O

Object

►Instance

Object Consolidation

►Entity Resolution

Object Space

►Example Space

Observation Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
Leiden Institute of Advanced Computer Science
The Netherlands

Synonyms

Instance language

Definition

The *observation language* used by a machine learning system is the language in which the observations it learns from are described.

Motivation and Background

Most machine learning algorithms can be seen as a procedure for deriving one or more hypotheses from a set of observations. Both the input (the observations) and the output (the hypotheses) need to be described in some particular language and this language is called the observation language or the ►Hypothesis Language respectively. These terms are mostly used in the context of symbolic learning, where these languages are often more complex than in subsymbolic or statistical learning.

The following sections describe some of the key observation languages.

Attribute-Value Learning

Probably the most used setting in machine learning is the *attribute-value* setting (see ►Attribute-Value Learning). Here, an example (observation) is described by a fixed set of attributes, each of which is given a value from the domain of the attribute. Such an observation is often called a vector or, in relational database terminology, a tuple. The attributes are usually atomic (i.e., not decomposable in component values) and single-valued (i.e., an attribute has only one value, not a set of values). So we have an instance space (or space of observations)

$$\mathcal{O} = A_1 \times \cdots \times A_n,$$

elements of which are denoted using an observation language that typically has the same structure:

$$\mathcal{L}_O = \mathcal{L}_{A_1} \times \cdots \times \mathcal{L}_{A_n}$$

(the language contains tuples of objects that represent the attribute values).

The attribute-value framework easily allows for both supervised and unsupervised learning; in the supervised learning setting, the label of an instance is simply included as an attribute in the tuple, where as for unsupervised learning, it is excluded.

The attribute-value setting assumes that all instances can be represented using the same fixed set of attributes. When instances can be of different types or are variable-sized (e.g., when an instance is set-valued), this assumption may not hold, and more powerful languages may have to be used instead.

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

Learning from Graphs, Trees, or Sequences

We here consider the case in which a single instance is a graph, or a node in a graph. Note that trees and sequences are special cases of graphs.

A graph is defined as a pair (V, E), where V is a set of vertices and E a set of edges each edge being a pair of vertices. If the pair is ordered, the graph is directed; otherwise it is undirected. For simplicity, we restrict ourselves to undirected graphs.

A graph can, in practice, not be encoded in attribute-value format without the loss of information. That is, one could use a number of properties of graphs as attributes in the encoding, but several graphs may then still map onto the same representation, which implies loss of information. In theory, one could imagine defining a total order on (certain classes of) graphs and representing each graph by its rank in that order (which is a single numerical attribute), thus representing graphs as numbers without loss of information; but then it is not obvious how to map patterns in this numerical representation to patterns in the original representation. No such approaches have been proposed till now.

Describing the instance space is more difficult here than in the attribute value case. Consider a task of graph classification, where in observations are of the form (G, y) with G a graph and y a value for a target attribute Y. Then we can define the instance space as

$$\mathcal{O} = \{(V, E) | V \subseteq \mathbf{N} \wedge E \subseteq V^2\} \times Y,$$

where $\mathbf{N}$ is the set of all natural numbers. (For each graph, there exists a graph defined over $\mathbf{N}$ that is isomorphic with it, so $\mathcal{O}$ contains all possible graphs up to isomorphism.)

A straightforward observation language in the case of graph classification is then

$$\{(G, y) | G = (V, E) \wedge V \subseteq \mathcal{L}_V \wedge E \subseteq V^2 \wedge y \in Y\},$$

where $\mathcal{L}_V$ is some alphabet for representing nodes.

In learning from graphs, there are essentially two settings: those where a prediction is made for entire graphs, and those where a prediction is made for single nodes in a graph. In the first case, observations are of the form (G, y), where as, in the second case, they are of the form (G, v, y), where $G = (V, E)$ and $v \in V$. That is, a node is given together with the graph in which it occurs (its "environment"), and a prediction is to be made for this specific node, using the information about its environment.

In many cases, the set of observations one learns from is of the form (G, v_i, y_i), where each instance is a different node of exactly the same graph G. This is the case when, for instance, classifying web pages, we take the whole web as their environment.

In a labeled graph, labels are associated with each node or edge. Often these are assumed atomic, being elements of a finite alphabet or real numbers, but they can also be vectors of reals.

Relational Learning

In ▶relational learning, it is assumed that relationships may exist between different instances of the instance space, or an instance may internally consist of multiple objects among which relationships exist.

This essentially corresponds to learning from graphs, except that in a graph only one binary relation exists (the edges E), whereas here there may be multiple relations and they may be non binary. The expressiveness of the two settings is the same, however, as any relation can be represented using only binary relations.

In the attribute-value setting, one typically uses one table where each tuple represents all the relevant information for one observation. In the relational setting, there may be multiple tables, and information on a single instance is contained in multiple tuples, possibly belonging to multiple relations.

