B

Backprop

►Backpropagation

Backpropagation

Paul Munro
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Backprop; BP; Generalized delta rule

Definition

Backpropagation of error (henceforth *BP*) is a method for training feed-forward neural networks see ►Artificial Neural Networks. A specific implementation of BP is an iterative procedure that adjusts network weight parameters according to the gradient of an error measure. The procedure is implemented by computing an error value for each output unit, and by *backpropagating* the error values through the network.

Characteristics

Feed-Forward Networks

A feed-forward neural network is a mathematical function that is composed of constituent "semi-linear" functions constrained by a feed-forward network architecture, wherein the constituent functions correspond to nodes (often called *units* or *artificial neurons*) in a graph, as in Fig. 1. A feedfoward network architecture has a connectivity structure that is an *acyclic graph*; that is, there are no closed loops.

In most cases, the unit functions have a finite range such as $[0,1]$. Thus, the network maps $\mathbb{R}^N$ to $[0,1]^M$, where N is the number of input values and M is the number of output units. Let $FanIn(k)$ refer to the set of units that provide input to unit k, and let $FanOut(k)$ denote the set of units that receive input from unit k.

In an acyclic graph, at least one unit has a FanIn that is the null set. These are the *input units*; the activity of an input unit is not computed; rather it is set to a value external to the network (i.e., from the training data). Similarly, at least one unit has a null FanOut set. Such units typically represent the output of the network; i.e., this set of values is the result of the network computation. Intermediate units (often called *hidden units*) receive input from other units and project outputs to other computational units.

For the BP procedure, the activity of each unit is computed in two steps:

Linear step: the activities of the FanIn are each multiplied by an independent "weight" parameter, to which a "bias" parameter is added; each computational unit has a single bias parameter, independent of the other units. Let this sum be denoted x_k for unit k.

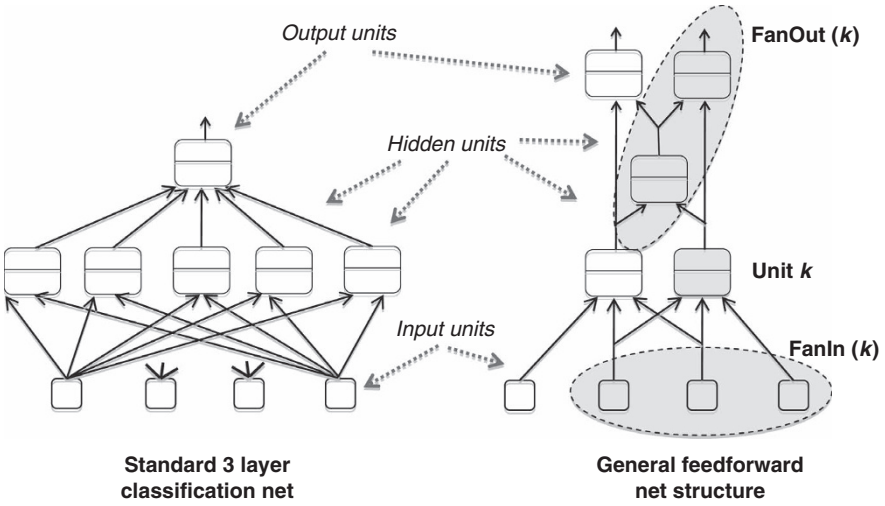
Nonlinear step: The activity a_k of unit k is a differentiable nonlinear function of x_k. A favorite function is the logistic $a = 1/(1 + \exp(-x))$, because it maps the range $[-\infty, +\infty]$ to $[0,1]$ and its derivative has properties conducive to the implementation of BP.

$$a_k = f_k(x_k); \quad \text{where } x_k = b_k + \sum_{j \in \text{FanIn}(k)} w_{kj} s_j$$

Gradient Descent

Derivation of BP is a direct application of the *gradient descent* approach to optimization and is dependent on a definition of network error, a function of the actual network response to a stimulus, $\mathbf{r}(\mathbf{s})$ and the target $\mathbf{T}(\mathbf{s})$. The two most common error functions are the summed squared error (SSE) and the cross entropy error (CE) (CE error as defined here is based on the presumption that the output values are in the range [0, 1]. Likewise

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



Backpropagation. Figure 1. Two networks are shown. Input units are shown as simple squares at the bottom of each figure. Other units are computational (designated by a horizontal line). Left: A standard 3-layer network. Four input units project to five hidden units, which in turn project to a single output unit. Not all connections are shown. Such a network is commonly used for classification tasks. Right: An example of a feed-forward network with four inputs, three hidden units, and two outputs

for the target values; this is often used for classification tasks, wherein target values are set to the endpoints of the range, 0 and 1).

$$E^{\mathrm{SSE}} \equiv \sum_{\substack{i \in \mathrm{Outut} \\ \mathbf{s} \in \mathrm{Train}}} \left(T_i(\mathbf{s}) - r_i(\mathbf{s})\right)^2$$

$$E^{\mathrm{CE}} \equiv \sum_{\substack{i \in \mathrm{Outut} \\ \mathbf{s} \in \mathrm{Train}}} \left[T_i(\mathbf{s}) \ln\left(r_i(\mathbf{s})\right) - \left(1 - T_i(\mathbf{s})\right) \ln\left(1 - r_i(\mathbf{s})\right)\right]$$

Each weight parameter, w_{ij} (the weight of the connection from j to i), is updated by an amount proportional to the negative gradient of the error measure with respect to that parameter:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}},$$

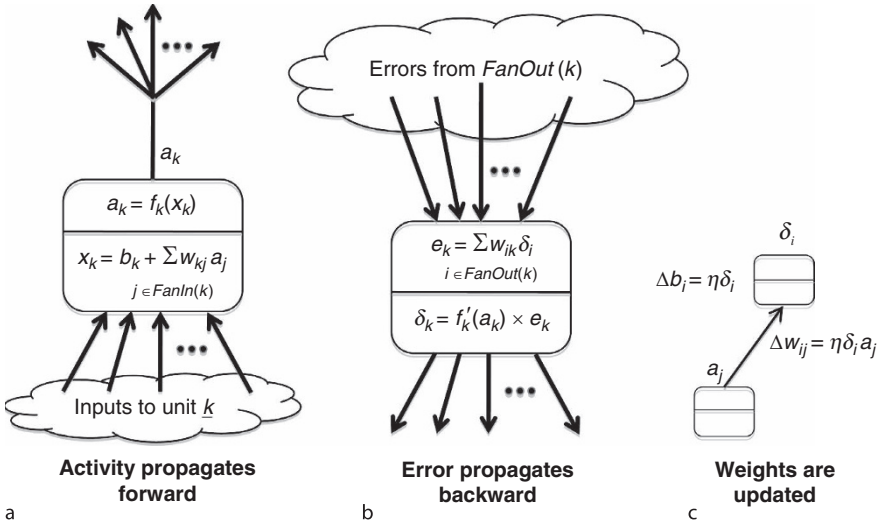
where the *step size*, η, modulates the intrinsic tradeoff between smooth convergence of the weights and the speed of convergence; in the regime where η is small, the system is well-behaved and converges smoothly, but slowly, and for larger η, the system may learn some subsets of the training set faster at the expense of smooth convergence on all patterns in the set. Thus, η is also called the *learning rate*.

Implementation

Several aspects of the feed-forward network must be defined prior to running a BP program, such as the configuration of the hidden units, the initial values of the weights, the functions they will compute, and the numerical representation of the input and target data. There are also parameters of the learning algorithm that must be chosen, such as the value of η and the form of the error function.

The weight and bias parameters are set to their initial values (these are usually random within specified limits). BP is implemented as an iterative process as follows:

1. A stimulus-target pair is drawn from the training set.
2. The activity values for the units in the network are computed for all the units in the network in a forward fashion from input to output (Fig. 2a).
3. The network output values are compared to the target and a *delta* (δ) value is computed for each output unit based on the difference between the target and the actual output response value.



Backpropagation. Figure 2. With each iteration of the backprop algorithm, (a) An activity value is computed for every unit in the network from the input to the output. (b) The network output is compared with the target. The error e_k for output unit k is defined as $(T_k - r_k)$. A value δ_k is computed for each output unit by multiplying e_k by the derivative of the activity function. For hidden units, the error is propagated backward using the weights. (c) The weight parameters w_{ij} are updated in proportion to the product of δ_i and a_j

4. The deltas are propagated backward through the network using the same weights that were used to compute the activity values (Fig. 2b).
5. Each weight is updated by an amount proportional to the product of the downstream delta value and the upstream activity (Fig. 2c).

The procedure can be run either in an *online* mode or *batch* mode. In the online mode, the network parameters are updated for each stimulus-target pair. In the batch mode, the weight changes are computed and accumulated over several iterations without updating the weights until a large number (B) of stimulus-target pairs have been processed (often, the entire training set), at which the weights are updated by the accumulated amounts.

$$\text{online}: \quad \Delta w_{ij}(t) = \eta \delta_i(t) a_j(t) \quad \Delta b_i(t) = \eta \delta_i(t)$$

$$\text{batch}: \quad \Delta w_{ij}(t+B) = \sum_{s=t-1}^{t+B} \eta \delta_i(s) a_j(s)$$

$$\Delta b_i(t+T) = \sum_{s=t+1}^{t-B} \eta \delta_i(s)$$

Classification Tasks with BP

The simplest and most common classification function returns a binary value, indicating membership in a particular class. The most common network architecture for a task of this kind is the three-layer network of Fig. 1 (left), with training values of 0 and 1. For classification tasks, the cross entropy error function generally gives significantly faster convergence. After training, the network is in test mode or production mode, and the responses are in the continuous range [0, 1]; the response must thus be interpreted. The value of the response could be interpreted as a probability or fuzzy Boolean value. Often, however, a single threshold is applied to give a binary answer. A double threshold is sometimes used, with the midrange defined as "uncertain."

Curve Fitting with BP

A feed-forward network can be trained to approximate any function, given the sufficient hidden units. The range of the output unit(s) must be capable of generating activity values in the required range. In order to accommodate an arbitrary range uniformly, a linear

function is advisable for the output units, and the SSE function is the basis for gradient descent.

The Autoencoder Architecture

The autoencoder is a network design in which the target pattern is identical to the input pattern. The hidden units are configured such that there is a "bottleneck layer" of units that is smaller than the input layer, through which information flows; i.e., there are no connections bypassing the bottleneck. Thus, any information necessary to reconstruct the input pattern at the output layer must be represented at the bottleneck. This approach has been successfully applied as an approach to *nonlinear dimensionality reduction* (e.g., Demers & Cottrell, 1993). It bears notable similarities and differences to linear techniques, such as ►*principal components analysis* (*PCA*).

Prediction with BP

The plain "vanilla" BP propagates input to output with no explicit representation of time. Several approaches to processing of temporal patterns have been put forward. Most prominent among these are:

Time delay neural network. In this approach, the input stimulus is simply a sample of a time varying signal. The input patterns are typically generated by a sliding window of samples over time or over a sequence.

►Simple recurrent network (Elman, 1990). A sequence of stimulus patterns is presented as input for the network, which has a single hidden layer design. With each iteration, the input is augmented by a secondary set of input units whose activity is a copy of the hidden layer activity from the previous iteration. Thus, the network is able to maintain a representation of the recent history of network stimuli.

Backpropagation through time (Rumelhart, Hinton, & Williams, 1986). A recurrent network (i.e., a cyclic network) is "unfolded in time" by forming a large multilayer network, in which each layer is a copy of the entire network shifted in time. Thus, the number of layers limits the temporal window available to the network.

Recurrent backpropagation (Pineda, 1989). An acyclic network is run with activity propagation and error propagation, until variables converge. Then the weights are updated.

Cognitive Modeling with BP

Interest in BP as a training technique for classifiers has waned somewhat since the introduction of ►*Support vector machines* (SVMs) in the mid 1990s. However, the influence of BP as an approach to modeling cognitive processes, including perception, concept learning, spatial cognition, and language learning, remains strong. Analysis of hidden unit representations (e.g., using clustering techniques) has given insight into plausible intermediate processes that may underlie cognitive phenomena. Also, many cognitive models trained with BP have exhibited time courses consistent with stages of human learning.

Biological Inspiration and Plausibility

The "connectionist" approach to modeling cognition is based on "neural network" models, which have been touted as "biologically inspired" since their inception. The similarities and differences between connectionist architectures and living brains have been exhaustively debated. Like the brain, the models consist of elements that are extremely limited, computationally. Computational power is derived by several units in network architecture. However, there are compelling differences as well. For example, the temporal dynamics in biological neurons is far more complex than the simple functions used in connectionist networks. It remains unclear what level of neurobiological detail is relevant to understand the cognitive functions.

Shortcomings of BP

The BP method is notorious for convergence problems. An inherent problem of gradient descent approaches to optimization is the issue of locally optimal values. Seeking a minimum value be heading downhill is like water running downhill. Not all water reaches the lowest point (sea level). Water that flows into a mountain lake has landed in a local minimum, a region that is bounded by higher ground.

Even when BP converges to a global minimum (or a local minimum that is "good enough"), it is sometimes very slow. The convergence properties of BP depend on the learning rate and random factors, such as the initial weight and bias values.

Another difficulty with BP is the selection of a network structure. The number of hidden units and the

interconnectivity among them has a strong influence on both the generalization performance and the convergence time. Since the nature of this influence is poorly understood, the design of the network is left to guesswork. The standard approach is to use a single hidden layer (as in Fig. 1, left), which has the advantage of relatively fast convergence.

History

The idea of training a multilayered network using error propagation was originated by Frank Rosenblatt (1958, 1962). However, he was unable to apply gradient descent because he was using linear threshold functions that were not differentiable; therefore, the technique of gradient descent was unavailable. He developed a technique known as the perceptron learning rule that is only applicable to two layer networks (no hidden units). Without hidden units, the computational power of the network is severely reduced. Work in the field virtually stopped with the publication of *Perceptrons* (Minsky & Papert, 1969). The backpropagation procedure was first published by Werbos (1974), but did not receive significant recognition until it was put forward by Rumelhart et al. (1986).

Cross References

►Artificial Neural Networks

Recommended Reading

Demers, D., & Cottrell, G. (1993). Non-linear dimensionality reduction. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems* (Vol. 5). San Mateo, CA: Morgan Kaufmann.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Pineda, F. J. (1989). Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, *1*, 161–172.

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

Rosenblatt, F. (1962). *Principles of statistical neurodynamics*. Washington, DC: Spartan.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Ph.D. thesis, Harvard University, Cambridge.

Bagging

Bagging is an ►ensemble learning technique. The name "Bagging" is an acronym derived from *B*ootstrap *AGG*regat*ING*. Each member of the ensemble is constructed from a different training dataset. Each dataset is a ►bootstrap sample from the original. The models are combined by a uniform average or vote. Bagging works best with ►unstable learners, that is those that produce differing generalization patterns with small changes to the training data. Bagging therefore tends not to work well with linear models. See ►ensemble learning for more details.

Bake-Off

Definition

Bake-off is a disparaging term for experimental evaluation of multiple learning algorithms by a process of applying each algorithm to a limited set of benchmark problems.

Cross References

►Algorithm Evaluation

Bandit Problem with Side Information

►Associative Reinforcement Learning

Bandit Problem with Side Observations

►Associative Reinforcement Learning

Basic Lemma

►Symmetrization Lemma

Basket Analysis

Hannu Toivonen
University of Helsinki, Helsinki, Finland

Synonyms
Market basket analysis

Definition
The goal of basket analysis is to utilize large volumes of electronic receipts, stored at the checkout terminals of supermarkets, for better understanding of customer behavior.

While many forms of learning and mining can be applied to market baskets, the term usually refers to some variant of ►association rule mining. In the basic setting, each market basket constitutes an example essentially defined by the set of purchased products. Association rules then identify sets of items that tend to be bought together. A classical, anecdotal discovery from supermarket data is that "if a basket contains diapers then it often also contains beer." This example illustrates several potential benefits of market basket analysis by association rules: simplicity and understandability of the results, actionability of the results, and a form of nonsupervised approach where the consequent of the rule has not been fixed by the user.

Association rules are often found with the ►Apriori algorithm, and are based on ►frequent itemsets.

Cross References
►Apriori Algorithm
►Association Rule
►Frequent Itemset
►Frequent Pattern

Batch Learning

Synonyms
Offline Learning

Definition
A *batch learning algorithm* accepts a single input that is a set or sequence of observations. The algorithm produces its ►model, and does no further learning. Batch learning stands in contrast to ►online learning.

Baum–Welch Algorithm

The Baum–Welch algorithm is used for computing maximum likelihood estimates and posterior mode estimates for the parameters (transition and emission probabilities) of a HMM, when given only output sequences (emissions) as training data.

The Baum–Welch algorithm is a particular instantiation of the expectation-maximization algorithm, suited for HMMs.

