical connectives. Propositional logic is a special case of predicate logic, where all ▸ predicates have zero arity; see the entry on first-order logic for details.

Cross-References

▸ First-Order Logic
▸ Propositionalization

Propositionalization

Nicolas Lachiche
University of Strasbourg, Strasbourg, France

Abstract

Propositionalization is the process of explicitly transforming a ▸ relational dataset into a propositional dataset.

Definition

The input data consists of examples represented by structured terms (cf. ▸ learning from structured data), several predicates in ▸ first-order logic, or several tables in a relational database. We will jointly refer to these as *relational representations*. The output is an ▸ attribute-value representation in a single table, where each example corresponds to one row and is described by its values for a fixed set of attributes. New attributes are often called features to emphasize that they are built from the original attributes. The aim of propositionalization is to preprocess relational data for subsequent analysis by attribute-value learners. There are several reasons for doing this, the most important of which are to reduce the complexity and speed up the learning, to separate modeling the data from hypothesis construction, or to use familiar attribute-value (or propositional) learners.

Motivation and Background

Most domains are naturally modeled by several tables in a relational database or several classes in an object-oriented language, for example, customers and their transactions; molecules, their atoms, and bonds; or patients and their examinations. A proper relational dataset involves at least two tables linked together. Typically, one table of the relational representation corresponds to the individuals of interest for the machine learning task, and the other tables contain related information that could be useful. The first table is the individual or the primary table; the other tables are complementary tables.

Example 1 Let us consider a simplified medical domain as an example. This is inspired by a real medical dataset (Tomečková et al. 2002). It consists of four tables.

The patient table is the primary table. It contains data on each patient such as the patient identifier (pid), name, date of birth, height, job, the identifier of the company where the patient works, etc.

Patient

pid	Name	Birth	Height	Job	Company	...
I	Smith	15/06/1956	1.67	Manager	a	...
II	Blake	13/02/1968	1.82	Salesman	a	...
⋮	⋮	⋮	⋮	⋮	⋮	...

The company table contains its name, its location, and so on. There is a many-to-one relationship from the patient table to the company table: A patient works for a single company, but a company may have several employees.

Company

cid	Name	Location	...
a	Eiffel	Paris	...
⋮	⋮	⋮	...

The examination table contains the information on all examinations of all patients. For each examination, its identifier (eid), the patient identifier (pid), the date, the patient's weight, whether the patient smokes, his or her blood pressure, etc. are recorded. Of course, each examination corresponds to a single patient, and a given patient can have several examinations, i.e., there is a one-to-many relationship from the patient table to the examination table.

Examination

eid	pid	Date	Weight	Smokes	BP	...
1	I	10/10/1991	60	Yes	10	...
2	I	04/06/1992	64	Yes	12	...
⋮	⋮	⋮	⋮	⋮	⋮	...
23	II	20/12/1992	80	Yes	10	...
24	II	15/11/1993	78	No	11	...
⋮	⋮	⋮	⋮	⋮	⋮	...

Additional tests can be prescribed at each examination. Their identifiers (tid), corresponding examinations (eid), names, values, and interpretations are recorded in the additional_test table.

Additional_test

tid	eid	Date	Name	Value	Interpretation
t237	1	19/10/1991	Red blood cells	35	Bad
t238	1	23/10/1991	Radiography	Nothing	Good
⋮	⋮	⋮	⋮	⋮	⋮
t574	2	07/06/1992	Red blood cells	43	Good
⋮	⋮	⋮	⋮	⋮	⋮

Several approaches exist to deal directly with relational data, e.g., ▶ inductive logic programming, ▶ relational data mining (Džeroski and Lavrač 2001), or ▶ statistical relational learning. However relational hypotheses can be transformed into propositional expressions.

Generally, a richer representation language permits the description of more complex concepts; however, the cost of this representational power is that the search space for learning greatly increases. Therefore, mapping a relational representation into a propositional one generally reduces search complexity.

A second motivation of propositionalization is to focus on the construction of features before combining them into a hypothesis (Srinivasan et al. 1996). This is related to ▶ feature construction and to the use of background knowledge. One could say that propositionalization aims at building an intermediate representation of the data in order to simplify the hypothesis subsequently found by a propositional learner.

A third motivation is pragmatic. Most available machine learning systems deal with propositional data only, but tend to include a range of algorithms in a single environment, whereas relational learning systems tend to concentrate on a single algorithm. Propositional systems are therefore often more versatile and give users the possibility to work with the algorithms they are used to.

Solutions

There are various ways to propositionalize relational data consisting of at least two tables linked together through a relationship. We will first focus on a single relationship between two tables. Most approaches can then iteratively deal with several relationships as explained below.