Example 1 *Assume we have a database about students, courses, and professors (see Fig. 1). We can define a single observation as all the information relevant to one student, that is: the name, year of entrance, etc. of the student and also the courses they take and the professors teaching these courses.*

Anne	1997
Bernard	1999
Celine	1996
Daniel	1999
Elisa	1997
Fabian	1999

Anne	Algebra	1998	A
Anne	Calculus	1998	B
Bernard	Databases	2000	A
Celine	Biology	1999	B
Celine	Databases	2000	B
Celine	Calculus	1998	A

Algebra
Biology
Calculus
Databases

Adams	Algebra	1998
Adams	Calculus	1999
Baeck	Biology	1999
Cools	Calculus	1998
Cools	Databases	1999

Adams
Baeck
Cools

Observation Language. Figure 1. A small database of students

The most obvious link to the graph representation is as follows: create one node for each tuple, labeled with that tuple, and create a link between two nodes if the corresponding tuples are connected by a foreign key relationship.

Defining a single observation as a set of tuples that are connected through foreign keys in the database corresponds to representing each observation (G, v, y) as (G', v, y), where G' is the connected component of G that contains v. The actual links are usually not explicitly written in this representation, as they are implicit: there is an edge between two tuples if they have the same value for a foreign key attribute.

Inductive Logic Programming

In ▶inductive logic programming, a language based on first order logic is used to represent the observations. Typically, an observation is then represented by a *ground fact*, which basically corresponds to a single tuple in a relational database. In some settings an observation is represented by an *interpretation*, a set of ground facts, which corresponds to the set of tuples mentioned in the previous subsection.

While the target variable can always be represented as an additional attribute, ILP systems often learn from examples and counterexamples of a concept. The target variable is then implicit: it is true or false depending on whether the example is in the positive or negative set, but it is not explicitly included in the fact.

Typical for the inductive logic programming setting is that the input of a system may contain, besides the observations, background knowledge about the application domain. The advantage of the ILP setting is that no separate language is needed for such background knowledge: the same first order logic-based language can be used for representing the observations as well as the background knowledge.

Example 2 *Take the following small dataset:*

```
sibling(bart,lisa).
sibling(lisa,bart).
:- sibling(bart, bart).
:- sibling(lisa, lisa).
father(homer, bart).
mother(marge, bart).
father(homer, lisa).
mother(marge, lisa).
```

There are positive and negative (preceded by `:-`*) examples of the Sibling relation. The following hypothesis might be learned:*

```
sibling(X,Y) :- father(Z,X),
  father(Z,Y), X ≠ Y.
sibling(X,Y) :- mother(Z,X),
  mother(Z,Y), X ≠ Y.
```

If the following clauses as included as background knowledge:

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
```

then the same ILP system might learn the following more compact definition:

```
sibling(X,Y) :- parent(Z,X),
parent(Z,Y), X ≠ Y.
```

Further Reading

Most of the literature on hypothesis and observation languages is found in the area of inductive logic programming. Excellent starting points to become familiar with this field are *Relational Data Mining* by Lavrač and Džeroski (2001) and *Logical and Relational Learning* by De Raedt (2008).

De Raedt (1998) compares a number of different observation and hypothesis languages with respect to their expressiveness, and indicates relationships between them.

Cross References

▶Hypothesis Language
▶Inductive Logic Programming
▶Relational Learning

Recommended Reading

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In D. Page (Ed.), *Proceedings of the eighth international conference on inductive logic programming. Lecture notes in artificial intelligence* (Vol. 1446, pp. 1–8). Berlin: Springer.

De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer. vfill

Occam's Razor

Geoffrey I. Webb
Monash University, Victoria 3800, Australia

Synonyms

Ockham's razor

Definition

Occam's Razor is the maxim that "entities are not to be multiplied beyond necessity," or as it is often interpreted in the modern context "of two hypotheses H and H', both of which explain E, the simpler is to be preferred" (Good, 1977).

Motivation and Background

Most attempts to learn a ►model from ►data confront the problem that there will be many models that are consistent with the data. In order to learn a single model, a choice must be made between the available models. The factors taken into account by a learner in choosing between models are called its ►inductive biases (Mitchell, 1980). A preference for simple models is a common inductive bias and is embodied in many learning techniques including ►pruning, ►minimum message length and ►minimum description length. ►Regularization is also sometimes viewed as an application of Occams' Razor.

Occam's Razor is an imperative, rather than a proposition. That is, it is neither true nor false. Rather, it is a call to act in a particular way without making any claim about the consequences of doing so. In machine learning the so-called *Occam thesis* is sometimes assumed, that

- given a choice between two plausible classifiers that perform identically on the training set, the simpler classifier is expected to classify correctly more objects outside the training set (Webb, 1996).

While there are many practical advantages in having an inductive bias toward simple models, there remains controversy as to whether the Occam thesis is true (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987; Domingos, 1999; Webb, 1996).

Recommended Reading

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, *24*(6), 377–380.

Domingos, P. (1999). The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, *3*(4), 409–425.

Good, I. J. (1977). Explicativity: A mathematical theory of explanation with statistical applications. *Proceedings of the Royal Society of London Series A*, *354*, 303–330.

Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Tech. Rep. CBM-TR-117. Rutgers University, Department of Computer Science.

Webb, G. I. (1996). Further experimental evidence against the utility of Occams razor. *Journal of Artificial Intelligence Research*, *4*, 397–417; Menlo Park: AAAI Press.

Ockham's Razor

►Occam's Razor

Offline Learning

►Batch Learning

One-Step Reinforcement Learning

►Associative Reinforcement Learning

Online Learning

Peter Auer
University of Leoben,
Leoben, Austria

Synonyms

Mistake-bounded learning; Perceptron; Prediction with expert advice; Sequential prediction

Definition

In the online learning model the learner needs to make predictions about a sequence of instances, one after the other, and receives a reward or loss after each prediction. Typically, the learner receives a description of

the instance before making a prediction. The goal of the learner is to maximize the accumulated reward (or equivalently minimize the accumulated losses).

The online learning model is essentially a worst-case model of learning, as it makes no statistical assumptions on how the sequence of inputs and rewards is generated. In particular, it is not assumed that inputs and observations are generated by a probability distribution. In contrast, they might be generated by an adversary who tries to fool the learner.

To compensate for the adversarial nature of the model, in most cases the performance guarantees for online learning algorithms are relative to the performance of the best predictor from a certain class. Often these performance guarantees are quite strong, showing that the learner can do nearly as well as the best predictor from a large class of predictors.

Motivation and Background

Online learning is one of the main models of learning theory, complementing the statistical approach of the ►PAC learning model by making no statistical assumptions. The distinctive properties of the online learning model are:

- Learning proceeds in trials.
- There is no designated learning phase, but performance of the learner is evaluated for each trial.
- No assumptions on the generation of the inputs to the learner are made; they may depend even on previous predictions of the learner.
- In most cases no assumptions on the losses or rewards are made; they may be selected by an adversary.
- The sequential predictions model an interaction between the learner and its environment.
- Performance guarantees for learning algorithms are typically in terms of the performance of the best predictor from some given class, after some number of observations.

The first explicit models of online learning were proposed by Angluin (1988) and Littlestone (1988), but related work on repeated games by Hannan (1957) dates back to 1957. Littlestone proposed online learning as a sequence of trials, in each of which the learner receives some input, makes a prediction of the associated output, and receives the correct output. It was assumed that some function from a known class maps the inputs to correct outputs. The performance of the learner is measured by the number of mistakes made by a learner, before it converges to the correct predictor. Angluin's equivalence query model of learning is formulated differently but is essentially equivalent to Littlestone's model.