Bayes Adaptive Markov Decision Processes

►Bayesian Reinforcement Learning

Bayes Net

►Bayesian Network

Bayes Rule

Geoffrey I. Webb
Monash University

Definition
Bayes rule provides a decomposition of a conditional probability that is frequently used in a family of learning techniques collectively called *Bayesian Learning*. Bayes rule is the equality

$$P(z \mid w) = \frac{P(z)P(w \mid z)}{P(w)} \quad (1)$$

$P(w)$ is called the *prior probability*, $P(w \mid z)$ is called the *posterior probability*, and $P(z \mid w)$ is called the *likelihood*.

Discussion
Bayes rule is used for two purposes. The first is *Bayesian update*. In this context, z represents some new information that has become available since an estimate $P(w)$

was formed of some hypothesis w. The application of Bayes' rule enables a new estimate of the probability of w (the posterior probability) to be calculated from estimates of the prior probability, the likelihood and $\mathrm{P}(z)$.

The second common application of Bayes' rule is for estimating posterior probabilities in probabilistic learning, where it is the core of ►Bayesian networks, ►naïve Bayes, and ►semi-naïve Bayesian techniques.

While Bayes' rule may initially appear mysterious, it is readily derived from the basic principle of conditional probability that

$$\mathrm{P}(w\,|\,z) = \mathrm{P}(w,z)\mathrm{P}(z) \tag{2}$$

As

$$\mathrm{P}(w,z) = \frac{\mathrm{P}(w)\mathrm{P}(w,z)}{\mathrm{P}(w)} \tag{3}$$

and

$$\frac{\mathrm{P}(w,z)}{\mathrm{P}(w)} = \mathrm{P}(z\,|\,w), \tag{4}$$

Bayes' rule (Eq. 1) follows by simple substitution of Eq. (4) into Eq. (3) and then of the result into Eq. (2).

Cross References

►Bayesian Methods
►Bayesian Network
►Naïve Bayes
►Semi-Naïve Bayesian Learning

Bayesian Methods

Wray Buntine
NICTA, Canberra, Australia

Definition

The two most important concepts used in Bayesian modeling are *probability* and *utility*. Probabilities are used to model our belief about the state of the world and utilities are used to model the *value* to us of different outcomes, thus to model costs and benefits. Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest from a space χ. The left and right arguments of the probability function are in general propositions (in the logical sense). Probabilities are updated based on new evidence or outcomes y using *Bayes rule*, which takes the form

$$p(x|C,y) = \frac{p(x|C)p(y|x,C)}{p(y|C)},$$

where χ is the discrete domain of x. More generally, any measurable set can be used for the domain χ. An integral or mixed sum and integral can replace the sum. For a utility function $u(x)$ of some event x, for instance the benefit of a particular outcome, the *expected value* of $u()$ is

$$\mathcal{E}_{x|C}[u(x)] = \sum_{x\in\mathcal{X}} p(x|C)u(x).$$

One then estimates the expected utility $\mathcal{E}_{x|C,y}[u(x)]$ based on different evidence, actions or outcomes y. An action is taken to maximize this expected utility, appealing to the principle of *maximum expected utility (MEU)*. A common application of this principle is recursive: one should take the action now that will maximize utility in the future, assuming all future actions are also taken to maximize utility.

Motivation and Background

In modeling a problem, primarily, one considers an interrelated space of *events* or *states, actions*, and *outcomes*. Events describe the state of the world, outcomes are also sometimes considered events but they are special in that one directly obtains from them costs or benefits. Actions allow one to influence the world. Some actions may instigate tests and thus also help measure the state of the world to reduce uncertainty. Some problems may be dynamic in that a sequence of actions and outcomes are considered and the resulting changes in states modeled.

The Bayesian approach is a modeling methodology that provides a principled approach of how to reason and act in the context of uncertainty and a dynamic environment. In the approach, probabilities are used to model all forms of belief or proportions about events and states, and then utilities are used to model the costs and benefits of any actions taken. An explicit assumption is that these probabilities and utilities can be adequately elicited and precisely modeled for the problem. An implicit assumption is that the computation required – recursive evaluation of

possibly nested integrals and sums (over domain variables) – can be done quickly enough so that the computation itself does not become a significant factor in the costs considered.

The Bayesian approach is named after Rev. Thomas Bayes, whose work was contributed to the Royal Society in 1763 after his death, although it was independently more generally presented as a theory by Laplace in 1812. The field was subsequently developed into a field of statistics, inference and decision theory by a stream of authors in the 1900s including Jeffreys (Bernardo and Smith, 1994). The field of statistics was dominated by the frequentist school during the 1990s, and for a time Bayesian methods were considered controversial. Like the different schools of theory in machine learning, these statistical approaches now coexist.

The Bayesian approach can be justified by axiomatic prescriptions of how a rational agent should reason and act, and by appeal to principles of consistency. In the context of learning, probabilities are used to infer models of the problem of interest, and then utilities are used to recommend predictions or analysis based on the models.

Theory

Basic Theory

First, consider definitions, the different kinds of probability, the process of reasoning (about probabilities), and making decisions.

Basic definitions: Probabilities are represented in the form of $p(x|C)$, where C is the current known context and x is some event(s) of interest. It is sufficient to place in C only terms relevant to x and ignore terms assumed by default. Moreover, both x and C must have well-defined events. For instance, x = "John is tall" is not considered a well-defined event since the word "tall" is not precise. One would instead replace it with something like x = "John is greater than 6 foot tall" or x = "Julie said John is tall."

An important functional used with probabilities is the *expected value*. For a function $f(x)$ of some event x from a space χ, the expected value of $f()$ is $\mathcal{E}_{x\in\chi}[f(x)]$.

Utility is used to measure value or relative satisfaction, and is usually represented as a function on outcomes. Costs are negative utility and benefits are positive. Utilities should be additive in worth, and are often practically interpreted in monetary units. Strictly speaking, the value of money is nonlinear (for most people, 2 billion dollars is not significantly better than 1 billion dollars), so it is not a correct utility measure. However, it is adequate when the range of financial transactions expected is reasonable.

Expected utility, which is the expected value of the utility function, is the fundamental quantity assessed with Bayesian methods. Some scenarios are the following:

Prediction: For prediction problems, the outcome is the "true" value, and the utility is sometimes the mean square error or the absolute error. In data mining, the choices are much richer, see ▶Model Evaluation.

Diagnosis: The outcome is the "true" diagnosis, and utility is made up of the differing costs of treatment, mistreatment, and delay or nontreatment, as well as any benefit from correct diagnosis.

Game playing: The utility comes from the eventual outcome of the game, each player has their own utility and the state of the game constantly changes as plays are made.

In Bayesian machine learning, we usually take utilities as a given, and the majority of the work revolves around evaluating and estimating probabilities and maximizing of expected utility. In some ranking tasks and generalized agent learning, the utilities themselves may be poorly understood.

Belief and proportions: Some probabilities correspond to *proportions* that exist in the real world, such as the proportion of school children in the general population of a given state. These real proportions can be measured by counting or sampling, and they are governed by Kolmogorov's Axioms for probability, including the probability of certainty is 1 and the probability of a disjunction of mutually exclusive events is the sum of the probabilities of the individual events. This kind of probability is used in the *Frequentist School* that only considers long term average proportions obtained from a series of independent and identical experiments. These proportions can be model parameters one wishes to reason about.

Probabilities can also represent beliefs. For instance, in 2000, one could have had a belief about the event that

George Bush would win the 2001 Presidential Election in the USA. This event is unique and has only one outcome, so the frequentist notion cannot be justified, i.e., there is no long-term sequence of different 2001 presidential elections with George Bush. Beliefs are usually considered to be *subjective*, in that they are specific to each agent, reflecting their sum of unique experiences, and the unique context in which the event in question occurs.

To better understand the role beliefs play in Bayesian methods, also see ▶Prior Probabilities.

Reasoning: A stylized version of probabilistic reasoning considers an event of interest one is reasoning about, x, and evidence, y, one may obtain. Typical scenarios are

Learning: $x = (\boldsymbol{\Theta}, M)$ are parameters $\boldsymbol{\Theta}$ of a model from family M, and $y = \boldsymbol{D}$ is a set of data $\boldsymbol{D} = \{d_1, \ldots, d_N\}$. So one considers $p(\boldsymbol{\Theta}, M|\boldsymbol{D}, C)$ versus $p(\boldsymbol{\Theta}, M|C)$.

Diagnosis: x a disease or condition, and y is a set of observable symptoms or diagnostic tests. One might choose a test y that maximizes the expected utility.

Hypothesis testing: x is a hypothesis H and y is some sequence of evidence $E_1, E_2, \ldots, E_n$, so we consider $p(H|E_1, E_2, \ldots, E_n)$ and hope it is sufficiently high.

Different probabilities are then considered:

$p(x|C)$: *The prior probability* for event x, called the base-rate in some contexts.

$p(y|C)$: The *prior probability* for evidence y. Once the evidence has been seen, this is also used as a proxy for the quality of the model.

$p(x|y, C)$: The *posterior probability* for event x given evidence y.

$p(y|x, C)$: The *likelihood* for the event x based on evidence y.

In the case of diagnostic reasoning, the prior $p(x|C)$ is usually the base rate for the disease or condition, and can be got from the population base rate.

In the case of learning, however, the prior $p(\boldsymbol{\Theta}, M|C)$ represents a prior distribution on parameters about which we may well be largely ignorant, or at least may not be able to readily elicit from experts. For instance, the proportion θ_D might be the probability of a new drug slowing the onset of AIDS related diseases. At the moment of initial testing, θ_D is unknown so one places a probability distribution over θ_D, which represents one's belief about the proportion.

These priors are second-order probabilities, beliefs about proportions, and they are the most challenging quantity modeled with the Bayesian approach. They can be a function on thousands of parameters, and can be critical in the success of applications. They are also challenging from the philosophical perspective.

Decision theory: The term *Bayesian inference* is usually reserved for the process of manipulating priors and posteriors, computing probabilities, and computing expected values. *Bayesian decision theory* describes the process of formulating utilities and then evaluating the (sometimes) recursive maximum expected utility formula, such as in game playing, or interactive advertising.

In Bayesian theory one takes the action that maximizes expected utility (MEU) in the current context, sometimes referred to as the *expected utility hypothesis*. Decision theory places this in a dynamic context and says each action should be taken to maximize expected future utility. This is defined recursively, so taken to the limit this implies the optimal future actions need to be determined before the optimal current action can be got via MEU.

Justifications

This section covers basic mathematical justifications of the theory. The best general reference for this is Bernardo and Smith (1994). Additional discussion of prior probabilities appears in ▶Prior Probabilities.

Note that Bayesian theory, with its acceptance as a branch of mainstream statistics, is widely accepted for the following reasons:

Application: It has extensive support through practical success, often times by clever combination of prior knowledge and statistical and computational finesse.

Explanation: It provides a convenient common language in which a variety of other theoretical approaches can be represented. For instance PAC, MDL methods, penalized likelihood methods, and the maximum margin approach all find good interpretations within the Bayesian framework.

Composition: It allows different reasoning tasks to be composed in a coherent way. With a probabilistic framework, the components can interoperate in a coherent manner, so that information may flow bidirectionally between components via probabilities.

Composition of processing steps in intelligent systems is a key application for Bayesian methods. For instance, natural language and vision recognition tasks can sometimes be broken down into a processing chain (for instance, doing a named entity recognition step before a dependency parsing step), but these components rarely work conclusively and unambiguously. By attaching probabilities to the output of components, and allowing probabilistic inputs, the uncertainty inherent in individual steps can be propagated and managed.

Theoretical justifications also exist to support each of the different components, probabilities, and utilities. These justifications are based on the concept of *normative* axioms, axioms that do not describe reasoning but rather prescribe basic principles it should follow. The axioms try to capture principles such as coherence and consistency in a quantitative manner. These various justifications have their reported shortcomings and a rich literature exists arguing about the details and postulating new variants. These axiomatic justifications are supportive of the Bayesian approach, but they are not irrefutable.

Justifying probabilities: In the Bayesian approach, beliefs and proportions are given the same mathematical treatment.

One set of arguably controversial justifications for this revolve around betting (Bernardo and Smith, 1994, Sect. 2.8.3). Someone's subjective beliefs about specific events, such as significant economic and political events (or horse races), are claimed to be measurable by offering them a series of options or bets. Moreover, if their beliefs do not behave like proportions, then a clever bookmaker can use a so-called Dutch book to consistently profit from them.

An alternative scheme for justifying probability by Cox is based on normative axioms that beliefs should follow. For instance, one controversial axiom by Cox is that belief about a single event should be represented by a single real number. These axioms are presented by Jaynes as rules for a robot (Jaynes, 2003), and as rules for intelligent systems by Horvitz et al. (1986).

Justifying decision theory: Another scheme again using normative axioms, by von Neumann and Morgenstern, is used to justify the use of utilities. This scheme assumes probabilities are the basis of inference about uncertainty. A different set of normative axiomatic schemes have been developed that justify the use of probabilities and utilities together under MEU, the best known is by Savage but others exist (Bernardo and Smith, 1994).

Bayesian Computation

The first part of this article has been devoted to a brief overview of the Bayesian approach. Computation for Bayesian inference is an extensive field itself. Here we review the basic aspects as a pointer to the literature. This is an active area of research in machine learning, statistics, and a many applied artificial intelligence communities such as natural language processing, image analysis, and others.

In general, in Bayesian reasoning one wants to estimate posterior average parameter values, or their average variance, or some other averaged quantity, then general formulas are given by (in the case of continuous parameters)

$$\overline{\boldsymbol{\Theta}} = \mathcal{E}_{\boldsymbol{\Theta}|\boldsymbol{D},M,C}[\boldsymbol{\Theta}] = \int_{\boldsymbol{\Theta}} \boldsymbol{\Theta} p(\boldsymbol{\Theta}|\boldsymbol{D},M,C)\mathrm{d}\boldsymbol{\Theta}$$
$$\mathrm{var}(\boldsymbol{\Theta}) = \mathcal{E}_{\boldsymbol{\Theta}|\boldsymbol{D},M,C}\left[(\boldsymbol{\Theta}-\overline{\boldsymbol{\Theta}})^2\right]$$

Marginal likelihood: A useful quantity to assist in evaluating results, and a worthy score in its own right is the marginal likelihood, in the continuous parameter case found from the likelihood $p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)$ by taking an average

$$p(\boldsymbol{D}|M,C) = \int_{\boldsymbol{\Theta}} p(\boldsymbol{\Theta}|M,C)p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)\mathrm{d}\boldsymbol{\Theta}.$$

This is also called the *normalizing constant* due to its occurrence in the posterior formula

$$p(\boldsymbol{\Theta}|\boldsymbol{D},M,C) = \frac{p(\boldsymbol{\Theta}|M,C)p(\boldsymbol{D}|\boldsymbol{\Theta},M,C)}{p(\boldsymbol{D}|M,C)}!.$$

It is generally difficult to estimate because of the multidimensional integrals and sums.

Exponential family distributions: Standard probability distributions covered in mathematical statistics, such as the ▶Gaussian Distribution, the Poisson, Dirichlet, Gamma, and Wishart, have very convenient mathematical properties that make Bayesian estimation easier. With these distributions, one computes statistics, called *sufficient statistics*, such as a mean and sum of squares (for the Gaussian), and then parameter estimation follows with a function inverse on a concave function. This is the basis of ▶linear regression, ▶principal components analysis, and some ▶decision tree learning methods, for instance. All good texts on mathematical statistics cover these in detail. Note the marginal likelihood is often computable in closed form for exponential family distributions.

Graphical models: ▶Graphical Models are a general family of of probabilistic models formed by composing graphs over variables. They work particularly well with exponential family distributions, and allow a rich variety of popular machine learning and data mining methods to be represented and manipulated. Graphical models allow complex models to be composed from simpler components and provide a family of algorithm schemes for developing inference and learning methods that operate on them. They have become the *de facto* standard for presenting (suitable decomposed) models and algorithms in the machine learning community.