Propositionalization mechanisms depend on whether that relationship is functional or nondeterminate. This distinction explains most common mistakes made by newcomers.

Functional Relationship (Many-to-One, One-to-One)

When the primary table has a many-to-one or one-to-one relationship to the complementary table, each row of the primary table links to one row of the complementary table. A simple join of the two tables results in a single table where each row of the primary table is completed with the information derived from the complementary table.

Example 2 In our simplified medical domain, there is a many-to-one relationship from each patient to his or her company. Let us focus on those two tables only. A join of the two tables results in a single table where each row describes a single patient and the company he or she works for.

Patient and his/her company

pid	Name	Birth	Height	Job	cid	Company	Location	...
I	Smith	15/06/ 1956	1.67	Mana- ger	a	Eiffel	Paris	...
II	Blake	13/02/ 1968	1.82	Sales- man	a	Eiffel	Paris	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

The resulting table is suitable for any attribute-value learner

Nondeterminate Relationship (One-to-Many, Many-to-Many)

Propositionalization is less trivial in a nondeterminate context, when there is a one-to-many or many-to-many relationship from the primary table to the complementary table, i.e., when one individual of the primary table is associated with a set of rows of the complementary table.

A propositional attribute is built by applying an aggregation function to a column of the complementary table over a selection of rows. Of course a lot of conditions can be used to select the rows. Those conditions can involve other columns than the aggregated column. Any aggregation function can be used, e.g., to check whether the set is not empty, to count how many elements there are, to find the mean (for numerical) or the mode (for categorical) values, etc.

Example 3 In our simplified medical domain, there is a one-to-many relationship from the patient to his or her examinations. Let us focus on those two tables only. Many features can be constructed. Simple features are aggregation functions applied to a scalar (numerical or categorical) column. The number of occurrences of the different values of every categorical attributes can be counted. For instance, the f60 feature in the table below counts in how many examinations the patient stated he or she smoked. The maximum, minimum, average, and standard deviation of every numerical column can be estimated, e.g., the f84 and f85 features in the table below, respectively, estimate the average and the maximum blood pressure of the patient over his or her examinations. The aggregation functions can be applied to any selection of rows, e.g., the f135 feature in the table below estimates the average blood pressure over the examinations when the patient smoked.

Patient and his/her examinations

pid	Name	…	f60	…	f84	f85	…	f135	…
I	Smith	…	2	…	11	12	…	11	…
II	Blake	…	1	…	10.5	11	…	10	…
⋮	⋮	…	⋮	…	⋮	⋮	…	⋮	…

From this example it is clear that nondeterminate relationships can easily lead to a combinatorial explosion of the number of features.

Common Mistakes and Key Rules to Avoid Them

Two mistakes are frequent when machine learning practitioners face a propositionalization problem, i.e., when they want to apply a propositional learner to an existing relational dataset (Lachiche 2005).

The first mistake is to misuse the (universal) join. Join is valid in a functional context, as explained earlier. When applied to a nondeterminate relationship, it produces a table where several rows correspond to a single individual, leading to a multiple-instance problem (Dietterich et al. 1997) (cf. ▶ multi-instance learning).

Example 4 In our simplified medical domain, there is a one-to-many relationship from the patient table to the examination table. If a join is performed, each row of the examination table is completed with the information on the examined patient, i.e., there are as many rows as examinations.

Examination and its patient

eid	Date	Weight	Smokes	BP	…	pid	Name	…
1	10/10/1991	60	Yes	10	…	I	Smith	…
2	04/06/1992	64	Yes	12	…	I	Smith	…
⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	…
23	20/12/1992	80	Yes	10	…	II	Blake	…
24	15/11/1993	78	No	11	…	II	Blake	…
⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	…

In this example, the joined table deals with the examinations rather than with the patients. An attribute-value learner could be used to learn hypotheses about the examinations, not about the patients

This example reinforces a key representation rule in attribute-value learning: "Each row corresponds to a single individual, and vice-versa."

The second mistake is a meaningless column concatenation. This is more likely when a one-to-many relationship can be misinterpreted as several one-to-one relationships, i.e., when the practitioner is led to think that a nondeterminate relationship is actually functional.