Many (including Littlestone (1991), Vovk (1990), and Littlestone and Warmuth (1994)) later removed the restriction that there must be a function that correctly predicts all the outputs. In their setting the learner competes with the best predictor from a given class. As the class of predictors can be seen as a set of experts advising the learner about the correct predictions, this led to the term "prediction with expert advice." A comprehensive treatment of binary predictions with expert advice can be found in Cesa-Bianchi et al. (1997). Relations of online learning to several other fields (e.g., compression, competitive analysis, game theory, and portfolio selection) are discussed in the excellent book on sequential prediction by Cesa-Bianchi and Lugosi (2006).

Structure of Learning System

The online learning model is formalized as follows. In each trial $t = 1, 2, \ldots$, the learner

1. Receives input $x_t \in X$
2. Makes prediction $y_t \in Y$
3. Receives response $z_t \in Z$
4. Incurs loss $\ell_t = \ell(y_t, z_t)$

where $\ell : Y \times Z \mapsto \mathbf{R}$ is some loss function. The performance of a learner up to trial T is measured by its accumulated loss $L_T = \sum_{t=1}^{T} \ell_t$.

Performance bounds for online learning algorithms are, typically, in respect to the performance of an optimal predictor (or expert) E^* from some class $\mathcal{E}$, $E^* \in \mathcal{E}$. A predictor E maps the past given by $(x_1, y_1, z_1), \ldots, (x_{t-1}, y_{t-1}, z_{t-1})$ and the current input x_t to a prediction y_t^E. As for the learner, the performance of a predictor is measured by its accumulated loss $L_T^E = \sum_{t=1}^{T} \ell_t^E$, where $\ell_t^E = \ell\left(y_t^E, z_t\right)$. Most bounds for the loss of online algorithms are of the form

$$L_T \le a \min_{E \in \mathcal{E}} L_T^E + b\mathcal{C}(\mathcal{E}),$$

where the constants a and b depend on the loss function and $\mathcal{C}(\mathcal{E})$ measures the complexity of the class of predictors. (e.g., the complexity $\mathcal{C}(\mathcal{E})$ could be $\log|\mathcal{E}|$ for a finite class $\mathcal{E}$.) Often it is possible to trade the constant a against the constant b such that bounds

$$L_T \le L_T^* + o\left(L_T^*\right)$$

can be achieved, where $L_T^* = \min_{E\in\mathcal{E}} L_T^E$ is the loss of the best predictor up to time T. These bounds are of particular interest as they show that the loss of the learning algorithm is only little larger than the loss of the best predictor. For such bounds the regret R_T of the learning algorithm,

$$R_T = L_T - L_T^*,$$

is the relevant quantity that measures the cost of not knowing the best predictor in advance. Again, it needs to be emphasized that these bounds hold for any sequence of inputs and responses without any additional assumptions. Such bounds are achieved by online learning algorithms that rely on the outputs of the predictors in $\mathcal{E}$ to form their own predictions.

The next section makes this general definition of online learning more concrete by presenting some important online learning algorithms, and it also discusses the related equivalence query model.

Theory/Solution

The Weighted Majority Algorithm

The weighted majority algorithm developed by Littlestone and Warmuth (1994) is one of the fundamental online learning algorithms, with many relatives using similar ideas. It will be presented for the basic scenario with a finite set of experts $\mathcal{E}$, binary predictions $y_t \in \{0,1\}$, binary responses $z_t \in \{0,1\}$, and the discrete loss which just counts mistakes, $\ell(y,z) = |y-z|$, such that $\ell(y,z) = 0$ if $y = z$ and $\ell(y,z) = 1$ if $y \neq z$. (We will use the terms experts and predictors interchangeably. In the literature finite sets of predictors are mostly called experts.)

The weighted majority algorithm maintains a weight w_t^E for each expert $E \in \mathcal{E}$ that are initialized as $w_1^E = 1$. The weights are used to combine the predictions y_t^E of the experts by a weighted majority vote: $y_t = 1$ if $\sum_E w_t^E y_t^E \ge \frac{1}{2}\sum_E w_t^E$, and $y_t = 0$ otherwise. After receiving the response z_t, the weights of experts that made incorrect predictions are reduced by multiplying with some constant $\beta < 1$, $w_{t+1}^E = \beta w_t^E$ if $y_t^E \neq z_t$, and $w_{t+1}^E = w_t^E$ if $y_t^E = z_t$. As a performance bound for the weighted majority algorithm one can achieve

$$L_T \le 2L_T^* + 2\sqrt{2L_T^*\log|\mathcal{E}|} + 4\log|\mathcal{E}|$$

with $L_T^* = \min_{E\in\mathcal{E}} L_T^E$ and an appropriate β. (Better constants on the square root and the logarithmic term are possible.)

While in this bound the loss of the deterministic weighted majority algorithm is twice the loss of the best expert, the randomized version of the weighted majority algorithm almost achieves the loss of the best expert. Instead of using a deterministic prediction, the randomized weighted majority algorithm tosses a coin and predicts $y_t = 1$ with probability $\sum_E w_t^E y_t^E / \sum_E w_t^E$.

Since a prediction of the randomized algorithm matches the prediction of the deterministic algorithm with probability at least 1/2, an incorrect prediction of the deterministic algorithm implies that the randomized algorithm will predict incorrectly with probability at least 1/2. Thus, the loss of the deterministic algorithm is at most twice the expected loss of the randomized algorithm. This can be used to transfer bounds for the randomized algorithm to the deterministic algorithm.

Below, the following bound will be proved on the expected loss of the randomized algorithm,