Maximum a posterior estimation: known as MAP, is usually the simplest form of parameter estimation that could be called Bayesian. It also corresponds to a penalized or regularized maximum likelihood method. Given the posterior for a stylized learning problem of the previous section, one finds the parameters $\boldsymbol{\Theta}$ that maximizes the posterior $p(\boldsymbol{\Theta}, M|\boldsymbol{D}, C)$, which can be conveniently done without computing the marginal likelihood above, so

$$\widehat{\boldsymbol{\Theta}_{MP}} = \underset{\boldsymbol{\Theta}}{\operatorname{argmax}} \log p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C),$$

where the log probability can be broken down as a prior and a likelihood term

$$\log p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C) = \log p(\boldsymbol{\Theta}|M, C) + \log p(\boldsymbol{D}|\boldsymbol{\Theta}, M, C).$$

The Laplace approximation: When the posterior is well behaved, and there is a large amount of data, the posterior is focused around a vanishing small region in parameter space of diameter $O(1/\sqrt{(}N))$. If this occurs away from the boundary of the parameter space, then one can make a second-order Taylor expansion of the log. posterior at the MAP point and the result is a Gaussian approximation to the posterior.

$$\log p(\boldsymbol{D}, \boldsymbol{\Theta}|M, C) \approx \log p(\boldsymbol{D}, \widehat{\boldsymbol{\Theta}_{MP}}|M, C) + \frac{1}{2}\left(\widehat{\boldsymbol{\Theta}_{MP}} - \boldsymbol{\Theta}\right)^T \left.\frac{\mathrm{d}^2 \log p(\boldsymbol{D}, \boldsymbol{\Theta}|M, C)}{\mathrm{d}\boldsymbol{\Theta}\mathrm{d}\boldsymbol{\Theta}^T}\right|_{\boldsymbol{\Theta}=\widehat{\boldsymbol{\Theta}_{MP}}} \left(\widehat{\boldsymbol{\Theta}_{M,P}} - \boldsymbol{\Theta}\right).$$

From this, one can approximate integrals such as the marginal likelihood $p(\boldsymbol{D}|M, C)$. This is known as the *Laplace approximation*, the name of the corresponding general method used for the asymptotic expansion of integrals. In general, this is a poor approximation, but it serves to aid our understanding of parameter estimation (MacKay, 2003 Chaps. 27 and 28), and is the approximate basis for some model selection criteria.

Latent variable models: Latent variables are data that are hidden and thus never observed in the evidence. However, their existence is postulated as a significant component of the model. For instance, in ▶Clustering (an unsupervised method) and finite mixture models generally, one assumes each data point has a hidden class label, thus the Bayesian model of clustering is a simple kind of latent variable model.

▶*Markov chain Monte Carlo* methods: The most general form of reasoning and estimation available are the *Markov chain Monte Carlo* (MCMC) methods. The MCMC methods couple two processes: first, they use *Monte Carlo* or simulation methods to estimate the integral, and second they use a *Markov Chain* to sample, so sampling is sequentially (Markovian) based, and samples are not independent.

Simulation methods generally use the functional form of $p(\boldsymbol{\Theta}, \boldsymbol{D}|M, C)$ so we do not need to compute the marginal likelihood. Hence, given a set of I samples $\{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_I\}$ the expected value is approximated with a weighted average

$$\overline{\boldsymbol{\Theta}} \approx \frac{1}{I}\sum_{i=1}^{I} w_i \boldsymbol{\Theta}_i.$$

The simplest case is where the samples are made independently according to the posterior itself and then the

weights $w_i = 1$, This is called the ordinary Monte Carlo (OMC) method, but it is not often usable in practice because efficient multidimensional posterior samplers rarely exist. Alternatively, one can sample according to a Markov Chain, $\boldsymbol{\Theta}_{i+1} \sim q(\boldsymbol{\Theta}_{i+1}|\boldsymbol{\Theta}_i)$, so each $\boldsymbol{\Theta}_{i+1}$ is conditionally dependent on $\boldsymbol{\Theta}_i$. So while samples are not independent, as long as the long run distribution of the Markov chain is the same as the posterior, the same approximation formula holds. There are a rich variety of MCMC methods, and this forms one of the key areas of current research.

Gibbs sampling: The simplest kind of MCMC method samples each dimension (or sub-vector) in turn. Suppose the parameter vector has K real components, $\boldsymbol{\Theta} = (\theta_1, \ldots, \theta_K)$. Sampling a complete $\boldsymbol{\Theta}$ in one go is not generally possible given just a functional form of the posterior $p(\boldsymbol{\Theta}|\boldsymbol{D}, M, C)$ but given no computable form for the normalizing constant. Gibbs sampling works in the one-dimensional case where normalizing bounds can be obtained and sampling tricks used. The conditional posterior of θ_k is given by

$$p(\theta_k|(\theta_1, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta_K), \boldsymbol{D}, M, C),$$

and this is usually easier to sample from.

The Gibbs (and MCMC) sample $\boldsymbol{\Theta}_{i+1}$ can be drawn given the previous sample $\boldsymbol{\Theta}_i$ by progressively resampling each dimension in turn and so slowly updating the full vector:

1. Sample $\theta_{i+1,1}$ according to $p(\theta_1|\theta_{i,2}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$.
...
k. Sample $\theta_{i+1,k}$ according to $p(\theta_2|\theta_{i+1,1}, \ldots, \theta_{i+1,k-1}, \theta_{i,k+1}, \ldots, \theta_{i,K}, \mathbf{D}, M, C)$.
...
K. Sample $\theta_{i+1,k}$ according to $p(\theta_K|\theta_{i+1,1}, \ldots, \theta_{i+1,K-1}, \mathbf{D}, M, C)$.

In samping terms, this method is no more successful than coordinate-wise ascent is as a primitive greedy search method: it is supported by theoretical results but can be very slow to converge.

Variational approximations: When the function you seek to optimize or average over presents difficulty, perhaps it is highly multimodal, then one option is to change the function itself, and replace it with a more readily approximated function. Variational methods provide a general principle for doing this safely. The general principle uses variational calculus, which is the calculus over functions, not just variables. Variational methods are a very general approach that can be used to develop a broad range of algorithms (Wainwright and Jordan, 2008).

Nonparametric models: The above discussion implicitly assumed the model has a fixed finite parameter vector $\boldsymbol{\Theta}$. If one is attempting to model a regression function, or a language grammar, or image model of unknown *a priori* structural complexity, then one cannot know the dimension ahead of time. Moreover, as in the case of functions, the dimension cannot always be finite. The ▶Bayesian Nonparametric Models address this situation, and are perhaps the most important family of techniques for general machine learning.

Cross References

▶Bayes Rule
▶Bayesian Nonparametric Models
▶Markov Chain Monte Carlo
▶Prior Probability

Recommended Reading

A good introduction to the problems of uncertainty and philosophical issues behind the Bayesian treatment of probability is in Lindley (2006). From the statistical machine learning perspective, a good introductory text is by MacKay (2003) who carefully covers information theory, probability, and inference but not so much statistical machine learning. Another alternative introduction to probabilities is the posthumously completed and published work of Jaynes (2003).

Discussions from the frequentist versus Bayesian battlefront can be found in works such as (Rosenkrantz and Jaynes, 1983), and in the approximate artificial intelligence versus probabilistic battlefront in discussion articles such as Cheeseman's (1988) and the many responses and rebuttals. It should be noted that it is the continued success in applications that have really led these methods into the mainstream, not the entertaining polemics.

Good mathematical statistics text books, such as Casella and Berger (2001) cover the breadth of statistical methods and therefore handle basic Bayesian theory. A more comprehensive treatment is given in Bayesian texts such as Gelman et al. (2003).

Most advanced statistical machine learning text books cover Bayesian methods, but to fully understand the subtleties of prior beliefs and Bayesian methodology one needs to view more advanced Bayesian literature. A detailed theoretical reference for Bayesian methods is Bernardo and Smith (1994).

Bernardo, J., & Smith, A. (1994). *Bayesian theory*. Chichester: Wiley.
Casella, G., & Berger, R. (2001). *Statistical inference* (2nd ed.). Pacific Grove: Duxbury.

Cheeseman, P. (1988). An inquiry into computer understanding. *Computational Intelligence, 4*(1), 58–66.
Gelman, A., Carlin, J., Stern, H., & Rubin, D. (2003). *Bayesian data analysis* (2nd ed.). Boca Raton: Chapman & Hall/CRC Press.
Horvitz, E., Heckerman, D., & Langlotz, C. (1986). A framework for comparing alternative formalisms for plausible reasoning. *Fifth National Conference on Artificial Intelligence*, Philadelphia, pp. 210–214.
Jaynes, E. (2003). *Probability theory: the logic of science*. New York: Cambridge University Press.
Lindley, D. (2006). *Understanding uncertainty*. Hoboken: Wiley.
MacKay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge: Cambridge University Press.
Rosenkrantz, R. (Ed.). (1983). *E.T. Jaynes: papers on probability, statistics and statistical physics*. Dordrecht: D. Reidel.
Wainwright, M. J., & Jordan, M. I. (2008). *Graphical models, exponential families, and variational inference*. Hanover: Now Publishers.

Bayesian Model Averaging

►Learning Graphical Models

Bayesian Network

Synonyms

Bayes net

Definition

A Bayesian network is a form of directed ►graphical model for representing multivariate probability distributions.

The nodes of the network represent a set of random variables, and the directed arcs represent causal relationships between variables. The *Markov property* is usually required: every direct dependency between a possible cause and a possible effect has to be shown with an arc. Bayesian networks with the Markov property are called *I-maps* (independence maps). If all arcs in the network correspond to a direct dependence on the system being modeled, then the network is said to be a *D-map* (dependence-map). Each node is associated with a conditional probability distribution, that quantifies the effects the parents of the node, if any, have on it. Bayesian support various forms of reasoning: *diagnosis*, to derive causes from symptoms, *prediction*, to derive effects from causes, and *intercausal reasoning*, to discover the mutual causes of a common effect.

Cross References

►Graphical Models

Bayesian Nonparametric Models

Peter Orbanz[1], Yee Whye Teh[2]
[1]Cambridge University, Cambridge, UK
[2]University College London, London, UK

Synonyms

Bayesian methods; Dirichlet process; Gaussian processes; Prior probabilities

Definition

A Bayesian nonparametric model is a Bayesian model on an infinite-dimensional parameter space. The parameter space is typically chosen as the set of all possible solutions for a given learning problem. For example, in a regression problem, the parameter space can be the set of continuous functions, and in a density estimation problem, the space can consist of all densities. A Bayesian nonparametric model uses only a finite subset of the available parameter dimensions to explain a finite sample of observations, with the set of dimensions chosen depending on the sample such that the effective complexity of the model (as measured by the number of dimensions used) adapts to the data. Classical adaptive problems, such as nonparametric estimation and model selection, can thus be formulated as Bayesian inference problems. Popular examples of Bayesian nonparametric models include Gaussian process regression, in which the correlation structure is refined with growing sample size, and Dirichlet process mixture models for clustering, which adapt the number of clusters to the complexity of the data. Bayesian nonparametric models have recently been applied to a variety of machine learning problems, including regression, classification, clustering, latent variable modeling, sequential modeling, image segmentation, source separation, and grammar induction.

Motivation and Background

Most of machine learning is concerned with learning an appropriate set of parameters within a model class from ►training data. The meta-level problems of determining appropriate model classes are referred to as model selection or model adaptation. These constitute important concerns for machine learning practitioners, not only for avoidance of over-fitting and under-fitting, but also for discovery of the causes and structures underlying data. Examples of model selection and adaptation include selecting the number of clusters in a clustering problem, the number of hidden states in a hidden Markov model, the number of latent variables in a latent variable model, or the complexity of features used in nonlinear regression.

Nonparametric models constitute an approach to model selection and adaptation where the sizes of models are allowed to grow with data size. This is as opposed to *parametric models*, which use a fixed number of parameters. For example, a parametric approach to density estimation would be to fit a Gaussian or a mixture of a fixed number of Gaussians by maximum likelihood. A nonparametric approach would be a Parzen window estimator, which centers a Gaussian at each observation (and hence uses one mean parameter per observation). Another example is the support vector machine with a Gaussian kernel. The representer theorem shows that the decision function is a linear combination of Gaussian radial basis functions centered at every input vector, and thus has a complexity that grows with more observations. Nonparametric methods have long been popular in classical (non-Bayesian) statistics (Wasserman, 2006). They often perform impressively in applications and, though theoretical results for such models are typically harder to prove than for parametric models, appealing theoretical properties have been established for a wide range of models.

Bayesian nonparametric methods provide a Bayesian framework for model selection and adaptation using nonparametric models. A Bayesian formulation of nonparametric problems is nontrivial, since a Bayesian model defines prior and posterior distributions on a single fixed parameter space, but the dimension of the parameter space in a nonparametric approach should change with sample size. The Bayesian nonparametric solution to this problem is to use an infinite-dimensional parameter space, and to invoke only a finite subset of the available parameters on any given finite data set. This subset generally grows with the data set. In the context of Bayesian nonparametric models, "infinite-dimensional" can therefore be interpreted as "of finite but unbounded dimension." More precisely, a Bayesian nonparametric model is a model that (1) constitutes a Bayesian model on an infinite-dimensional parameter space and (2) can be evaluated on a finite sample in a manner that uses only a finite subset of the available parameters to explain the sample.

We make the above description more concrete in the next section when we describe a number of standard machine learning problems and the corresponding Bayesian nonparametric solutions. As we will see, the parameter space in (1) typically consists of functions or of measures, while (2) is usually achieved by marginalizing out surplus dimensions over the prior. Random functions and measures and, more generally, probability distributions on infinite-dimensional random objects are called *stochastic processes*; examples that we will encounter include Gaussian processes, Dirichlet processes, and beta processes. Bayesian nonparametric models are often named after the stochastic processes they contain. The examples are then followed by theoretical considerations, including formal constructions and representations of the stochastic processes used in Bayesian nonparametric models, exchangeability, and issues of consistency and convergence rate. We conclude this chapter with future directions and a list of literature available for reading.

Examples

Clustering with mixture models. Bayesian nonparametric generalizations of finite mixture models provide an approach for estimating both the number of components in a mixture model and the parameters of the individual mixture components simultaneously from data. Finite mixture models define a density function over data items x of the form $p(x) = \sum_{k=1}^{K} \pi_k p(x|\theta_k)$, where π_k is the mixing proportion and θ_k are parameters associated with component k. The density can be written in a non-standard manner as an integral: $p(x) = \int p(x|\theta)G(\theta)d\theta$, where $G = \sum_{k=1}^{K} \pi_k \delta_{\theta_k}$ is a discrete mixing distribution encapsulating all the parameters of the mixture model and δ_θ is a dirac distribution (atom) centered at θ. Bayesian nonparametric mixtures use

mixing distributions consisting of a *countably infinite* number of atoms instead:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}. \quad (1)$$

This gives rise to mixture models with an infinite number of components. When applied to a finite training set, only a finite (but varying) number of components will be used to model the data, since each data item is associated with exactly one component but each component can be associated with multiple data items. Inference in the model then automatically recovers both the number of components to use and the parameters of the components. Being Bayesian, we need a prior over the mixing distribution G, and the most common prior to use is a *Dirichlet process* (DP). The resulting mixture model is called a DP mixture.

Formally, a Dirichlet process $DP(\alpha, H)$ parametrized by a concentration paramter $\alpha > 0$ and a base distribution H is a prior over distributions (probability measures) G such that, for any finite partition $A_1, \ldots, A_m$ of the parameter space, the induced random vector $(G(A_1), \ldots, G(A_m))$ is Dirichlet distributed with parameters $(\alpha H(A_1), \ldots, \alpha H(A_m))$ (see entitled Section "Theory" for a discussion of subtleties involved in this definition). It can be shown that draws from a DP will be discrete distributions as given in (1). The DP also induces a distribution over partitions of integers called the *Chinese restaurant process* (CRP), which directly describes the prior over how data items are clustered under the DP mixture. For more details on the DP and the CRP, see ►Dirichlet Process.

Nonlinear regression. The aim of regression is to infer a continuous function from a training set consisting of input–output pairs $\{(t_i, x_i)\}_{i=1}^n$. Parametric approaches parametrize the function using a finite number of parameters and attempt to infer these parameters from data. The prototypical Bayesian nonparametric approach to this problem is to define a prior distribution over continuous functions directly by means of a *Gaussian process* (GP). As explained in the Chapter ►Gaussian Process, a GP is a distribution on an infinite collection of random variables X_t, such that the joint distribution of each finite subset $X_{t_1}, \ldots, X_{t_m}$ is a multivariate Gaussian. A value x_t taken by the variable X_t can be regarded as the value of a continuous function f at t, that is, $f(t) = x_t$. Given the training set, the Gaussian process posterior is again a distribution on functions, conditional on these functions taking values $f(t_1) = x_1, \ldots, f(t_n) = x_n$.

Latent feature models. These models represent a set of objects in terms of a set of latent features, each of which represents an independent degree of variation exhibited by the data. Such a representation of data is sometimes referred to as a distributed representation. In analogy to nonparametric mixture models with an unknown number of clusters, a Bayesian nonparametric approach to latent feature modeling allows for an unknown number of latent features. The stochastic processes involved here are known as the *Indian buffet process* (IBP) and the *beta process* (BP). Draws from BPs are random discrete measures, where each of an infinite number of atoms has a mass in $(0, 1)$ but the masses of atoms need not sum to 1. Each atom corresponds to a feature, with the mass corresponding to the probability that the feature is present for an object. We can visualize the occurrences of features among objects using a binary matrix, where the (i, k) entry is 1 if object i has feature k and 0 otherwise. The distribution over binary matrices induced by the BP is called the IBP.

►*Hidden Markov models* (*HMMs*). HMMs are popular models for sequential or temporal data, where each time step is associated with a state, with state transitions dependent on the previous state. An *infinite HMM* is a Bayesian nonparametric approach to HMMs, where the number of states is unbounded and allowed to grow with the sequence length. It is defined using one DP prior for the transition probabilities going out from each state. To ensure that the set of states reachable from each outgoing state is the same, the base distributions of the DPs are shared and given a DP prior recursively. The construction is called a *hierarchical Dirichlet process* (HDP); see below.