Example 5 In our simplified medical domain, let us assume that the physician numbered the successive examinations (1, 2, 3, and so on) of each patient. Then given that each patient has a first examination, it is tempting to consider that there is a functional relationship from the patient to his or her "first" examination, "second" examination, and so on. This would result in a new patient table with concatenated columns: weight at the first examination, whether he or she smoked at the first examination, …, weight at the second examination, etc. This could easily lead

Patient and his/her examinations (incorrect representation!)

pid	Name	. . .	“First” examination			“Second” examination			. . .
			Weight	Smokes	. . .	Weight	Smokes	. . .	. . .
I	Smith	. . .	60	Yes	. . .	64	Yes	. . .	. . .
II	Blake	. . .	80	Yes	. . .	78	No	. . .	. . .
⋮	⋮	. . .	⋮	. . .	⋮	⋮	⋮	. . .	. . .

to an attribute-value learner generalizing over a patient’s weight at their *i*th examination, which is very unlikely to be meaningful

Two aspects should warn the user of such a representation problem: first, the number of columns depends on the dataset, and as a consequence, lots of columns are not defined for all individuals. Moreover, when the absolute numbering does not make sense, there is no functional relationship. Such a misunderstanding can be avoided by remembering that in an attribute-value representation, “each column is uniquely defined for each row.”

Further Relationships

The first complementary table can itself have a nondeterminate relationship with another complementary table and so on. Two approaches are available.

A first approach is to consider the first complementary table, the one having a one-to-many relationship, as a new primary table in a recursive propositionalization.

Example 6 In our simplified medical domain, the examination table has a one-to-many relationship with the additional_test table. The propositionalization of the examination and additional test tables will lead to a new examination table completed with new features, such as a count of how many tests were bad.

Examination and its additional_tests

eid	pid	Date	Weight	Smokes	BP	. . .	Bad tests	. . .
1	I	10/10/1991	60	Yes	10	. . .	1	. . .
2	I	04/06/1992	64	Yes	12	. . .	0	. . .
⋮	⋮	⋮	⋮	⋮	⋮	. . .	⋮	. . .

Then the propositionalization of the patient table and the already propositionalized examination tables is performed, producing a new patient table completed with new features such as the mean value for each patient of the number of bad tests among all his or her examinations (f248)

Patient, his/her examinations and additional_tests

pid	name	. . .	f60	. . .	f248	. . .
I	Smith	. . .	2	. . .	1	. . .
⋮	⋮	. . .	⋮	. . .	⋮	. . .

It is not necessarily meaningful to aggregate at an intermediate level. An alternative is to join complementary tables first and apply the aggregation at the individual level only. A variant consists in replacing the join by a propagation of the identifier, i.e., adding the identifier of the individual into all related tables. Both lead to a kind of “star schema” where the individual is directly linked to all complementary tables.

Example 7 In our simplified medical domain, it is perhaps more interesting to first relate all additional tests to their patients, then aggregate on similar tests. First the complementary tables are joined

Additional_test and its examination

tid	Name	Value	Interpretation	eid	pid	Weight	. . .
t237	Red blood cells	35	Bad	1	I	60	. . .
t238	Radiography	Nothing	Good	1	I	60	. . .
⋮	⋮	⋮	⋮	⋮	⋮	⋮	. . .
t574	Red blood cells	43	Good	2	I	64	. . .
⋮	⋮	⋮	⋮	⋮	⋮	⋮	. . .

Let us emphasize the difference with the propositionalized examination and its additional_tests table of Example 6

There is a one-to-many relationship from the patient table to that new additional_test and its examination table. Aggregation functions can be used to build features such as the minimum percentage of red blood cells (f352)

Patient, his/her additional_tests and examinations

pid	Name	...	f60	...	f352	...
I	Smith	...	2	...	35	...
⋮	⋮	...	⋮	...	⋮	...

Finally, different propositionalization approaches can be combined, by a simple join.

Future Directions

Propositionalization explicitly aims at leaving attribute selection to the propositional learner applied afterward. The number of potential features is large. No existing propositionalization system is able to enumerate all imaginable features. Historically existing approaches have focused on a subset of potential features, e.g., numerical aggregation functions without selection (Knobbe et al. 2001) and selection based on a single elementary condition and existential aggregation (Flach and Lachiche 1999; Kramer et al. 2001). Most approaches can be combined to provide more features. The propositionalization should be guided by the user.

Propositionalization is closely related to knowledge representation. Specific representational issues require appropriate propositionalization techniques, e.g., Perlich and Provost (2006) introduce new propositionalization operators to deal with high-cardinality categorical attributes. New data sources, such as geographical or multimedia data, will need an appropriate representation and perhaps appropriate propositionalization operators to apply off-the-shelf attribute-value learners.

Propositionalization raises three fundamental questions. The first question is related to knowledge representation. That question is whether the user should adapt to existing representations, and accept a need to propositionalize, or whether data can be mined from the data sources, requiring the algorithms to be adapted or invented. The second question is whether propositionalization is needed. Propositionalization explicitly allows the user to contribute to the feature elaboration and invites him or her to guide the search, thanks to that language bias. It separates feature elaboration from model extraction. Conversely, relational data mining techniques automate the elaboration of the relevant attributes during the model extraction, but at the same time leave less opportunity to select the features by hand.