$$\mathbf{E}[L_T] \le \frac{\log(1/\beta)}{1-\beta} L_T^* + \frac{1}{1-\beta}\log|\mathcal{E}|. \tag{1}$$

Approximately optimizing for β yields $\beta = 1-\varepsilon$, where $\varepsilon = \min\left\{1/2, \sqrt{2(\log|\mathcal{E}|)/L_T^*}\right\}$, and

$$\mathbf{E}[L_T] \le L_T^* + \sqrt{2L_T^*\log|\mathcal{E}|} + 2\log|\mathcal{E}|. \tag{2}$$

The expectation in these bounds is only in respect to the randomization of the algorithm, no probabilistic assumptions on the experts or the sequence of responses are made. These bounds hold for any set of experts and any fixed sequence of responses. This type of bounds assumes that the sequence of inputs and responses does not depend on the randomization of the algorithm. If the inputs x_t and the responses z_t may depend on the past predictions of the algorithm, $y_1, \ldots, y_{t-1}$, then

L_T^* also becomes a random variable and the following bound is achieved:

$$\mathbf{E}[L_T] \le \mathbf{E}[L_T^*] + \sqrt{2\mathbf{E}[L_T^*]\log|\mathcal{E}|} + 2\log|\mathcal{E}|.$$

It can be even shown that the following similar bound holds with probability $1-\delta$ (in respect to the randomization of the algorithm):

$$L_T \le L_T^* + \sqrt{T\log(|\mathcal{E}|/\delta)}.$$

The proof of bound (1) shows many of the ideas used in the proofs for online learning algorithms. Key ingredients are a potential function and how the changes of the potential function relate to losses incurred by the learning algorithm. For the weighted majority algorithm a suitable potential function is the sum of the weights, $W_t = \sum_E w_t^E$. Then, since the losses are 0 or 1,

$$\frac{W_{t+1}}{W_t} = \frac{\sum_E w_{t+1}^E}{\sum_E w_t^E} = \frac{\sum_E \beta^{\ell_t^E} w_t^E}{\sum_E w_t^E} = \frac{\sum_E \left[1-(1-\beta)\ell_t^E\right] w_t^E}{\sum_E w_t^E}$$
$$= 1-(1-\beta)\frac{\sum_E \ell_t^E w_t^E}{\sum_E w_t^E}.$$

Since the probability that the randomized weighted majority algorithm makes a mistake is given by $\mathbf{E}[\ell_t] = \sum_E \ell_t^E w_t^E / \sum_E w_t^E$, we get by taking logarithms that

$$\log W_{t+1} - \log W_t = \log(1-(1-\beta)\mathbf{E}[\ell_t]) \le -(1-\beta)\mathbf{E}[\ell_t]$$

(since $\log(1-x) \le -x$ for $x \in (0,1)$). Summing over all trials $t = 1,\ldots,T$ we find

$$\log W_{T+1} - \log W_1 \le -(1-\beta)\mathbf{E}[L_t].$$

Since $W_1 = |\mathcal{E}|$ and $W_{T+1} = \sum_E w_{T+1}^E = \sum_E \beta^{L_T^E} \ge \beta^{L_T^*}$, rearranging the terms gives (1).

Extensions and Modifications of the Weighted Majority Algorithm Variants and improved versions of the weighed majority algorithm have been analyzed for various learning scenarios. An excellent coverage of the material can be found in Cesa-Bianchi and Lugosi (2006). This section mentions a few of them.

General loss functions. The analysis of the weighted majority algorithm can be generalized to any convex set of predictions Y and any set of outcomes Z, as long as the loss function $\ell(y,z)$ is bounded and convex in the first argument. Vovk (1998) analyzed for quite general Y, Z, and loss functions ℓ, which constants a and b allow a learning algorithm with loss bound

$$L_T \le aL_T^* + b\log|\mathcal{E}|.$$

Of particular interest is the smallest b for which a loss bound with $a = 1$ can be achieved.

Tracking the best expert and other structured experts. For a large number of experts, the loss bound of the weighted majority algorithm is still interesting since it scales only logarithmically with the number of experts. Nevertheless, the weighted majority algorithm and other online learning algorithms become computationally demanding as they need to keep track of the performance of all experts (computation time scales linearly with the number of experts). If the experts exhibit a suitable structure, then this computational burden can be avoided.

As an example, the problem of tracking the best expert is considered. Let $\mathcal{E}_0$ be a small set of base experts. The learning algorithm is required to compete with the best sequence of at most S experts from $\mathcal{E}_0$: the trials are divided into S periods, and in each period another expert might predict optimally. Thus, the minimal loss of a sequence of S experts is given by

$$L_{T,S}^* = \min_{0=T_0\le T_1\le T_2\le\cdots\le T_S=T} \sum_{i=1}^{S} \min_{E\in\mathcal{E}_0} \sum_{t=T_{i-1}+1}^{T_i} \ell_t^E,$$

where the trials are optimally divided into S periods $[T_{i-1}+1, T_i]$, and the best base expert is chosen for each period. Such sequences of base experts can be seen as experts themselves, but the number of such compound experts is $\binom{T-1}{S-1}|\mathcal{E}_0|^S$ and thus computationally prohibitive. Fortunately, a slightly modified weighted majority algorithm applied to the base experts, achieves almost the same performance as the weighted majority algorithm applied to the compound experts (Herbster & Warmuth, 1998). The modification of the weighted majority algorithm just lower bounds the relative weight of each base expert. This allows the relative weight of a base expert to grow large quickly if this expert predicts best in the current period. Hence, also the learning algorithm will predict almost optimally in each period.

Other examples of structured experts include tree experts and shortest path problems (see Cesa-Bianchi and Lugosi (2006) for further references).

The doubling trick. The optimal choice of β in the performance bound (1) requires knowledge about the loss of the best expert L_T^*. If such knowledge is not available, the doubling trick can be used. The idea is to start with an initial guess $\hat{L}^*$ and choose β according to this guess. When the loss of the best expert exceeds this guess, the guess is doubled, β is modified, and the learning algorithm is restarted. The bound (2) increases only slightly when L_T^* is not known and the doubling trick is used. It can be shown that still

$$\mathbf{E}[L_T] \le L_T^* + c_1\sqrt{L_T^* \log|\mathcal{E}|} + c_2 \log|\mathcal{E}|$$