►*Density estimation*. A nonparametric Bayesian approach to density estimation requires a prior on densities or distributions. However, the DP is not useful in this context, since it generates discrete distributions. A useful density estimator should smooth the empirical density (such as a Parzen window estimator), which requires a prior that can generate smooth distributions. Priors applicable in density estimation problems include DP mixture models and Pólya trees.

If $p(x|\theta)$ is a smooth density function, the density $\sum_{k=1}^{\infty} \pi_k p(x|\theta_k)$ induced by a DP mixture model is a

smooth random density, such that DP mixtures can be used as prior in density estimation problems.

Pólya trees are priors on probability distributions that can generate both discrete and piecewise continuous distributions, depending on the choice of parameters. Pólya trees are defined by a recursive infinitely deep binary subdivision of the domain of the generated random measure. Each subdivision is associated with a beta random variable which describes the relative amount of mass on each side of the subdivision. The DP is a special case of a Pólya tree corresponding to a particular parametrization. For other parametrizations the resulting random distribution can be smooth, so it is suitable for density estimation.

Power-law Phenomena. Many naturally occurring phenomena exhibit power-law behavior. Examples include natural languages, images, and social and genetic networks. An interesting generalization of the DP, called the *Pitman-Yor process*, $\mathrm{PYP}(\alpha, d, H)$, has recently been successfully used to model power-law data. The Pitman-Yor process augments the DP by a third parameter $d \in [0,1)$. When $d = 0$ the PYP is a $DP(\alpha, H)$, while when $\alpha = 0$ it is a so called *normalized stable process*.

Sequential modeling. HMMs model sequential data using latent variables representing the underlying state of the system, and assuming that each state only depends on the previous state (the so called Markov property). In some applications, for example language modeling and text compression, we are interested in directly modeling sequences without using latent variables, and without making any Markov assumptions, i.e., modeling each observation conditional on all previous observations in the sequence. Since the set of potential sequences of previous observations is unbounded, this calls for nonparametric models. A *hierarchical Pitman-Yor process* can be used to construct a Bayesian nonparametric solution whereby the conditional probabilities are coupled hierarchically.

Dependent and hierarchical models. Most of the Bayesian nonparametric models described so far are applied in settings where observations are homogeneous or exchangeable. In many real world settings observations are not homogeneous, and in fact are often structured in interesting ways. For example, the data generating process might change over time thus observations at different times are not exchangeable, or observations might come in distinct groups with those in the same group being more similar than across groups.

Significant recent efforts in Bayesian nonparametrics research have been placed in developing extensions that can handle these non-homogeneous settings. Dependent Dirichlet processes are stochastic processes, typically over a spatial or temporal domain, which define a Dirichlet process (or a related random measure) at each point with neighboring DPs being more dependent. These are used for spatial modeling, nonparametric regression, as well as for modeling temporal changes. Alternatively, hierarchical Bayesian nonparametric models like the hierarchical DP aim to couple multiple Bayesian nonparametric models within a hierarchical Bayesian framework. The idea is to allow sharing of statistical strength across multiple groups of observations. Among other applications, these have been used in the infinite HMM, topic modeling, language modeling, word segmentation, image segmentation, and grammar induction. For an overview of various dependent Bayesian nonparametric models and their applications in biostatistics please refer to Dunson (2010). Teh and Jordan (2010) is an overview of hierarchical Bayesian nonparametric models as well as a variety of applications in machine learning.

Theory

As we saw in the preceding examples, Bayesian nonparametric models often make use of priors over functions and measures. Because these spaces typically have uncountable number of dimensions, extra care has to be taken to define the priors properly and to study the asymptotic properties of estimation in the resulting models. In this section we give an overview of the basic concepts involved in the theory of Bayesian nonparametric models. We start with a discussion of the importance of exchangeability in Bayesian parametric and nonparametric statistics. This is followed by representations of the priors and issues of convergence.

Exchangeability

The underlying assumption of all Bayesian methods is that the parameter specifying the observation model is a random variable. This assumption is subject to

much criticism, and at the heart of the Bayesian versus non-Bayesian debate that has long divided the statistics community. However, there is a very general type of observation for which the existence of such a random variable can be derived mathematically: For so-called *exchangeable* observations, the Bayesian assumption that a randomly distributed parameter exists is not a modeling assumption, but a mathematical consequence of the data's properties.

Formally, a sequence of variables $X_1, X_2, \ldots, X_n$ over the same probability space $(\mathcal{X}, \Omega)$ is *exchangeable* if their joint distribution is invariant to permuting the variables. That is, if P is the joint distribution and σ any permutation of $\{1, \ldots, n\}$, then

$$P(X_1 = x_1, X_2 = x_2 \ldots X_n = x_n) = P(X_1 = x_{\sigma(1)}, X_2 = x_{\sigma(2)} \ldots X_n = x_{\sigma(n)}). \quad (2)$$

An infinite sequence $X_1, X_2, \ldots$ is *infinitely exchangeable* if $X_1, \ldots, X_n$ is exchangeable for *every* $n \geq 1$. In this chapter, we mean infinite exchangeability whenever we write exchangeability. Exchangeability reflects the assumption that the variables do not depend on their indices although they may be dependent among themselves. This is typically a reasonable assumption in machine learning and statistical applications, even if the variables are not themselves independently and identically distributed (iid).

Exchangeability is a much weaker assumption than iid since iid variables are automatically exchangeable.

If θ parametrizes the underlying distribution, and one assumes a prior distribution over θ, then the resulting marginal distribution over $X_1, X_2, \ldots$ with θ marginalized out will still be exchangeable. A fundamental result credited to de Finetti (1931) states that the converse is also true. That is, if $X_1, X_2, \ldots$ is (infinitely) exchangeable, then there is a random θ such that:

$$P(X_1, \ldots, X_n) = \int P(\theta) \prod_{i=1}^{n} P(X_i|\theta) d\theta \quad (3)$$

for every $n \geq 1$. In other words, the seemingly innocuous assumption of exchangeability automatically implies the existence of a hierarchical Bayesian model with θ being the random latent parameter. This the crux of the fundamental importance of exchangeability to Bayesian statistics.

In de Finetti's Theorem it is important to stress that θ can be infinite dimensional (it is typically a random measure), thus the hierarchical Bayesian model (3) is typically a nonparametric one. For an example, the Blackwell–MacQueen urn scheme (related to the CRP) is exchangeable and thus implicitly defines a random measure, namely the DP (see ▶Dirichlet Process for more details). In this sense, we will see below that de Finetti's theorem is an alternative route to Kolmogorov's extension theorem, which implicitly defines the stochastic processes underlying Bayesian nonparametric models.

Model Representations

In finite dimensions, a probability model is usually defined by a density function or probability mass function. In infinite dimensional spaces, this approach is not generally feasible, for reasons explained below. To define or work with a Bayesian nonparametric model, we have to choose alternative mathematical representations.

Weak distributions. A weak distribution is a representation for the distribution of a stochastic process, that is, for a probability distribution on an infinite-dimensional sample space. If we assume that the dimensions of the space are indexed by $t \in T$, the stochastic process can be regarded as the joint distribution P of an infinite set of random variables $\{X_t\}_{t \in T}$. For any finite subset $S \subset T$ of dimensions, the joint distribution P_S of the corresponding subset $\{X_t\}_{t \in S}$ of random variables is a finite-dimensional marginal of P. The *weak distribution* of a stochastic process is the set of all its finite-dimensional marginals, that is, the set $\{P_S : S \subset T, |S| < \infty\}$. For example, the customary definition of the Gaussian process as an infinite collection of random variables, each finite subset of which has a joint Gaussian distribution, is an example of a weak distribution representation. In contrast to the explicit representations to be described below, this representation is generally not generative, because it represents the distribution rather than a random draw, but is more widely applicable.

Apparently, just defining a weak distribution in this manner need not imply that it is a valid representation of a stochastic process. A given collection of finite-dimensional distributions represents a stochastic

process only (1) if a process with these distributions as its marginals actually exists, and (2) if it is uniquely defined by the marginals. The mathematical result which guarantees that weak distribution representations are valid is the *Kolmogorov extension theorem* (also known as the Daniell–Kolmogorov theorem or the Kolmogorov consistency theorem). Suppose that a collection $\{P_S : S \subset T, |S| < \infty\}$ of distributions is given. If all distributions in the collection are marginals of each other, that is, if P_{S_1} is a marginal of P_{S_2} whenever $S_1 \subset S_2$, the set of distributions is called a *projective family*. The Kolmogorov extension theorem states that, if the set T is countable, and if the distributions P_S form a projective family, then there exists a uniquely defined stochastic process with the collection $\{P_S\}$ as its marginal distributions. In other words, any projective family for a countable set T of dimensions is the weak distribution of a stochastic process. Conversely, any stochastic process can be represented in this manner, by computing its set of finite-dimensional marginals.

The weak distribution representation assumes that all individual random variable X_t of the stochastic process take values in the same sample space Ω. The stochastic process P defined by the weak distribution is then a probability distribution on the sample space Ω^T, which can be interpreted as the set of all functions $f : T \to \Omega$. For example, to construct a GP we might choose $T = \mathbb{Q}$ and $\Omega = \mathbb{R}$ to obtain real-valued functions on the countable space of rational numbers. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, the function f can then be extended to all of $\mathbb{R}$ by continuity. To define the DP as a distribution over probability measures on $\mathbb{R}$, we note that a probability measure is a set function that maps "random events," i.e., elements of the Borel σ-algebra $\mathcal{B}(\mathbb{R})$ of $\mathbb{R}$, into probabilities in $[0,1]$. We could therefore choose a weak distribution consisting of Dirichlet distributions, and set $T = \mathcal{B}(\mathbb{R})$ and $\Omega = [0,1]$. However, this approach raises a new problem because the set $\mathcal{B}(\mathbb{R})$ is not countable. As in the GP, we can first define the DP on a countable "base" for $\mathcal{B}(\mathbb{R})$ then extend to all random events by continuity of measures. More precise descriptions are unfortunately beyond the scope of this chapter.

Explicit representations. Explicit representations directly describe a random draw from a stochastic process, rather than its distribution. A prominent example of an explicit representation is the so-called *stick-breaking representation* of the Dirichlet process. The discrete random measure G in (1) is completely determined by the two infinite sequences $\{\pi_k\}_{k\in\mathbb{N}}$ and $\{\theta_k\}_{k\in\mathbb{N}}$. The stick-breaking representation of the DP generates these two sequences by drawing $\theta_k \sim H$ iid and $v_k \sim Beta(1,\alpha)$ for $k = 1, 2, \ldots$. The coefficients π_k are then computed as $\pi_k = v_k \prod_{j=1}^{k-1}(1 - v_k)$. The measure G so obtained can be shown to be distributed according to a $DP(\alpha, G_0)$. Similar representations can be derived for the Pitman–Yor process and the beta process as well. Explicit representations, if they exist for a given model, are typically of great practical importance for the derivation of algorithms.

Implicit Representations. A third representation of infinite dimensional models is based on de Finetti's Theorem. Any exchangeable sequence $X_1, \ldots, X_n$ uniquely defines a stochastic process θ, called the de Finetti measure, making the X_i's iid. If the X_i's are sufficient to define the rest of the model and their conditional distributions are easily specified, then it is sufficient to work directly with the X_i's and have the underlying stochastic process implicitly defined. Examples include the Chinese restaurant process (an exchangeable distribution over partitions) with the DP as the de Finetti measure, and the Indian buffet process (an exchangeable distribution over binary matrices) with the BP being the corresponding de Finetti measure. These implicit representations are useful in practice as they can lead to simple and efficient inference algorithms.

Finite representations. A fourth representation of Bayesian nonparametric models is as the infinite limit of finite (parametric) Bayesian models. For example, DP mixtures can be derived as the infinite limit of finite mixture models with particular Dirichlet priors on mixing proportions, GPs can be derived as the infinite limit of particular Bayesian regression models with Gaussian priors, while BPs can be derived as from the limit of an infinite number of independent beta variables. These representations are sometimes more intuitive for practitioners familiar with parametric models. However, not all Bayesian nonparametric models can be expressed in this fashion, and they do not necessarily make clear the mathematical subtleties involved.

Consistency and Convergence Rates

A recent series of works in mathematical statistics examines the convergence properties of Bayesian

nonparametric models, and in particular the questions of *consistency* and *convergence rates*. In this context, a Bayesian model is called consistent if, given that an infinite amount of data is available, the model posterior will concentrate in a neighborhood of the true solution (e.g., true function or density). A rate of convergence specifies, for a finite sample, how rapidly the posterior concentrates depending on the sample size. In their pioneering article Diaconis and Freedman (1986) showed, to the great surprise of much of the Bayesian community, that models such as the Dirichlet process can be inconsistent, and may converge to arbitrary solutions even for an infinite amount of data.

More recent results, notably by van der Vaart and Ghosal, apply modern methods of mathematical statistics to study the convergence properties of Bayesian nonparametric models (see e.g., Gho, (2010) and references therein). Consistency has been shown for a number of models, including Gaussian processes and Dirichlet process mixtures. However, a particularly interesting aspect of this line of work are results on convergence rates, which specify the rate of concentration of the posterior depending on sample size, on the complexity of the model, and on how much probability mass the prior places around the true solution. To make such results quantitative requires a measure for the complexity of a Bayesian nonparametric model. This is done by means of complexity measures developed in empirical process theory and statistical learning theory, such as metric entropies, covering numbers and bracketing, some of which are well-known in theoretical machine learning.

Inference

There are two aspects to inference from Bayesian nonparametric models: the analytic tractability of posteriors for the stochastic processes embedded in Bayesian nonparametric models, and practical inference algorithms for the overall models. Bayesian nonparametric models typically include stochastic processes such as the Gaussian process and the Dirichlet process. These processes have an infinite number of dimensions, hence naïve algorithmic approaches to computing posteriors are generally infeasible. Fortunately, these processes typically have analytically tractable posteriors, so all but finitely many of the dimensions can be analytically integrated out efficiently. The remaining dimensions, along with the parametric parts of the models, can then be handled by the usual inference techniques employed in parametric Bayesian modeling, including Markov chain Monte Carlo, sequential Monte Carlo, variational inference, and message-passing algorithms like expectation propagation. The precise choice of approximations to use will depend on the specific models under consideration, with speed/accuracy trade-offs between different techniques generally following those for parametric models. In the following, we will give two examples to illustrate the above points, and discuss a few theoretical issues associated with the analytic tractability of stochastic processes.

Examples

In Gaussian process regression, we model the relationship between an input x and an output y using a function f, so that $y \sim f(x) + \epsilon$, where ϵ is iid Gaussian noise. Given a GP prior over f and a finite training data set $\{(x_i, y_i)\}_{i=1}^n$ we wish to compute the posterior over f. Here we can use the weak representation of f and note that $\{f(x_i)\}_{i=1}^n$ is simply a finite-dimensional Gaussian with mean and covariance given by the mean and covariance functions of the GP. Inference for $\{f(x_i)\}_{i=1}^n$ is then straightforward. The approach can be thought of equivalently as marginalizing out the whole function except its values on the training inputs. Note that although we only have the posterior over $\{f(x_i)\}_{i=1}^n$, this is sufficient to reconstruct the function evaluated at any other point x_0 (say the test input), since $f(x_0)$ is Gaussian and independent of the training data $\{(x_i, y_i)\}_{i=1}^n$ given $\{f(x_i)\}_{i=1}^n$. In GP regression the posterior over $\{f(x_i)\}_{i=1}^n$ can be computed exactly. In GP classification or other regression settings with nonlinear likelihood functions, the typical approach is to use sparse methods based on variational approximations or expectation propagation; see Chapter ▶Gaussian Process for details.

Our second example involves Dirichlet process mixture models. Recall that the DP induces a clustering structure on the data items. If our training set consists of n data items, since each item can only belong to one cluster, there are at most n clusters represented in the training set. Even though the DP mixture itself has an infinite number of potential clusters, all but finitely

many of these are not associated with data, thus the associated variables need not be explicitly represented at all. This can be understood either as marginalizing out these variables, or as an implicit representation which can be made explicit whenever required by sampling from the prior. This idea is applicable for DP mixtures using both the Chinese restaurant process and the stick-breaking representations. In the CRP representation, each data item x_i is associated with a cluster index z_i, and each cluster k with a parameter θ_k^* (these parameters can be marginalized out if H is conjugate to F), and these are the only latent variables that need be represented in memory. In the stick-breaking representation, clusters are ordered by decreasing prior expected size, with cluster k associated with a parameter θ_k^* and a size π_k. Each data item is again associated with a cluster index z_i, and only the clusters up to $K = \max(z_1, \ldots, z_n)$ need to be represented. All clusters with index $> K$ need not be represented since their posterior conditioning on $\{(x_i, z_i)\}_{i=1}^n$ is just the prior.