The third issue is one of efficiency. A more expressive representation necessitates a more complex search. Relational learning algorithms face the same dilemma as attribute-value learning in the form of a choice between an intractable search in the complete search space and an ad hoc heuristic/search bias (cf. ▶ search bias). They only differ in the size of the search space (cf. ▶ hypothesis space). Propositionalization is concerned with generating the search space. Generating all potential features is usually impossible. So practitioners have to constrain the propositionalization, e.g., by choosing the aggregation functions, the complexity of the selections, etc.; by restricting the numbers of operations; and so on. Different operators fit different problems and might lead to differences in performance (Krogel et al. 2003).

Cross-References

▶ Attribute
▶ Feature Construction in Text Mining
▶ Feature Selection
▶ Inductive Logic Programming
▶ Language Bias

▶ Learning from Structured Data
▶ Multi-instance Learning
▶ Relational Learning
▶ Statistical Relational Learning

Recommended Reading

Dietterich TG, Lathrop RH, Lozano-Pérez T(1997) Solving the multiple-instance problem with axis-parallel rectangles. Artif Intell 89(1–2):31–71

Džeroski S, Lavrač N (eds) (2001) Relational data mining. Springer, New York

Flach P, Lachiche N (1999) 1BC: a first-order Bayesian classifier. In: Džeroski S, Flach P (eds) Proceedings of the ninth international workshop on inductive logic programming (ILP'99). Volume 1634 of lecture notes in computer science. Springer, pp 92–103

Knobbe AJ, de Haas M, Siebes A (2001) Propositionalisation and aggregates. In: Proceedings of the sixth European conference on principles of data mining and knowledge discovery. Volume 2168 of lecture notes in artificial intelligence. Springer, pp 277–288

Kramer S, Lavrač N, Flach P (2001) Propositionalization approaches to relational data mining. In: Džeroski S, Lavrač N (eds) Relational data mining. Springer, New York, chap 11, pp 262–291

Krogel M-A, Rawles S, Železný F, Flach PA, Lavrač N, Wrobel S (2003) Comparative evaluation of approaches to propositionalization. In: Horváth T, Yamamoto A (eds) Proceedings of the thirteenth international conference on inductive logic programming. volume 2835 of lecture notes in artificial intelligence. Springer, pp 197–214

Lachiche N (2005) Good and bad practices in propositionalisation. In: Bandini S, Manzoni S (eds) Proceedings of advances in artificial intelligence, ninth congress of the Italian association for artificial intelligence (AI*IA'05). Volume 3673 of lecture notes in computer science. Springer, pp 50–61

Perlich C, Provost F (2006) Distribution-based aggregation for relational learning with identifier attributes. Mach Learn 62:62–105

Srinivasan A, Muggleton S, King RD, Stenberg M (1996) Theories for mutagenicity: a study of first-order and feature based induction. Artif Intell 85(1–2):277–299

Tomečková M, Rauch J, Berka P (2002) Stulong – data from longitudinal study of atherosclerosis risk factors. In: Berka P (ed) Discovery challenge workshop notes, ECML/PKDD'02.

Prospective Evaluation

Prospective evaluation is an approach to ▶ Out-Of-Sample Evaluation whereby a model learned from historical data is evaluated by observing its performance on new data as they become available. Prospective evaluation is likely to provide a less biased estimation of future performance than evaluation on historical data.

Cross-References

▶ Algorithm Evaluation

Pruning

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

Pruning describes the idea of avoiding ▶ Overfitting by simplifying a learned concept, typically after the actual induction phase.

Method

The term originates from decision tree learning, where the idea of improving the decision tree by cutting some of its branches may be viewed as an analogy to the concept of pruning in gardening.

Commonly, one distinguishes two types of pruning:

Pre-pruning monitors the learning process and prevents further refinements if the current hypothesis becomes too complex.

Post-pruning first learns a possibly overfitting hypothesis and then tries to simplify it in a separate learning phase.

Pruning techniques are particularly important for state-of-the-art ▶ Decision Tree and ▶ Rule Learning algorithms (see there for more details).

The key idea of pruning is essentially the same as ▶ Regularization in statistical learning, with the key difference that regularization incorporates a complexity penalty directly into the learning heuristic, whereas pruning uses a separate pruning criterion or pruning algorithm.

Cross-References

▶ Decision Tree
▶ Regularization
▶ Rule Learning

Pruning Set

Definition

A pruning set is a subset of a ▶ training set containing data that are used by a learning system to evaluate models that are learned from a ▶ growing set.

Cross-References

▶ Data Set