for suitable constants c_1 and c_2. A thorough analysis of the doubling trick can be found in Cesa-Bianchi et al. (1997). Variations of the doubling trick can be used for many online learning algorithms to "guess" unknown quantities. A drawback of the doubling trick is that it restarts the learning algorithm and forgets about all previous trials. An alternative approach is an iterative adaptation of the parameter β, which can be shown to give better bounds than the doubling trick. The advantage of the doubling trick is that its analysis is quite simple.

Follow the perturbed leader. *Follow the perturbed leader* is a simple prediction strategy that was originally proposed by Hannan (1957). In each trial t, it generates identically distributed random values ψ_t^E for every expert E, adds these random values to the losses of the experts so far, and predicts with the expert that achieves the minimum sum,

$$\hat{E}_t = \arg\min_{E\in\mathcal{E}} L_{t-1}^E + \psi_t^E,$$
$$y_t = y_t^{\hat{E}_t}.$$

For suitably chosen distributions of the ψ_t^E, this simple prediction strategy achieves loss bounds similar to the more involved weighted majority like algorithms.

Prediction with limited feedback and the multiarmed bandit problem. In some online learning scenarios, the learner might not receive the original response z_t but only a projected version $\tilde{z}_t = \zeta(y_t, z_t)$ for some $\zeta : Y \times Z \to \tilde{Z}$. The value of the incurred loss $\ell(y_t, z_t)$ might be unknown to the learner. A general model for this situation is called *prediction with partial monitoring*. With suitable assumptions there are prediction strategies for partial monitoring that achieve a regret of order $O(T^{2/3})$.

A special case of partial monitoring is the multiarmed bandit problem. In the multiarmed bandit problem the learner chooses a prediction $y_t \in Y = \{1, \ldots, K\}$ and receives the loss of the chosen prediction $\ell_t(y_t) = \ell(y_t, z_t)$. The losses of the other predictions, $\ell_t(y)$, $y \ne y_t$, are not revealed to the learner. The goal of the learner is to compete with the loss of the single best prediction, $L_T^* = \min_{y\in Y} L_T^y$, $L_T^y = \sum_{t=1}^T \ell_t(y)$. The multiarmed bandit problem looks very much like the original online learning problem with the predictions $y \in Y$ as experts. But the main difference is that in the multiarmed bandit problem only the loss of the *chosen* expert/prediction is revealed, while in the original online learning problem the losses of *all* experts can be calculated by the learner. Therefore, algorithms for the multiarmed bandit problem estimate the unseen losses and use these estimates to make their predictions. Since accurate estimates need a sufficient amount of data, this leads to a trade-off between choosing the (apparently) best prediction to minimize loss, and choosing another prediction for which more data need to be collected. This exploration–exploitation trade-off also appears elsewhere in online learning, but it is most clearly displayed in the bandit problem. An algorithm that deals well with this trade-off is again a simple variant of the weighted majority algorithm. This algorithm does exploration trials with some small probability, and in such exploration trials it chooses a prediction uniformly at random. This algorithm has been analyzed in Auer, Cesa-Bianchi, Freund, and Schapire (2002) for gains instead of losses. Formally, this is equivalent to considering negative losses, $\ell \in [-1, 0]$, with the equivalent gain $g = -\ell$. For losses $\ell \in [-1, 0]$ a bound for the algorithm is

$$\mathbf{E}[L_T] \le L_T^* + 3\sqrt{K|L_T^*|\log K},$$

which for gains translates into

$$\mathbf{E}[G_T] \ge G_T^* - 3\sqrt{KG_T^* \log K}$$

where G_T and G_T^* denote the accumulated gains. Compared with (2), the regret increases only by a factor of $\sqrt{K}$. Auer et al. (2002) show that the order of the regret is essentially optimal. They also present similar bounds that hold with high probability, and analyze several extensions of the bandit problem.

The Perceptron Algorithm This section considers an example for an online learning algorithm that competes with a *continuous* set of experts, in contrast to the *finite* sets of experts considered so far. This algorithm – the perceptron algorithm (Rosenblatt 1958)– was among the first online learning algorithms developed. Another of this early online learning algorithms with a continuous set of experts is the Winnow algorithm by Littlestone (1988). A unified analysis of these algorithms can be found in Cesa-Bianchi and Lugosi (2006). This analysis covers a large class of algorithms, in particular the p-norm perceptrons (Grove, Nittlestone & Schuurmans, 2001), which smoothly interpolate between the perceptron algorithm and Winnow.

The perceptron algorithm aims at learning a linear classification function. Thus inputs are from a Euclidean space, $X = \mathbf{R}^d$, the predictions and responses are binary, $Y = Z = \{0,1\}$, and the discrete misclassification loss is used. Each expert is a linear classifier, represented by its weight vector $v \in \mathbf{R}^d$, whose linear classification is given by $\Phi_v : X \to \{0,1\}$, $\Phi_v(x) = 1$ if $v \cdot x \geq 0$ and $\Phi_{v,\theta}(x) = 0$ if $v \cdot x < 0$.

The perceptron algorithm maintains a weight vector $w_t \in \mathbf{R}^d$ that is initialized as $w_1 = (0, \ldots, 0)$. After receiving input x_t, the perceptron's prediction is calculated using this weight,

$$y_t = \Phi_{w_t}(x_t),$$

and the weight vector is updated,

$$w_{t+1} = w_t + \eta(z_t - y_t)x_t,$$

where $\eta > 0$ is a learning rate parameter. Thus, if the prediction is correct, $y_t = z_t$, then the weights are not changed. Otherwise, the product $w_{t+1} \cdot x_t$ is moved into the correct direction: since $w_{t+1} \cdot x_t = w_t \cdot x_t + \eta(z_t - y_t)||x_t||^2$, $w_{t+1} \cdot x_t > w_t \cdot x_t$ if $y_t = 0$ but $z_t = 1$, and $w_{t+1} \cdot x_t < w_t \cdot x_t$ if $y_t = 1$ but $z_t = 0$.