On Bayes Equations and Conjugacy

It is worth noting that the posterior of a Bayesian model is, in abstract terms, defined as the conditional distribution of the parameter given the data and the hyperparameters, and this definition does not require the existence of a Bayes equation. If a Bayes equation exists for the model, the posterior can equivalently be defined as the left-hand side of the Bayes equation. However, for some stochastic processes, notably the DP on an uncountable space such as $\mathbb{R}$, it is not possible to define a Bayes equation even though the posterior is still a well-defined mathematical object. Technically speaking, existence of a Bayes equation requires the family of all possible posteriors to be dominated by the prior, but this is not the case for the DP. That posteriors of these stochastic processes can be evaluated at all is solely due to the fact that they admit an analytic representation.

The particular form of tractability exhibited by many stochastic processes in the literature is that of a *conjugate* posterior, that is, the posterior belongs to the same model family as the prior, and the posterior parameters can be computed as a function of the prior hyperparameters and the observed data. For example, the posterior of a $DP(\alpha, G_0)$ under observations $\theta_1, \ldots, \theta_n$ is again a Dirichlet process, $DP(\alpha + n, \frac{1}{\alpha+n}(\alpha G_0 + \sum \delta_{\theta_i}))$. Similarly the posterior of a GP under observations of $f(x_1), \ldots, f(x_n)$ is still a GP. It is this conjugacy that allows practical inference in the examples above. A Bayesian nonparametric model is conjugate if and only if the elements of its weak distribution, i.e., its finite-dimensional marginals, have a conjugate structure as well (Orbanz, 2010). In particular, this characterizes a class of conjugate Bayesian nonparametric models whose weak distributions consist of exponential family models. Note however, that lack of conjugacy does not imply intractable posteriors. An example is given by the Pitman–Yor process in which the posterior is given by a sum of a finite number of atoms and a Pitman-Yor process independent from the atoms.

Future Directions

Since MCMC (see ▶Markov Chain Monte Carlo) sampling algorithms for Dirichlet process mixtures became available in the 1990s and made latent variable models with nonparametric Bayesian components applicable to practical problems, the development of Bayesian nonparametrics has experienced explosive growth (Escobar & West, 1995; Neal, 2000). Arguably, though, the results available so far have only scratched the surface. The repertoire of available models is still mostly limited to using the Gaussian process, the Dirichlet process, the beta process, and generalizations derived from those. In principle, Bayesian nonparametric models may be defined on any infinite-dimensional mathematical object of possible interest to machine learning and statistics. Possible examples are kernels, infinite graphs, special classes of functions (e.g., piece-wise continuous or Sobolev functions), and permutations.

Aside from the obvious modeling questions, two major future directions are to make Bayesian nonparametric methods available to a larger audience of researchers and practitioners through the development of software packages, and to understand and quantify the theoretical properties of available methods.

General-Purpose Software Package

There is currently significant growth in the application of Bayesian nonparametric models across a

variety of application domains both in machine learning and in statistics. However significant hurdles still exist, especially the expense and expertise needed to develop computer programs for inference in these complex models. One future direction is thus the development of software packages that can compile efficient inference algorithms automatically given model specifications, thus allowing a much wider range of modeler to make use of these models. Current developments include the R DPpackage (http://cran.r-project.org/web/packages/DPpackage), the hierarchical Bayesian compiler (http://www.cs.utah.edu/hal/HBC), adaptor grammars (http://www.cog.brown.edu/mj/Software.htm), the MIT-Church project (http://projects.csail.mit.edu/church/wiki/Church), as well as efforts to add Bayesian nonparametric models to the repertoire of current Bayesian modeling environments like OpenBugs (http://mathstat.helsinki.fi/openbugs) and infer.NET (http://research.microsoft.com/en-us/um/cambridge/projects/infernet).

Statistical Properties of Models

Recent work in mathematical statistics provides some insight into the quantitative behavior of Bayesian nonparametric models (cf theory section). The elegant, methodical approach underlying these results, which quantifies model complexity by means of empirical process theory and then derives convergence rates as a function of the complexity, should be applicable to a wide range of models. So far, however, only results for Gaussian processes and Dirichlet process mixtures have been proven, and it will be of great interest to establish properties for other priors. Some models developed in machine learning, such as the infinite HMM, may pose new challenges to theoretical methodology, since their study will probably have to draw on both the theory of algorithms and mathematical statistics. Once a wider range of results is available, they may in turn serve to guide the development of new models, if it is possible to establish how different methods of model construction affect the statistical properties of the constructed model.

In addition to the references embedded in the text above, we recommend the books Hjort, Holmes, Müller, and Walker (2010), Ghosh and Ramamoorthi (2002), and the review articles Walker, Damien, Laud, and Smith (1999), Müller and Quintana (2004) on Bayesian nonparametrics. Further references can be found in the chapter by they Teh and Jordan (2010) of the book Hjort et al. (2010).

Cross References

►Bayesian Methods
►Dirichlet Processes
►Gaussian Processes
►Mixture Modelling
►Prior Probabilities

Recommended Reading

Diaconis, P., & Freedman, D. (1986) On the consistency of Bayes estimates (with discussion). *Annals of Statistics, 14*(1), 1–67.

Dunson, D. B. (2010). Nonparametric Bayes applications to biostatistics. In N. Hjort, C. Holmes, P. Müller, & S. Walker (Eds.), *Bayesian nonparametrics*. Cambridge: Cambridge University Press.

Escobar, M. D., & West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association, 90*, 577–588.

de Finetti, B. (1931). Funzione caratteristica di un fenomeno aleatorio. *Atti della R. Academia Nazionale dei Lincei, Serie 6. Memorie, Classe di Scienze Fisiche, Mathematice e Naturale, 4*, 251–299.

Ghosh, J. K., & Ramamoorthi, R. V. (2002). *Bayesian nonparametrics*. New York: Springer.

Hjort, N., Holmes, C., Müller, P., & Walker, S. (Eds.) (2010). Bayesian nonparametrics. In *Cambridge series in statistical and probabilistic mathematics* (No. 28). Cambridge: Cambridge University Press.

Müller, P., & Quintana, F. A. (2004). Nonparametric Bayesian data analysis. *Statistical Science, 19*(1), 95–110.

Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics, 9*, 249–265.

Orbanz, P. (2010). Construction of nonparametric Bayesian models from parametric Bayes equations. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, & A. Culotta (Eds.), *Advances in neural information processing systems, 22*, 1392–1400.

Teh, Y. W., & Jordan, M. I. (2010). Hierarchical Bayesian nonparametric models with applications. In N. Hjort, C. Holmes, P. Müller, & S. Walker (Eds.), *Bayesian nonparametrics*. Cambridge: Cambridge University Press.

Walker, S. G., Damien, P., Laud, P. W., & Smith, A. F. M. (1999). Bayesian nonparametric inference for random distributions and related functions. *Journal of the Royal Statistical Society, 61*(3), 485–527.

Wasserman, L. (2006). *All of nonparametric statistics*. New York: Springer.

Bayesian Reinforcement Learning

Pascal Poupart
University of Waterloo, Waterloo, Ontario, Canada

Synonyms

Adaptive control processes; Bayes adaptive Markov decision processes; Dual control; Optimal learning

Definition

Bayesian reinforcement learning refers to ►reinforcement learning modeled as a Bayesian learning problem (see ►Bayesian Methods). More specifically, following Bayesian learning theory, reinforcement learning is performed by computing a posterior distribution on the unknowns (e.g., any combination of the transition probabilities, reward probabilities, value function, value gradient, or policy) based on the evidence received (e.g., history of past state–action pairs).

Motivation and Background

Bayesian reinforcement learning can be traced back to the 1950s and 1960s in the work of Bellman (1961), Fel'Dbaum (1965), and several of Howard's students (Martin, 1967). Shortly after ►Markov decision processes were formalized, the above researchers (and several others) in Operations Research considered the problem of controlling a Markov process with uncertain transition and reward probabilities, which is equivalent to reinforcement learning. They considered Bayesian techniques since Bayesian learning is performed by probabilistic inference, which naturally combines with decision theory. In general, Bayesian reinforcement learning distinguishes itself from other reinforcement learning approaches by the use of probability distributions (instead of point estimates) to fully capture the uncertainty. This enables the learner to make more informed decisions, with the potential of learning faster with less data. In particular, the exploration/exploitation tradeoff can be naturally optimized. The use of a prior distribution also facilitates the encoding of domain knowledge, which is exploited in a natural and principled way by the learning process.

Structure of Learning Approach

A Markov decision process (MDP) (Puterman, 1994) can be formalized by a tuple $\langle S, A, T \rangle$ where S is the set of states s, A is the set of actions a, $T(s,a,s') = \Pr(s'|s,a)$ is the transition distribution indicating the probability of reaching s' when executing a in s. Let s_r denote the reward feature of a state and $\Pr(s'_r|s,a)$ be the probability of earning r when executing a in s. A policy $\pi : S \rightarrow A$ consists of a mapping from states to actions. For a given discount factor $0 \leq \gamma \leq 1$ and horizon h, the value V^π of a policy π is the expected discounted total reward earned while executing this policy: $V^\pi(s) = \sum_{t=o}^{h} \gamma^t E_{s|\pi}[s_r^t]$. The value function V^π can be written in a recursive form as the expected sum of the immediate reward s'_r with the discounted future rewards: $V^\pi(s) = \sum_{s'} \Pr(s'|s,\pi(s))[s'_r + \gamma V^\pi(s')]$. The goal is to find an optimal policy π^*, that is, a policy with the highest value V^* in all states (i.e., $V^*(s) \geq V^\pi(s)\ \forall \pi, s$). Many algorithms exploit the fact that the optimal value function V^* satisfies Bellman's equation:

$$V^*(s) = \max_a \sum_{s'} \Pr(s'|s,a)[s'_r + \gamma V^*(s)] \qquad (1)$$

Reinforcement learning (Sutton & Barto, 1998) is concerned with the problem of finding an optimal policy when the transition (and reward) probabilities T are unknown (or uncertain). Bayesian learning is a learning approach in which unknowns are modeled as random variables X over which distributions encode the uncertainty. The process of learning consists of updating the prior distribution $\Pr(X)$ based on some evidence e to obtain a posterior distribution $\Pr(X|e)$ according to Bayes theorem: $\Pr(X|e) = k \Pr(X) \Pr(e|X)$. (Here $k = 1/\Pr(e)$ is a normalization constant.) Hence, *Bayesian reinforcement learning* consists of using Bayesian learning for reinforcement learning. The unknowns are the transition (and reward) probabilities T, the optimal value function V^*, and the optimal policy π^*. Techniques that maintain a distribution on T are known as *model-based* Bayesian reinforcement learning since they explicitly learn the underlying model T. In contrast, techniques that maintain a distribution on V^* or π^* are known as *model-free* Bayesian reinforcement learning since they directly learn the optimal value function or policy without learning a model.

Model-Based Bayesian Learning

In model-based Bayesian reinforcement learning, the learner starts with a prior distribution over the parameters of T, which we denote by θ. For instance, let $\theta_{sas'} = \Pr(s'|s,a,\theta)$ be the unknown probability of reaching s' when executing a in s. In general, we denote by θ the set of all $\theta_{sas'}$. Then, the prior $b(\theta)$ represents the initial *belief* of the learner regarding the underlying model. The learner updates its belief after every s,a,s' triple observed by computing a posterior $b_{sas'}(\theta) = b(\theta|s,a,s')$ according to Bayes theorem:

$$b_{sas'}(\theta) = kb(\theta)\Pr(s'|s,a,\theta) = kb(\theta)\theta_{sas'}. \quad (2)$$

In order to facilitate belief updates, it is convenient to pick the prior from a family of distributions that is closed under Bayes updates. This ensures that beliefs are always parameterized in the same way. Such families are called *conjugate priors*. In the case of a discrete model (i.e., $\Pr(s'|s,a,\theta)$ is a discrete distribution), Dirichlets are conjugate priors and form a family of distributions corresponding to monomials over the simplex of discrete distributions (DeGroot, 1970). They are parameterized as follows: $Dir(\theta;n) = k\prod_i \theta_i^{n_i-1}$. Here θ is an unknown discrete distribution such that $\sum_i \theta_i = 1$ and n is a vector of strictly positive real numbers n_i (known as the hyperparameters) such that $n_i - 1$ can be interpreted as the number of times that the θ_i-probability event has been observed. Since the unknown transition model θ is made up of one unknown distribution θ_a^s per s,a pair, let the prior be $b(\theta) = \prod_{s,a} Dir(\theta_a^s; n_a^s)$ such that n_a^s is a vector of hyperparameters $n_a^{s,s'}$. The posterior obtained after transition $\hat{s},\hat{a},\hat{s}'$ is

$$\begin{aligned} b_a^{s,s'}(\theta) &= k\theta_a^{s,s'} \prod_{s,a} Dir(\theta_a^s; n_a^s) \\ &= \prod_{s,a} Dir(\theta_a^s; n_a^s + \delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')) \end{aligned} \quad (3)$$

where $\delta_{\hat{s},\hat{a},\hat{s}'}(s,a,s')$ is a Kronecker delta that returns 1 when $s = \hat{s}$, $a = \hat{a}$, $s' = \hat{s}'$ and 0 otherwise. In practice, belief monitoring is as simple as incrementing the hyperparameter corresponding to the observed transition.

Belief MDP Equivalence

At any point in time, the belief b provides an explicit representation of the uncertainty of the learner about the underlying model. This information is very useful to decide whether future actions should focus on exploring or exploiting. Hence, in Bayesian reinforcement learning, policies π are mappings from state-belief pairs $\langle s,b\rangle$ to actions. Equivalently, the problem of Bayesian reinforcement learning can be thought as one of planning with a belief MDP (or a partially observable MDP). More precisely, every Bayesian reinforcement learning problem has an equivalent belief MDP formulation $\langle S_{bel}, A_{bel}, T_{bel}\rangle$ where $S_{bel} = S \times B$ (B is the space of beliefs b), $A_{bel} = A$, and $T_{bel}(s_{bel}, a_{bel}, b'_{bel}) = \Pr(b'_{bel}|b_{bel}, a_{bel}) = \Pr(s',b'|s,b,a) = \Pr(b'|s,b,a,s')\Pr(s'|s,b,a)$. The decomposition of the transition dynamics is particularly interesting since $\Pr(b'|s,b,a,s')$ equals 1 when $b' = b_a^{s,s'}$ (as defined in Eq. 3) and 0 otherwise. Furthermore, $\Pr(s'|s,b,a) = \int_\theta b(\theta)\Pr(s'|s,\theta,a)d\theta$, which can be computed in closed form when b is a Dirichlet. As a result, the transition dynamics of the belief MDP are fully known. This is a remarkable fact since it means that Bayesian reinforcement learning problems, which by definition have unknown/uncertain transition dynamics, can be recast as belief MDPs with known transition dynamics. While this doesn't make the problem any easier since the belief MDP has a hybrid state space (discrete s with continuous b), it allows us to treat policy optimization as a problem of planning and in particular to adapt algorithms originally designed for belief MDPs (also known as partially observable MDPs).

Optimal Value Function Parameterization

Many planning techniques compute the optimal value function V^*, from which an optimal policy π^* can easily be extracted. Despite the hybrid nature of the state space, the optimal value function (for a finite horizon) has a simple parameterization corresponding to the upper envelope of a set of polynomials (Poupart, Vlassis, Hoey, & Regan, 2006). Recall that the optimal value function satisfies Bellman's equation, which can be adapted as follows for a belief MDP:

$$V^*(s,b) = \max_a \sum_{s'} \Pr(s',b'|s,b,a)\left[s'_r + \gamma V^*(s',b')\right]. \quad (4)$$

Using the fact that b' must be $b_a^{s,s'}$ (otherwise $\Pr(s',b'|s,b,a) = 0$) allows us to rewrite Bellman's equation as follows:

$$V^*(s,b) = \max_a \sum_{s'} \Pr(s'|s,b,a)\left[s'_r + \gamma V^*\left(s', b_a^{s,s'}\right)\right]. \quad (5)$$

Let Γ^n be a set of polynomials in θ such that the optimal value function V^n with n steps to go is $V^n(s,b) = \int_\theta b(\theta) poly_{s,b}(\theta) d\theta$ where $poly_{s,b} = \text{argmax}_{poly \in \Gamma_s^n} \int_\theta b(\theta) poly(\theta) d\theta$. It suffices to replace $\Pr(s'|s,b,a)$, $b_a^{s,s'}$ and V^n by their definitions in Bellman's equation

$$V^{n+1}(s,b) = \max_a \sum_{s'} \int_\theta b(\theta) \Pr(s'|s,\theta,a) \left[r'_s + \gamma\, poly_{s', b_a^{s,s'}}(\theta)\right] d\theta \quad (6)$$

$$= \max_a \int_\theta b(\theta) \sum_{s'} \theta_a^{s,s'} \left[r'_s + \gamma\, poly_{s', b_a^{s,s'}}(\theta)\right] d\theta \quad (7)$$

to obtain a similar set of polynomials $\Gamma_s^{n+1} = \left\{\sum_{s'} \theta_a^{s,s'} \left[r'_s + \gamma\, poly'_s(\theta)\right] | a \in A,\ poly_{s'} \in \Gamma_{s'}^n\right\}$ that represents V^{n+1}.

The fact that the optimal value function has a closed form with a simple parameterization is quite useful for planning algorithms based on value iteration. Instead of using an arbitrary function approximator to fit the value function, one can take advantage of the fact that the value function can be represented by a set of polynomials to choose a good representation. For instance, the Beetle algorithm (Poupart et al., 2006) performs point-based value iteration and approximates the value function with a bounded set of polynomials that each consists of a linear combination of monomial basis functions.

Exploration/Exploitation Tradeoff

Since the underlying model is unknown in reinforcement learning, it is not clear whether actions should be chosen to explore (gain more information about the model) or exploit (maximize immediate rewards based on information gathered so far). Bayesian reinforcement learning provides a principled solution to the exploration/exploitation tradeoff. Despite the appearance of multiple objectives induced by exploration and exploitation, there is a single objective in reinforcement learning: maximize total discounted rewards. Hence, an optimal policy (which maximizes total discounted rewards) must naturally optimize the exploration/exploitation tradeoff. In order for a policy to be optimal, it must use all the information available. The information available to the learner consists of the history of past states and actions. One can show that state–belief pairs $\langle s, b\rangle$ are sufficient statistics of the history. Hence, by searching for the mapping from state–belief pairs to actions that maximizes total discounted rewards, Bayesian reinforcement learning implicitly seeks an optimal tradeoff between exploration and exploitation. In contrast, traditional reinforcement learning approaches search in the space of mappings from states to actions. As a result, such techniques typically focus on asymptotic convergence (i.e., convergence to a policy that is optimal in the limit), but do not effectively balance exploration and exploitation since they do not use histories or beliefs to quantify the uncertainty about the underlying model.

Related Work

Michael Duff's PhD thesis (Duff, 2002) provides an excellent survey of Bayesian reinforcement learning up until 2002. The above text pertains mostly to model-based Bayesian reinforcement learning applied to discrete, fully observable, single agent domains. Bayesian learning has also been explored in model-free reinforcement learning (Dearden, Friedman, & Russell, 1998; Engel, Mannor, & Meir, 2005; Ghavamzadeh & Engel, 2006) continuous-valued state spaces (Ross, Chaib-Draa, & Pineau, 2008), partially observable domains (Poupart & Vlassis, 2008; Ross, Chaib-Draa, & Pineau, 2007), and multi-agent systems (Chalkiadakis & Boutilier, 2003, 2004; Gmytrasiewicz & Doshi, 2005).

Cross References

►Active Learning
►Markov Decision Processes
►Reinforcement Learning

Recommended Reading

Bellman, R. (1961). *Adaptive control processes: A guided tour*. Princeton, NJ: Princeton University Press.

Chalkiadakis, G., & Boutilier, C. (2003). Coordination in multiagent reinforcement learning: A Bayesian approach. In *International joint conference on autonomous agents and multiagent systems (AAMAS)*, Melbourne, Australia (pp. 709–716).
Chalkiadakis, G., & Boutilier, C. (2004). Bayesian reinforcement learning for coalition formation under uncertainty. In *International joint conference on autonomous agents and multiagent systems (AAMAS)*, New York (pp. 1090–1097).
Dearden, R., Friedman, N., & Russell, S. J. (1998). Bayesian Q-learning. In *National conference on artificial intelligence (AAAI)*, Madison, Wisconsin (pp. 761–768).
DeGroot, M. H. (1970). *Optimal statistical decisions*. New York: McGraw-Hill.
Duff, M. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts, Amherst.
Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. In *International conference on machine learning (ICML)*, Bonn, Germany.
Fel'Dbaum, A. (1965). *Optimal control systems*. New York: Academic.
Ghavamzadeh, M., & Engel, Y. (2006). Bayesian policy gradient algorithms. In *Advances in neural information processing systems (NIPS)*, (pp. 457–464).
Gmytrasiewicz, P., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR), 24*, 49–79.
Martin (1967). *Bayesian decision problems and Markov chains*. New York: Wiley.
Poupart, P., & Vlassis, N. (2008). Model-based Bayesian reinforcement learning in partially observable domains. In *International symposium on artificial intelligence and mathematics (ISAIM)*.
Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. In *International conference on machine learning (ICML)*, Pittsburgh, Pennsylvania (pp. 697–704).
Puterman, M. L. (1994). *Markov decision processes*. New York: Wiley.
Ross, S., Chaib-Draa, B., & Pineau, J. (2007). Bayes-adaptive POMDPs. In *Advances in neural information processing systems (NIPS)*.
Ross, S., Chaib-Draa, B., & Pineau, J. (2008). Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *IEEE International conference on robotics and automation (ICRA)*, (pp. 2845–2851).
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Beam Search

Claude Sammut
University of New South Wales, Sydney, Australia

A beam search is a heuristic search technique that combines elements of breadth-first and best-first searches. Like a breadth-first search, the beam search maintains a list of nodes that represent a frontier in the search space. Whereas the breadth-first adds all neighbors to the list, the beam search orders the neighboring nodes according to some heuristic and only keeps the n best, where n is the *beam size*. This can significantly reduce the processing and storage requirements for the search.

In machine learning, the beam search has been used in algorithms, such as AQ11 (Dietterich & Michalski, 1977).

Cross References

►Learning as Search

Recommended Reading

Dietterich, T. G., & Michalski, R. S. (1977). Learning and generalization of characteristic descriptions: Evaluation criteria and comparative review of selected methods. In *Fifth international joint conference on artificial intelligence* (pp. 223–231). Cambridge, MA: William Kaufmann.

Behavioral Cloning

Caude Sammut
The University of New South Wales, Sydney, Australia

Synonyms

Apprenticeship learning; Behavioral cloning; Learning by demonstration; Learning by imitation; Learning control rules

Definition

Behavioral cloning is a method by which human subcognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training.

Motivation and Background

Behavioral cloning (Michie, Bain, & Hayes-Michie, 1990) is a form of *learning by imitation* whose main motivation is to build a model of the behavior of a human when performing a complex skill. Preferably, the model should be in a readable form. It is related to other forms of learning by imitation, such as ▶inverse reinforcement learning (Abbeel & Ng, 2004; Amit & Matarić, 2002; Hayes & Demiris, 1994; Kuniyoshi, Inaba, & Inoue, 1994; Pomerleau, 1989) and methods that use data from human performances to model the system being controlled (Atkeson & Schaal, 1997; Bagnell & Schneider, 2001).

Experts might be defined as people who know what they are doing not what they are talking about. That is, once a person becomes highly skilled in some task, the skill becomes sub-cognitive and is no longer available to introspection. So when the person is asked to explain why certain decisions were made, the explanation is a *post hoc* justification rather than a true explanation.

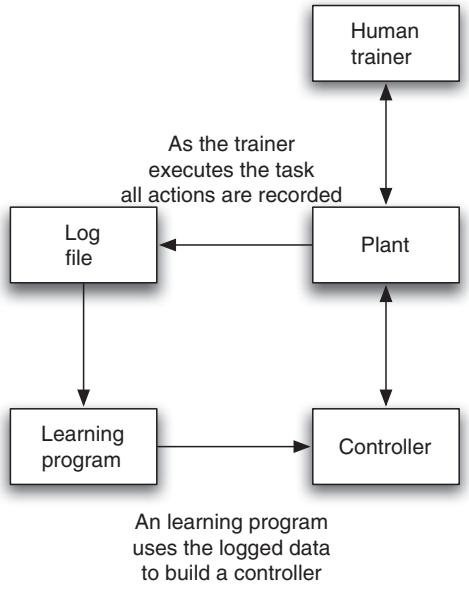
Michie et al. (1990) used an induction program to learn rules for balancing a pole (in simulation) and earlier work by Donaldson (1960), Widrow and Smith (1964), and Chambers and Michie (1969) demonstrated the feasibility of learning by imitation, also for pole-balancing.

Structure of the Learning System

Behavioral cloning assumes that there is a plant of some kind that is under the control of a human operator. The plant may be a physical system or a simulation. In either case, the plant must be instrumented so that it is possible to capture the state of the system, including all the control settings. Thus, whenever the operator performs an action, that is, changes a control setting, we can associate that action with a particular state.

Let us use a simple example of a system that has only one control action. A *pole balancer* has four state variables: the angle of the pole, θ, and its angular velocity, $\dot{\theta}$ and the position, x, and velocity $\dot{x}$, of the cart on the track. The only action available to the controller is to apply a fixed positive of negative force, F, to accelerate the cart left or right.

We can create an experimental setup where a human can control a pole and cart system (either real or in simulation) by applying a left push or a right push at



Behavioral Cloning. Figure 1. Structure of learning system

the appropriate time. Whenever a control action is performed, we record the action as well as values of the four state variables at the time of the action. Each of these records can be viewed as an example of a mapping from state to action.

Michie et al. (1990) demonstrated that it is possible to construct a controller by learning from these examples. The learning task is to predict the appropriate action, given the state. They used a ▶decision tree learning program to produce a classifier that, given the values of the four state variables, would output an action. A decision tree is easily convertible into an executable code as a nested **if** statement. The quality of the controller can be tested by inserting the decision tree into the simulator, replacing the human operator.

If the goal of learning is simply to produce an operational controller then any program capable of building a classifier could be used. The reason that Michie et al. (1990) chose a symbolic learner was their desire to produce a controller whose decision making was transparent as well as operational. That is, it should be possible to extract an explanation of the behavior that is meaningful to an expert in the task.

Learning Direct (Situation–Action) Controllers

A controller such as the one described above is referred to as a *direct controller* because it maps situations to actions. Other examples of learning a direct controller

are building an autopilot from behavioral traces of human pilots flying aircraft in a flight simulator (Sammut, Hurst, Kedzier, & Michie, 1992) and building a control system for a container crane (Urbančič & Bratko, 1994). These systems extended the earlier work by operating in domains in which there is more than one control variable and the task is sufficiently complex that it must be decomposed into several subtasks.

An operator of a container crane can control the speed of the cart and the length of the rope. A pilot of a fixed-wing aircraft can control the ailerons, elevators, rudder, throttle, and flaps. To build an autopilot, the learner must build a system that can set each of the control variables. Sammut et al. (1992), viewed this as a multitask learning problem.

Each training example is a feature vector that includes the position, orientation, and velocities of the aircraft as well as the values of each of the control settings: ailerons, elevator, throttle, and flaps. The rudder is ignored. A separate decision tree is built for each control variable. For example, the aileron setting is treated as the dependent variable and all the other variables, including the other controls, are treated as the attributes of the training example. A decision tree is built for ailerons, then the process is repeated for the elevators, etc. The result is a decision tree for each control variable.

The autopilot code executes each decision tree in each cycle of the control loop. This method treats the setting of each control as a separate task. It may be surprising that this method works since it is often necessary to adjust more than one control simultaneously to achieve the desired result. For example, to turn, it is normal to use the ailerons to roll the aircraft while adjusting the elevators to pull it around. This kind of multivariable control does result from multiple decision trees. When, say, the aileron decision tree initiates a roll, the elevator's decision tree detects the roll and causes the aircraft to pitch up and execute a turn.

Limitations Direct controllers work quite well for systems that have a relatively small state space. However, for complex systems, behavioral cloning of direct situation–action rules tends to produce very brittle controllers. That is, they cannot tolerate large disturbances. For example, when air turbulence is introduced into the flight simulator, the performance of the clone degrades very rapidly. This is because the examples provided by logging the performance of a human only cover a very small part of the state space of a complex system such as an aircraft in flight. Thus, the"expertise" of the controller is very limited. If the system strays outside the controller's region of expertise, it has no method for recovering and failure is usually catastrophic.

More robust control is possible but only with a significant change in approach. The more successful methods decompose the learning task into two stages: learning goals and learning the actions to achieve those goals.

Learning Indirect (Goal-Directed) Controllers

The problem of learning in a large search space can partially be addressed by decomposing the learning into subtasks. A controller built in this way is said to be an *indirect controller*. A control is "indirect" if it does not compute the next action directly from the system's current state but uses, in addition, some intermediate information. An example of such intermediate information is a subgoal to be attained before achieving the final goal.

Subgoals often feature in an operator's control strategies and can be automatically detected from a trace of the operator's behavior (Šuc & Bratko, 1997). The problem of subgoal identification can be treated as the inverse of the usual problem of controller design, that is, given the actions in an operator's trace, find the goal that these actions achieve. The limitation of this approach is that it only works well for cases in which there are just a few subgoals, not when the operator's trajectory contains many subgoals. In these cases, a better approach is to generalize the operator's trajectory. The generalized trajectory can be viewed as defining a continuously changing subgoal (Bratko & Šuc, 2002; Šuc & Bratko, 1999a) (see also the use of flow tubes in dynamic plan execution (Hofmann & Williams, 2006)).

Subgoals and generalized trajectories are not sufficient to define a controller. A model of the systems dynamics is also required. Therefore, in addition to inducing subgoals or a generalized trajectory, this approach also requires learning approximate system dynamics, that is a model of the controlled system. Bratko and Šuc (2003) and Šuc and Bratko (1999b) use a combination of the Goldhorn (Križman & Džeroski,

1995) discovery program and locally weighted regression to build the model of the system's dynamics. The next action is then computed "indirectly" by (1) computing the desired next state (e.g., next subgoal) and (2) determining an action that brings the system to the desired next state. Bratko and Šuc also investigated building qualitative control strategies from operator traces (Bratko & Šuc, 2002).

An analog to this approach is ►inverse reinforcement learning (Abbeel & Ng, 2004; Atkeson & Schaal, 1997; Ng & Russell, 2000) where the reward function is learned. Here, the learning the reward function corresponds to learning the human operator's goals.

Isaac and Sammut (2003) uses an approach that is similar in spirit to Šuc and Bratko but incorporates classical control theory. Learned skills are represented by a two-level hierarchical decomposition with an anticipatory goal level and a reactive control level. The goal level models how the operator chooses goal settings for the control strategy and the control level models the operator's reaction to any error between the goal setting and actual state of the system. For example, in flying, the pilot can achieve goal values for the desired heading, altitude, and airspeed by choosing appropriate values of turn rate, climb rate, and acceleration. The controls can be set to correct errors between the current state and the desired state of these goal-directing quantities. Goal models map system states to a goal setting. Control actions are based on the error between the output of each of the goal models and the current system state.

The control level is modeled as a set of proportional integral derivative (PID) controllers, one for each control variable. A PID controller determines a control value as a linear function proportional to the error on a goal variable, the integral of the error, and the derivative of the error.

Goal setting and control models are learned separately. The process begins be deciding which variables are to be used for the goal settings. For example, trainee pilots will learn to execute a "constant-rate turn," that is, their goal is to maintain a given turn rate. A separate goal rule is constructed for each goal variable using a ►model tree learner (Potts & Sammut, 2005).

A goal rule gives the setting for a goal variable and therefore, we can find the difference (error) between the current state value and the goal setting. The integral and derivative of the error can also be calculated. For example, if the set turn rate is 180° min, then the error on the turn rate is calculated as the actual turn rate minus 180. The integral is then the running sum of the error multiplied by the time interval between time samples, starting from the first time sample of the behavioral trace, and the derivative is calculated as the difference between the error and previous error all divided by the time interval.

For each control available to the operator, a model tree learner is used to predict the appropriate control setting. ►Linear regression is used in the leaf nodes of the model tree to produce linear equations whose coefficients are the *P*, *I*, and *D* of values of the PID controller. Thus the learner produces a collection of PID controllers that are selected according to the conditions in the internal nodes of the tree. In control theory, this is known as *piecewise linear control*.

Another indirect method is to learn a model of the dynamics of the system and use this to learn, in simulation, a controller for the system (Bagnell & Schneider, 2001; Ng, Jin Kim, Jordan, & Sastry, 2003). This approach does not seek to directly model the behavior of a human operator. A behavioral trace may be used to generate data for modeling the system but then a reinforcement learning algorithm is used to generate a policy for controlling the simulated system. The learned policy can then be transferred to the physical system. ►Locally weighted regression is typically used for system modeling, although ►model trees can also be used.

Cross References

►Apprenticeship Learning
►Inverse Reinforcement Learning
►Learning by Imitation
►Locally Weighted Regression
►Model Trees
►Reinforcement Learning
►System Identification

Recommended Reading

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International conference on machine learning*, Banff, Alberta, Canada. New York: ACM.

Amit, R., & Matarić, M. (2002). Learning movement sequences from demonstration. In *Proceedings of the second international conference on development and learning*, Cambridge, MA, USA (pp. 203–208). Washington, D.C.: IEEE.