It may be assumed that the inputs are normalized, $||x_t|| = 1$, otherwise a normalized x_t can be used in the update of the weight vector. Furthermore, it is noted that the learning rate η is irrelevant for the performance of the perceptron algorithm, since it scales only the size of the weights but does not change the predictions. Nevertheless, the learning rate is kept since it will simplify the analysis.

Analysis of the perceptron algorithm. To compare the perceptron algorithm with a fixed (and optimal) linear classifier v a potential function $||w_t - v||^2$ is again used. For the change of the potential function when $y_t \neq z_t$, one finds

$$\begin{aligned}
&||w_{t+1} - v||^2 - ||w_t - v||^2 \\
&= ||w_t + \eta(z_t - y_t)x_t - v||^2 - ||w_t - v||^2 \\
&= ||w_t - v||^2 + 2\eta(z_t - y_t)(w_t - v) \cdot x_t \\
&\quad + \eta^2(z_t - y_t)^2||x_t||^2 - ||w_t - v||^2 \\
&= 2\eta(z_t - y_t)(w_t \cdot x_t - v \cdot x_t) + \eta^2.
\end{aligned}$$

Since $w_t \cdot x_t < 0$ if $y_t = 0$ and $w_t \cdot x_t \geq 0$ if $y_t = 1$, we get $(z_t - y_t)(w_t \cdot x_t) \leq 0$ and

$$||w_{t+1} - v||^2 - ||w_t - v||^2 \leq -2\eta(z_t - y_t)(v \cdot x_t) + \eta^2.$$

Analogously, the linear classifier v makes a mistake in trial t if $(z_t - y_t)(v \cdot x_t) < 0$, and in this case $-(z_t - y_t)(v \cdot x_t) \leq ||v||$. Hence, summing over all trials (where $y_t \neq z_t$) gives

$$\begin{aligned}
&||w_{T+1} - v||^2 - ||w_1 - v||^2 \\
&\quad \leq -2\eta \sum_{t:\ell_t=1,\ell_t^v=0} |v \cdot x_t| + 2\eta||v||L_T^v + \eta^2 L_T,
\end{aligned}$$

where the sum is over all trials where the perceptron algorithm makes a mistake but the linear classifier v makes no mistake. To proceed, it is assumed that for the correct classifications of the linear classifier v, the product $v \cdot x_t$ is bounded away from 0 (which describes the decision boundary). It is assumed that $|v \cdot x_t| \geq \gamma_v > 0$. Then

$$\begin{aligned}
||w_{T+1} - v||^2 - ||w_1 - v||^2 \leq &-2\eta\gamma_v (L_T - L_T^v) \\
&+ 2\eta||v||L_T^v + \eta^2 L_T,
\end{aligned}$$

and

$$L_T(2\eta\gamma_v - \eta^2) \leq ||v||^2 + L_T^v(2\eta\gamma_v + 2\eta||v||),$$

since $\|w_{T+1} - v\|^2 \geq 0$ and $w_1 = (0, \ldots, 0)$. For $\eta = \gamma_v$ the following loss bound for the perceptron algorithm is achieved:

$$L_T \leq \|v\|^2/\gamma_v^2 + 2L_T^v(1 + \|v\|/\gamma_v).$$

Thus, the loss of the perceptron algorithm does not only depend on the loss of a (optimal) linear classifier v, but also on the gap by which the classifier can separate the inputs with $z_t = 0$ from the inputs with $z_t = 1$. The size of this gap is essentially given by $\gamma_v/\|v\|$.

Relation between the perceptron algorithm and support vector machines. The gap $\gamma_v/\|v\|$ is the quantity maximized by support vector machines, and it is the main factor determining the prediction accuracy (in a probabilistic sense) of a support vector machine. It is not coincidental that the same quantity appears in the performance bound of the perceptron algorithm, since it measures the difficulty of the classification problem.

As for support vector machines, kernels $K(\cdot, \cdot)$ can be used in the perceptron algorithm. For that, the dot product $w_t \cdot x_t$ is replaced by the kernel representation $\sum_{\tau=1}^{t-1}(z_\tau - y_\tau)K(x_\tau, x)$. Obviously this has the disadvantage that all previous inputs for which mistakes were made must be kept available.

Learning with Equivalence Queries In the *learning with equivalence queries* model (Angluin, 1988) the learner has to identify a function $f : X \to \{0,1\}$ from a class $\mathcal{F}$ by asking equivalence queries. An equivalence query is a hypothesis $h : X \to \{0,1\}$ about the function f. If $h = f$ then the answer to the equivalence query is YES, otherwise a counterexample $x \in X$ with $f(x) \neq h(x)$ is returned. The performance of a learning algorithm is measured by the number of counter examples received as response to equivalence queries.

The equivalence query model is essentially equivalent to the online learning model with $\mathcal{F}$ as the set of experts, $Y = Z = \{0,1\}$, and the discrete misclassification loss. First, it is considered how a learner for the equivalence query model can be used as a learner in the online learning model: For making a prediction y_t, the equivalence query learner uses its current hypothesis, $y_t = h_t(x_t)$. If the prediction is correct, the hypothesis is not changed. If the prediction is incorrect, the input x_t is used as a counterexample for the hypothesis h_t.

Second, it is considered how a deterministic online learner can be used in the equivalence query model. The current prediction function of the online learner that maps inputs to predictions, can be used as a hypothesis h_t in the equivalence query model. A counterexample x_t is interpreted as an input for which the response z_t disagrees with the prediction y_t of the online learner. Thus the online learner might update its prediction function, which gives a new hypothesis for an equivalence query.

These reductions show that the counterexamples of the equivalence query learner and the mistakes of the online learner coincide. Thus, the performance bounds for the equivalence query model and the online model are the same. (Here it is assumed that there is an expert that incurs no loss. If this is not the case, then an extension of the equivalence query model can be considered, where a number of equivalence queries might be answered with incorrect counterexamples.)

Cross References

►Incremental Learning

Recommended Reading

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*, 319–342.

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. (2002). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing, 32*, 48–77.

Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D., Schapire, R., & Warmuth, M. (1997). How to use expert advice. *Journal of the ACM, 44*, 427–485.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. New York, USA: Cambridge University Press.