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. In D. H. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning*, Nashville, TN, USA (pp. 12–20). San Francisco: Morgan Kaufmann.

Bagnell, J. A., & Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *International conference on robotics and automation*, South Korea. IEEE Press, New York.

Bratko, I., & Šuc, D. (2002). Using machine learning to understand operator's skill. In *Proceedings of the 15th international conference on industrial and engineering applications of artificial intelligence and expert systems* (pp. 812–823). London: Springer. AAAI Press, Menlo Park, CA.

Bratko, I., & Šuc, D. (2003). Learning qualitative models. *AI Magazine, 24*(4), 107–119.

Chambers, R. A., & Michie, D. (1969). Man-machine co-operation on a learning task. In R. Parslow, R. Prowse, & R. Elliott-Green (Eds.), *Computer graphics: techniques and applications*. London: Plenum.

Donaldson, P. E. K. (1960). Error decorrelation: A technique for matching a class of functions. In *Proceedings of the third international conference on medical electronics* (pp. 173–178).

Hayes, G., & Demiris, J. (1994). A robot controller using learning by imitation. In *Proceedings of the international symposium on intelligent robotic systems*, Grenoble, France (pp. 198–204). Grenoble: LIFTA-IMAG.

Hofmann, A. G., & Williams, B. C. (2006). Exploiting spatial and temporal flexiblity for plan execution of hybrid, under-actuated systems. In *Proceedings of the 21st national conference on artficial intelligence*, July 2006, Boston, MA (pp. 948–955).

Isaac, A., & Sammut, C. (2003). Goal-directed learning to fly. In T. Fawcett & N. Mishra (Eds.), *Proceedings of the twentieth international conference on machine learning*, Washington, D.C. (pp. 258–265). Menlo Park: AAAI.

Križman, V., & Džeroski, S. (1995). Discovering dynamics from measured data. *Electrotechnical Review, 62*(3–4), 191–198.

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation, 10*, 799–822.

Michie, D., Bain, M., & Hayes-Michie, J. E. (1990). Cognitive models from subcognitive skills. In M. Grimble, S. McGhee, & P. Mowforth (Eds.), *Knowledge-based systems in industrial control*. Stevenage: Peter Peregrinus.

Ng, A. Y., Jin Kim, H., Jordan, M. I., & Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems 16*. Cambridge: MIT Press.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of 17th international conference on machine learning*, Stanford, CA, USA (pp. 663–670). San Francisco: Morgan Kaufmann.

Pomerleau, D. A. (1989). ALVINN: An autonomous land vehicle in a neural network. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*. San Mateo: Morgan Kaufmann.

Potts, D., & Sammut, C. (November 2005). Incremental learning of linear model trees. *Machine Learning, 6*(1–3), 5–48.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the ninth international conference on machine learning*, Aberdeen (pp. 385–393). San Francisco: Morgan Kaufmann.

Šuc, D., & Bratko, I. (1997). Skill reconstruction as induction of LQ controllers with subgoals. In *IJCAI-97: Proceedings of the fifteenth international joint conference on artificial intelligence*, Nagoya, Japan (Vol. 2, pp. 914–920). San Francisco: Morgan Kaufmann.

Šuc, D., & Bratko, I. (1999a). Modelling of control skill by qualitative constraints. In *Thirteenth international workshop on qualitative reasoning*, 7–9 June 1999, Lock Awe, Scotland (pp. 212–220). Aberystwyth: University of Aberystwyth.

Šuc, D., & Bratko, I. (1999b). Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence, 3*(B), 1–22.

Urbančič, T., & Bratko, I. (1994). Reconstructing human skill with machine learning. In A. Cohn (Ed.), *Proceedings of the 11th European conference on artificial intelligence*. Wiley. Amsterdam: New York.

Widrow, B., & Smith, F. W. (1964). Pattern recognising control systems. In J. T. Tou & R. H. Wilcox (Eds.), *Computer and information sciences*. London: Clever Hume.

Belief State Markov Decision Processes

►Partially Observable Markov Decision Processes

Bellman Equation

The *Bellman Equation* is a recursive formula that forms the basis for ►dynamic programming. It computes the expected total reward of taking an action from a state in a ►Markov decision process by breaking it into the immediate reward and the total future expected reward. (See ►dynamic programming.)

Bias

Bias has two meanings, ►inductive bias, and *statistical bias* (see ►bias variance decomposition).

Bias Specification Language

Hendrik Blockeel
Katholieke Universiteit Leuven, Belgium
The Netherlands

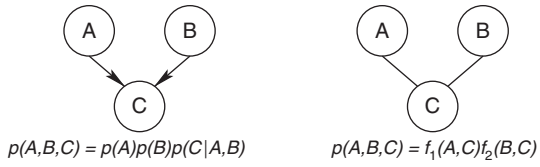
Definition

A *bias specification language* is a language in which a user can specify a ►Language Bias. The language bias of a learner is the set of hypotheses (or hypothesis descriptions) that this learner may return.

In contrast to the ►hypothesis language, the bias specification language allows us to describe not single hypotheses but sets (languages) of hypotheses.

Examples

In learning approaches based on ►graphical models or ►artificial neural networks, whenever the user provides the graph structure of the model, he or she is specifying a bias. The "language" used to specify this bias, in this case, consists of graphs. Figure 1 shows examples of such graphs. Not every kind of bias can necessarily be expressed by some bias specification language; for instance, the bias defined by the ►Bayesian network structure in Fig. 1 cannot be expressed using a ►Markov network. Bayesian networks and Markov networks have a different expressiveness, when viewed as bias specification languages.



Bias Specification Language. Figure 1. Graphs defining a bias for learning joint distributions. The Bayesian network structure to the left constrains the form of the joint distribution in a particular way (shown as the equation below the graph). Notably, it guarantees that only distributions can be learned in which the variables *A* and *B* are (unconditionally) independent. The Markov network structure to the right constrains the form of the joint distribution in a different way: it states that it must be possible to write the distribution as a product of a function of *A* and *C* and a function of *B* and *C*. These two biases are different. In fact, no Markov network structure over the variables *A*, *B*, and *C* exists that expresses the bias specified by the Bayesian network structure

Also certain parameters of decision tree learners or rule set learners effectively restrict the hypothesis language (for instance, an upper bound on the rule length or the size of the decision tree).

A combination of parameter values can hardly be called a language, and even the "language" of graphs is a relatively simple kind of language. More elaborate types of bias specification languages are typically found in the field of ►inductive logic programming (ILP).

Bias Specification Languages in Inductive Logic Programming

In ILP, the hypotheses returned by the learning algorithm are typically written as first-order logic clauses. As the set of all possible clauses is too large to handle, a subset of these clauses is typically defined; this subset is called the language bias. Several formalisms ("bias specification languages") have been proposed for specifying such subsets. We here focus on a few representative ones.

DLAB

In the DLAB bias specification language (Dehaspe & De Raedt, 1996), the language bias is defined in a declarative way, by defining a syntax that clauses must fulfill. In its simplest form, a DLAB specification simply gives a set of possible head and body literals out of which the system can build a clause.

Example 1 *The actual syntax of the DLAB specification language is relatively complicated (see Dehaspe & De Raedt, 1996), but in essence, one can write down a specification such as:*

```
{ parent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z},
   {X,Y,Z}) }
:-
{ parent({X,Y,Z},{X,Y,Z}),
  parent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z},{X,Y,Z}),
  grandparent({X,Y,Z}, {X,Y,Z}) }
```

which states that the hypothesis language consists of all clauses that have at most one `parent` *and at most one*

`grandparent` *literal in the head, and at most two of these literals in the body; the arguments of these literals may be variables* `X`, `Y`, `Z`. *Thus, the following clauses are in the hypothesis language:*

```
grandparent(X, Y) :- parent(X, Z),
  parent(Z,Y)
:- parent(X,Y), parent(Y,X)
:- parent(X,X)
```

These express the usual definition of grandparent as well as the fact that there can be no cycles in the parent relation.

Note that for each argument of each literal, all the variables and constants that may occur have to be enumerated explicitly. This can make a DLAB specification quite complex. While DLAB contains advanced constructs to alleviate this problem, it remains the case that often very elaborate bias specifications are needed in practical situations.

Type- and Mode-Based Biases

A more flexible bias specification language is used by Progol (Muggleton, 1995) and many other ILP systems. It is based on the notions of types and modes. In Progol, arguments of a predicate can be typed, and a variable can never occur in two locations with different types. Similarly, arguments of a predicate have an input (+) or output (−) mode; each variable that occurs as an input argument of some literal must occur elsewhere as an output argument, or must occur as input argument in the head literal of a clause.

Example 2 *In Progol, the specifications*

```
type(parent(human,human)).
type(grandparent(human,human)).
modeh(grandparent(+,+)).
  % modeh: specifies a head literal
modeb(grandparent(+,-)).
  % modeb: specifies a body literal
modeb(parent(+,-)).
```

allow the system to construct a clause such as

```
grandparent(X,Y) :- parent(X,Z),
  parent(Z,Y)
```

but not the following clause:

```
grandparent(X,Y) :- parent(Z,Y)
```

because `Z` *occurs as an input parameter for* `parent` *without occurring elsewhere as an output parameter (i.e., it is being used without having been given a value first).*

FLIPPER's Bias Specification Language

The FLIPPER system (Cohen, 1996) employs a powerful, but somewhat more procedural, bias specification formalism. The user does not specify a set of valid hypotheses directly, but rather, specifies a ▶Refinement Operator. The language bias is the set of all clauses that can be obtained from one or more starting clauses through repeated application of this refinement operator. The operator itself is defined by specifying under which conditions certain literals can be added to a clause.

Rules defining the operator have one of the following forms:

$A \leftarrow B$ `where` *Pre* `asserting` *Post*
L `where` *Pre* `asserting` *Post*

The first form defines a set of "starting clauses," and the second form defines when a literal L can be added to a clause. Each rule can only be applied when its preconditions *Pre* are fulfilled, and upon application will assert a set of literals *Post*. As an example (Cohen, 1996), the rules

$$\begin{aligned} &\mathit{illegal}(A,B,C,D,E,F) \leftarrow \\ &\quad \texttt{where}\ \mathit{true} \\ &\quad \texttt{asserting}\ \{\mathit{linked}(A), \mathit{linked}(B), \ldots, \\ &\qquad \mathit{linked}(F)\} \end{aligned}$$

$$R(X,Y)\ \texttt{where}\ \mathit{rel}(R), \mathit{linked}(X), \mathit{linked}(Y)\ \texttt{asserting}\ \varnothing$$

state that any clause of the form

$$\mathit{illegal}(A,B,C,D,E,F) \leftarrow$$

can be used as a starting point for the refinement operator, and the variables in this clause are all *linked*. Further, any literal of the form $R(X,Y)$ with R a relation

symbol (as defined by the *Rel* predicate) and *X* and *Y* linked variables can be added.

Other Approaches

Grammars or term rewriting systems have been proposed several times as a means of defining the hypothesis language. A relatively recent approach along these lines was given by Lloyd, who uses a rewriting system to define the tests that can occur in the nodes of a decision tree built by the Alkemy system (Lloyd, 2003).

Boström & Idestam-Almquist (1999) present an approach where the language bias is implicitly defined through the ▶Background Knowledge given to the learner.

Knobbe et al. (2000) propose the use of UML as a "common" bias specification language, specifications in which could be translated automatically to languages specific to a particular learner.

Further Reading

An overview of bias specification formalisms in ILP is given by Nédellec et al. (1996). The bias specification languages discussed above are discussed in more detail in Dehaspe and De Raedt (1996), Muggleton (1995), and Cohen (1996). De Raedt (1992) discusses language bias and the concept of bias shift (learners weakening their bias, i.e., extending the set of hypotheses that can be represented, when a given language bias turns out to be too restrictive). A more recent approach to learning declarative bias is presented by Bridewell and Todorovski (2008).

Cross References

▶Hypothesis Language
▶Inductive Logic Programminllg

Recommended Reading

Boström, H., & Idestam-Almquist, P. (1999). Induction of logic programs by example-guided unfolding. *Journal of Logic Programming, 40*(2–3), 159–183.

Bridewell, W., & Todorovski, L. (2008). Learning declarative bias. In *Proceedings of the 17th international conference on inductive logic programming. Lecture notes in computer science* (Vol. 4894, pp. 63–77). Berlin: Springer.

Cohen, W. (1996). Learning to classify English text with ILP methods. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 124–143). Amsterdam: IOS Press.

De Raedt, L. (1992). *Interactive theory revision: An inductive logic programming approach*. New York: Academic Press.

Dehaspe, L., & De Raedt, L. (1996). DLAB: A declarative language bias formalism. In *Proceedings of the international symposium on methodologies for intelligent systems. Lecture notes in artificial intelligence* (Vol. 1079, pp. 613–622). Berlin: Springer.

Knobbe, A. J., Siebes, A., Blockeel, H., & van der Wallen, D. (2000). Multi-relational data mining, using UML for ILP. In *Proceedings of PKDD-2000 – The fourth European conference on principles and practice of knowledge discovery in databases. Lecture notes in artificial intelligence* (Vol. 1910, pp. 1–12), Lyon, France. Berlin: Springer.

Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, Special Issue on Inductive Logic Programming, 13*(3–4), 245–286.

Nédellec, C., Adé, H., Bergadano, F., & Tausend, B. (1996). Declarative bias in ILP. In L. De Raedt (Ed.), *Advances in inductive logic programming. Frontiers in artificial intelligence and applications* (Vol. 32, pp. 82–103). Amsterdam: IOS Press.

Bias Variance Decomposition

Definition

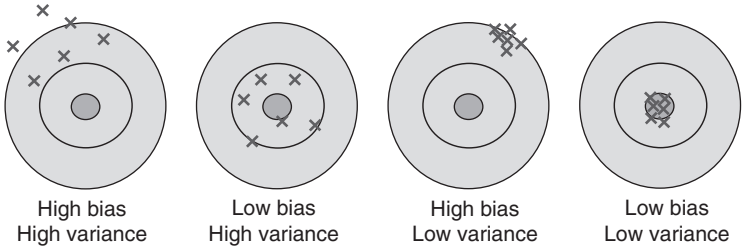
The bias-variance decomposition is a useful theoretical tool to understand the performance characteristics of a learning algorithm. The following discussion is restricted to the use of *squared loss* as the performance measure, although similar analyses have been undertaken for other loss functions. The case receiving most attention is the zero-one loss (i.e., classification problems), in which case the decomposition is nonunique and a topic of active research. See Domingos (1992) for details.

The decomposition allows us to see that the mean squared error of a model (generated by a particular learning algorithm) is in fact made up of two components. The *bias* component tells us how accurate the model is, on average across different possible training sets. The *variance* component tells us how sensitive the learning algorithm is to small changes in the training set (Fig. 1).

Mathematically, this can be quantified as a decomposition of the mean squared error function. For a testing example $\{\mathbf{x}, d\}$, the decomposition is:

$$\mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x}) - d)^2\} = (\mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\} - d)^2 + \mathcal{E}_{\mathcal{D}}\{(f(\mathbf{x}) - \mathcal{E}_{\mathcal{D}}\{f(\mathbf{x})\})^2\},$$

$$\text{MSE} = \text{bias}^2 + \text{variance},$$



Bias Variance Decomposition. Figure 1. The bias-variance decomposition is like trying to hit the bullseye on a dartboard. Each dart is thrown after training our "dart-throwing" model in a slightly different manner. If the darts vary wildly, the learner is *high variance*. If they are far from the bullseye, the learner is *high bias*. The ideal is clearly to have both low bias and low variance; however this is often difficult, giving an alternative terminology as the bias-variance "dilemma" (*Dartboard analogy*, Moore & McCabe (2002))

where the expectations are with respect to all possible training sets. In practice, this can be estimated by cross-validation over a single finite training set, enabling a deeper understanding of the algorithm characteristics. For example, efforts to reduce variance often cause increases in bias, and vice versa. A large bias and low variance is an indicator that a learning algorithm is prone to ►overfitting the model.

Cross References

►Bias-Variance Trade-offs: Novel Applications

Recommended Reading

Domingos, P. (1992). A unified bias-variance decomposition for zero-one and squared loss. In *Proceedings of national conference on artificial intelligence*. Austin, TX: AAAI Press.