Grove, A. J., Nittlestone, N., & Schuurmans, D. (2001). General convergence results for linear discriminant updates. *Machine Learning, 43*, 173–210.

Hannan, J. (1957). Approximation to Bayes risk in repeated play. *Contributions to the Theory of Games, 3*, 97–139.

Herbster, M., & Warmuth, M. (1998). Tracking the best expert. *Machine Learning, 32*, 151–178.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*, 285–318.

Littlestone, N. (1991). Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proceedings of the fourth annual workshop on computational learning theory, Santa Cruz, California* (pp. 147–156). San Francisco: Morgan Kaufmann.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation, 108*, 212–261.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*, 386–408.

Vovk, V. (1990). Aggregating strategies. In *Proceedings of third annual workshop on computational learning theory, Rochester, New York* (pp. 371–386). San Francisco: Morgan Kaufmann.
Vovk, V. (1998). A game of prediction with expert advice. *Journal of Computer and System Sciences, 56*, 153–173.

Ontology Learning

Different approaches have been used for building ontologies, most of them to date mainly using manual methods (►Text Mining for the Semantic Web). An approach to building ontologies was set up in the CYC project, where the main step involved manual extraction of common sense knowledge from different sources. Ontology construction methodologies usually involve several phases including *identifying the purpose of the ontology* (why to build it, how will it be used, the range of the users), *building the ontology*, *evaluation and documentation*. Ontology learning relates to the phase of building the ontology using semiautomatic methods based on text mining or machine learning.

Opinion Mining

Opinion mining is the application of mining methods concerned not with the topic a document is about, but with the opinion it expresses. Opinion mining builds on techniques from ►text mining, ►information retrieval, and computational linguistics. It is especially popular for analyzing documents that are often or even by definition opinionated, including blogs (►text mining for news and blogs analysis) and online product reviews.

Opinion mining is also known as *sentiment analysis* (*sentiment mining*, *sentiment classification*, ...) or *opinion extraction*.

Optimal Learning

►Bayesian Reinforcement Learning

OPUS

►Rule Learning

Ordered Rule Set

►Decision List

Ordinal Attribute

An *ordinal attribute* classifies data into categories that can be ranked. However, the differences between the ranks cannot be calculated by arithmetic. See ►Attribute and ►Measurement Scales.

Out-of-Sample Data

Out-of-sample data are data that were not used to learn a ►model. ►Holdout evaluation creates out-of-sample data for evaluation purposes.

Out-of-Sample Evaluation

Definition

Out-of-sample evaluation refers to ►algorithm evaluation whereby the learned model is evaluated on ►out-of-sample data. Out-of-sample evaluation provides an unbiased estimate of learning performance, in contrast to ►in-sample evaluation.

Cross References

►Algorithm Evaluation

Overall and Class-Sensitive Frequencies

The underlying idea for learning strategies processing ►missing attribute values relies on the class distribution; i.e., the class-sensitive frequencies are utilized. As soon as we substitute a missing value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the overall (class-independent) frequencies are applied within classification.

Overfitting

Geoffrey I. Webb
Monash University, Victoria, Australia

Synonyms

Overtraining

Definition

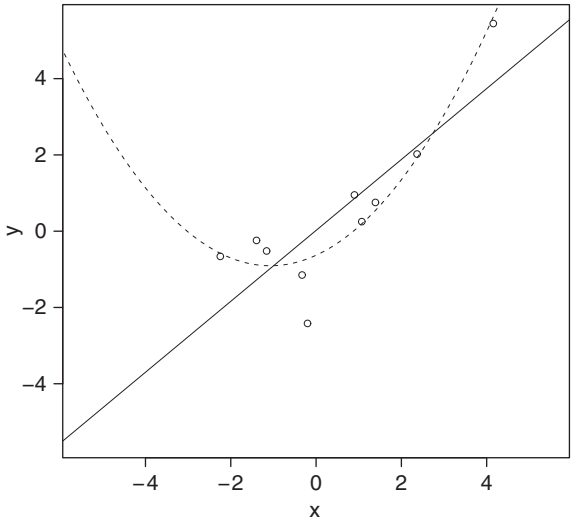
A ▶model *overfits* the ▶training data when it describes features that arise from noise or variance in the data, rather than the underlying distribution from which the data were drawn. Overfitting usually leads to loss of ▶accuracy on ▶out-of-sample data.

Discussion

In general there is a trade-off between the size of the space of distinct models that a ▶learner can produce and the risk of overfitting. As the space of models between which the learner can select increases, the risk of overfitting will increase. However, the potential for finding a model that closely fits the true underling distribution will also increase. This can be viewed as one facet of the ▶bias and variance trade-off.

Figure 1 illustrates overfitting. The points are drawn randomly from a distribution in which $y = x + \varepsilon$, where ε is random noise. The best single line fit to this distribution is $y = x$. ▶Linear regression finds a model $y = 0.02044 + 0.92978 \times x$, shown as the solid line in Fig. 1. In contrast, second degree polynomial regression finds the model $-0.6311 + 0.5128 \times x + 0.2386 \times x^2$, shown as the dashed line. The space of second degree polynomial models is greater than that of linear models, and so the second degree polynomial more closely fits the example data, returning the lower ▶squared error. However, the linear model more closely fits the true distribution and is more likely to obtain lower squared error on future samples.

While this example relates to ▶regression, the same effect also applies to ▶classification problems. For example, an overfitted ▶decision tree may include splits that reflect noise rather than underlying regularities in the data.



Overfitting. Figure 1. Linear and polynomial models fitted to random data drawn from a distribution for which the linear model is a better fit

The many approaches to avoiding overfitting include

- Using low ▶variance learners;
- ▶Minimum Description Length and ▶Minimum Message Length techniques
- ▶Pruning
- ▶Regularization
- ▶Stopping criteria

Cross References

▶Bias and Variance
▶Minimum Description Length
▶Minimum Message Length
▶Pruning
▶Regularization

Overtraining

▶Overfitting