Geman, S. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*(1)

Moore, D. S., & McCabe, G. P. (2002). Introduction to the practice of statistics. Michelle Julet

Bias-Variance Trade-offs: Novel Applications

Dev Rajnarayan, David Wolpert
NASA Ames Research Center, Moffett Field, CA, USA

Definition

Consider a given random variable $\underline{F}$ and a random variable that we can modify, $\underline{\hat{F}}$. We wish to use a sample of $\underline{\hat{F}}$ as an estimate of a sample of $\underline{F}$. The mean squared error (MSE) between such a pair of samples is a sum of four terms. The first term reflects the statistical coupling between $\underline{F}$ and $\underline{\hat{F}}$ and is conventionally ignored in bias-variance analysis. The second term reflects the inherent noise in $\underline{F}$ and is independent of the estimator $\underline{\hat{F}}$. Accordingly, we cannot affect this term. In contrast, the third and fourth terms depend on $\underline{\hat{F}}$. The third term, called the bias, is independent of the precise samples of both $\underline{F}$ and $\underline{\hat{F}}$, and reflects the difference between the means of $\underline{F}$ and $\underline{\hat{F}}$. The fourth term, called the variance, is independent of the precise sample of $\underline{F}$, and reflects the inherent noise in the estimator as one samples it. These last two terms can be modified by changing the choice of the estimator. In particular, on small sample sets, we can often decrease our mean squared error by, for instance, introducing a small bias that causes a large reduction the variance. While most commonly used in machine learning, this article shows that such bias-variance trade-offs are applicable in a much broader context and in a variety of situations. We also show, using experiments, how existing bias-variance trade-offs can be applied in novel circumstances to improve the performance of a class of optimization algorithms.

Motivation and Background

In its simplest form, the bias-variance decomposition is based on the following idea. Say we have a random variable $\underline{F}$ taking on values F distributed according to a density function $p(F)$. We want to estimate the value of a sample from $p(F)$. To form our estimate, we sample a different random variable $\underline{\hat{F}}$ taking on values $\hat{F}$ distributed according to $p(\hat{F})$. Assuming a quadratic loss function, the quality of our estimate is measured by its MSE:

$$\mathrm{MSE}(\hat{\underline{F}}) \equiv \int p(\hat{F}, F)\,(\hat{F} - F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F.$$

In many situations, $\underline{F}$ and $\hat{\underline{F}}$ are dependent variables. For example, in supervised machine learning, $\underline{F}$ is a "target" conditional distribution, stochastically mapping elements of an input space X into a space Y of output variables. The associated distribution $p(F)$ is the "prior" of $\underline{F}$. A random sample $\mathcal{D}$ of $\underline{F}$, called "the training set," is generated, and $\mathcal{D}$ is used in a "learning algorithm" to produce $\hat{\underline{F}}$, which is our estimate of $\underline{F}$. Clearly, this $\underline{F}$ and $\hat{\underline{F}}$ are statistically dependent, via $\mathcal{D}$. Indeed, intuitively speaking, the goal in designing a learning algorithm is that the $\hat{\underline{F}}$'s it produces are positively correlated with $\underline{F}$'s.

In practice this coupling is simply ignored in analyses of bias plus variance, without any justification (one such justification could be that the coupling has little effect on the value of the MSE). We shall follow that practice here. Accordingly, our equation for MSE reduces to

$$\mathrm{MSE}(\hat{\underline{F}}) = \int p(\hat{F})p(F)\,(\hat{F} - F)^2 \mathrm{d}\hat{F}\,\mathrm{d}F. \qquad (1)$$

If we were to account for the coupling of $\hat{F}$ and $\hat{\underline{F}}$ an additive correction term would need to be added to the right-hand side. For instance, see Wolpert (1997).

Using simple algebra, the right hand side of (1) can be written as the sum of three terms. The first is the variance of $\underline{F}$. Since this is beyond our control in designing the estimator $\hat{\underline{F}}$, we ignore it for the rest of this article. The second term involves a mean that describes the deterministic component of the error. This term depends on both the distribution of $\underline{F}$ and that of $\hat{\underline{F}}$, and quantifies how close the means of those distributions are. The third term is a variance that describes stochastic variations from one sample to the next. This term is independent of the random variable being estimated. Formally, up to an overall additive constant, we can write

$$\begin{aligned}
\mathrm{MSE}(\hat{F}) &= \int p(\hat{F})(\hat{F}^2 - 2F\hat{F} + F^2)\,\mathrm{d}\hat{F} \\
&= \overbrace{\int p(\hat{F})\hat{F}^2\,\mathrm{d}\hat{F} - 2F \int p(\hat{F})\hat{F}\,\mathrm{d}\hat{F}} + F^2 \\
&= \mathbb{V}(\hat{F}) + [\mathbb{E}(\hat{F})]^2 - 2F\,\mathbb{E}(\hat{F}) + F^2 \\
&= \mathbb{V}(\hat{F}) + \underbrace{[F - \mathbb{E}(\hat{F})]^2} \\
&= \text{variance} + \text{bias}^2. \qquad (2)
\end{aligned}$$

In light of (2), one way to try to reduce expected quadratic error is to modify an estimator to trade-off bias and variance. Some of the most famous applications of such bias-variance trade-offs occur in parametric machine learning, where many techniques have been developed to exploit the trade-off. Nonetheless, the trade-off also arises in many other fields, including integral estimation and optimization. In the rest of this paper we present a few novel applications of bias-variance trade-off, and describe some interesting features in each case. A recurring theme is the following: whenever a bias-variance trade-off arises in a particular field, we can use many techniques from parametric machine learning that have been developed for exploiting this trade-off. See Wolpert and Rajnarayan (2007) for further details of many of these applications.

Applications

In this section, we describe some applications of the bias-variance tradeoff. First, we describe Monte Carlo (MC) techniques for the estimation of integrals, and provide a brief analysis of bias-variance trade-offs in this context. Next, we introduce the field of Monte Carlo optimization (MCO), and illustrate that there are more subtleties involved than in simple MC. Then, we describe the field of parametric machine learning, which, as will show, is formally identical to MCO. Finally, we describe the application of parametric learning (PL) techniques to improve the performance of MCO algorithms. We do this in the context of an MCO problem that addresses black-box optimization.

Monte Carlo Estimation of Integrals Using Importance Sampling

Monte Carlo methods are often the method of choice for estimating difficult high-dimensional integrals. Consider a function $f: X \to \mathbb{R}$, which we want to integrate over some region $\mathcal{X} \subseteq X$, yielding the value F, as given by

$$F = \int_{\mathcal{X}} \mathrm{d}x f(x).$$

We can view this as a random variable $\underline{F}$, with density function given by a Dirac delta function centered on F. Therefore, the variance of $\underline{F}$ is 0, and (2) is exact.

A popular MC method to estimate this integral is importance sampling (see Robert & Casella, 2004). This exploits the law of large numbers as follows: i.i.d. samples $x^{(i)}, i = 1, \ldots, m$ are generated from a so-called importance distribution $h(x)$ that we control, and the associated values of the integrand, $f(x^{(i)})$ are computed. Denote these "data" by

$$\mathcal{D} = \{(x^{(i)}, f(x^{(i)}),\ i = 1, \ldots, m\}. \tag{3}$$

Now,

$$\begin{aligned} F &= \int_{\mathcal{X}} \mathrm{d}x\, h(x) \frac{f(x)}{h(x)} \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} \frac{f(x^{(i)})}{h(x^{(i)})} \quad \text{with probability 1.} \end{aligned}$$

Denote by $\underline{\hat{F}}$ the random variable with value given by the sample average for $\mathcal{D}$:

$$\hat{F} = \frac{1}{m} \sum_{i=1}^{m} \frac{f(x^{(i)})}{h(x^{(i)})}.$$

We use $\underline{\hat{F}}$ as our statistical estimator for $\underline{F}$, as we broadly described in the introductory section. Assuming a quadratic loss function, $L(\hat{F}, F) = (F - \hat{F})^2$, the bias-variance decomposition described in (2) applies *exactly*. It can be shown that the estimator $\underline{\hat{F}}$ is unbiased, that is, $\mathbb{E}(\underline{\hat{F}}) = F$, where the mean is over samples of h. Consequently, the MSE of this estimator is just its variance. The choice of sampling distribution h that minimizes this variance is given by (see Robert & Casella, 2004)

$$h^{\star}(x) = \frac{|f(x)|}{\int_{\mathcal{X}} |f(x')|\, \mathrm{d}x'}.$$

By itself, this result is not very helpful, since the equation for the optimal importance distribution contains a similar integral to the one we are trying to estimate. For non-negative integrands $f(x)$, the VEGAS algorithm (Lepage, 1978) describes an adaptive method to find successively better importance distributions, by iteratively estimating $\underline{F}$, and then using that estimate to generate the next importance distribution h. In the case of these unbiased estimators, there is no trade-off between bias and variance, and minimizing MSE is achieved by minimizing variance.

Monte Carlo Optimization

Instead of a *fixed* integral to evaluate, consider a parametrized integral

$$F(\theta) = \int_{\mathcal{X}} \mathrm{d}x f_{\theta}(x).$$

Further, suppose we are interested in finding the value of the parameter $\theta \in \Theta$ that minimizes $F(\theta)$:

$$\theta^{\star} = \arg\min_{\theta\in\Theta} F(\theta).$$

In the case where the functional form of f_{θ} is not explicitly known, one approach to solve this problem is a technique called MCO (see Ermoliev & Norkin, 1998), involving repeated MC estimation of the integral in question with adaptive modification of the parameter θ.

We proceed by analogy to the case with MC. First, we introduce the θ-indexed random variable $\underline{F}(\theta)$, all of whose components have delta-function distributions about the associated values $F(\theta)$. Next, we introduce a θ-indexed vector random variable $\underline{\hat{F}}$ with values

$$\hat{F} \equiv \{\hat{F}(\theta)\ \forall\ \theta \in \Theta\}. \tag{4}$$

Each real-valued component $\underline{\hat{F}}(\theta)$ can be sampled and viewed as an estimate of $\underline{F}(\theta)$.

For example, let $\mathcal{D}$ be a data set as described in (3). Then for every θ, any sample of $\mathcal{D}$ provides an associated estimate

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{f_{\theta}(x^{(i)})}{h(x^{(i)})}.$$

That average serves as an estimate of $\underline{F}(\theta)$. Formally, $\underline{\hat{F}}$ is a function of the random variable $\mathcal{D}$, and is given by such averaging over the elements of $\mathcal{D}$. So, a sample of $\mathcal{D}$ provides a sample of $\underline{\hat{F}}$. A priori, we make no restrictions on $\underline{\hat{F}}$, and so, in general, its components may be statistically coupled with one another. Note that this coupling arises even though we are, for simplicity, treating each function $\underline{F}(\theta)$ as having a delta-function distribution, rather than as having a non-zero variance that would reflect our lack of knowledge of the $f(\theta)$ functions.

However $\underline{\hat{F}}$ is defined, given a sample of $\hat{F}$, one way to estimate $\theta^\star$ is

$$\hat{\theta}^\star = \arg\min_{\theta \in \Theta} \hat{F}(\theta).$$

We call this approach "natural" MCO. As an example, say that $\mathcal{D}$ is a set of m samples of h, and let

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{f_\theta(x^{(i)})}{h(x^{(i)})},$$

as above. Under this choice for $\underline{\hat{F}}$,

$$\hat{\theta}^\star = \arg\min_{\theta \in \Theta} \frac{1}{m} \sum_{i=1}^{m} \frac{f_\theta(x^{(i)})}{h(x^{(i)})}. \tag{5}$$

We call this approach "naive" MCO.

Consider *any* algorithm that estimates $\theta^\star$ as a single-valued function of $\hat{F}$. The estimate of $\theta^\star$ produced by that algorithm is itself a random variable, since it is a function of the random variable $\underline{\hat{F}}$. Call this random variable $\underline{\hat{\theta}}^\star$, taking on values $\hat{\theta}^\star$. Any MCO algorithm is defined by $\underline{\hat{\theta}}^\star$; that random variable encapsulates the output estimate made by the algorithm.

To analyze the error of such an algorithm, consider the associated random variable given by the true parametrized integral $F(\underline{\hat{\theta}}^\star)$. The difference between a sample of $F(\underline{\hat{\theta}}^\star)$ and the true minimal value of the integral, $F(\theta^\star) = \min_\theta F(\theta)$, is the error introduced by our estimating that optimal θ as a sample of $\underline{\hat{\theta}}^\star$. Since our aim in MCO is to minimize $F(\theta)$, we adopt the loss function $L(\underline{\hat{\theta}}^\star, \theta^\star) \triangleq F(\underline{\hat{\theta}}^\star) - F(\theta^\star)$. This is in contrast to our discussion on MC integration, which involved quadratic loss. The current loss function just equals $F(\underline{\hat{\theta}}^\star)$ up to an additive constant $F(\theta^\star)$ that is fixed by the MCO problem at hand and is beyond our control. Up to that additive constant, the associated expected loss is

$$\mathbb{E}(L) = \int d\hat{\theta}^\star p(\hat{\theta}^\star) F(\hat{\theta}^\star). \tag{6}$$

Now change coordinates in this integral from the values of the scalar random variable $\underline{\hat{\theta}}^\star$ to the values of the underlying vector random variable $\underline{\hat{F}}$. The expected loss now becomes

$$\mathbb{E}(L) = \int \mathrm{d}\hat{F}\, p(\hat{F}) F(\hat{\theta}^\star(\hat{F})).$$

The natural MCO algorithm provides some insight into these results. For that algorithm,

$$\begin{aligned}\mathbb{E}(L) &= \int \mathrm{d}\hat{F}\, p(\hat{F}) F\left(\arg\min_\theta \hat{F}(\theta)\right) \\ &= \int \mathrm{d}\hat{F}(\theta_1)\, \mathrm{d}\hat{F}(\theta_2) \ldots\ p(\hat{F}(\theta_1), \hat{F}(\theta_2), \ldots) \\ &\quad F\left(\arg\min_\theta \hat{F}(\theta)\right).\end{aligned} \tag{7}$$

For any fixed θ, there is an error between samples of $\hat{F}(\theta)$ and the true value $F(\theta)$. Bias-variance considerations apply to this error, exactly as in the discussion of MC above. We are not, however, concerned with $\hat{F}$ for a single component θ, but rather for a set Θ of θ's.

The simplest such case is where the components of $\hat{F}(\Theta)$ are independent. Even so, $\arg\min_\theta \hat{F}(\theta)$ is distributed according to the laws for extrema of multiple independent random variables, and this distribution depends on higher-order moments of each random variable $\hat{F}(\theta)$. This means that $\mathbb{E}[L]$ also depends on such higher-order moments. Only the first two moments, however, arise in the bias and variance for any single θ. Thus, even in the simplest possible case, the bias-variance considerations for the individual θ do not provide a complete analysis.

In most cases, the components of $\hat{F}$ are *not* independent. Therefore, in order to analyze $\mathbb{E}[L]$, in addition to higher moments of the distribution for each θ, we must now also consider higher-order moments coupling the estimates $\hat{F}(\theta)$ for different θ.

Due to these effects, it may be quite acceptable for all the components $\hat{F}(\theta)$ to have both a large bias and a large variance, as long as they still order the θ's correctly with respect to the true $F(\theta)$. In such a situation, large covariances could ensure that if some $\hat{F}(\theta)$ were incorrectly large, then $\hat{F}(\theta'), \theta' \neq \theta$ would also be incorrectly large. This coupling between the components of $\hat{F}$ would preserve the ordering of θ's under $\underline{F}$. So, even with large bias and variance for each θ, the estimator as a whole would still work well.

Nevertheless, it *is* sufficient to design estimators $\hat{F}(\theta)$ with sufficiently small bias plus variance for each single θ. More precisely, suppose that those terms are very small on the scale of differences $F(\theta) - F(\theta')$ for any θ and θ'. Then by Chebychev's inequality,

we know that the density functions of the random variables $\hat{\underline{F}}(\theta)$ and $\hat{\underline{F}}(\theta')$ have almost no overlap. Accordingly, the probability that a sample of $\hat{\underline{F}}(\theta) - \hat{\underline{F}}(\theta')$ has the opposite sign of $F(\theta) - F(\theta')$ is almost zero.

Evidently, $\mathbb{E}[L]$ is generally determined by a complicated relationship involving bias, variance, covariance, and higher moments. Natural MCO in general, and naive MCO in particular, ignore all of these effects, and consequently, often perform quite poorly in practice. In the next section we discuss some ways of addressing this problem.

Parametric Machine Learning

There are many versions of the basic MCO problem described in the previous section. Some of the best-explored arise in parametric density estimation and parametric supervised learning, which together comprise the field of parametric machine learning (PL).

In particular, parametric supervised learning attempts to solve

$$\arg\min_{\theta\in\Theta} \int \mathrm{d}x\, p(x) \int \mathrm{d}y\, p(y \mid x) f_\theta(x).$$

Here, the values x represent inputs, and the values y represent corresponding outputs, generated according to some stochastic process defined by a set of conditional distributions $\{p(y \mid x),\ x \in \mathcal{X}\}$. Typically, one tries to solve this problem by casting it as an MCO problem. For instance, say we adopt a quadratic loss between a predictor $z_\theta(x)$ and the true value of y. Using MCO notation, we can express the associated supervised learning problem as finding $\arg\min_\theta F(\theta)$, where

$$\begin{aligned} l_\theta(x) &= \int \mathrm{d}y\, p(y \mid x)\, (z_\theta(x) - y)^2, \\ f_\theta(x) &= p(x)\, l_\theta(x), \\ F(\theta) &= \int \mathrm{d}x f_\theta(x). \end{aligned} \tag{8}$$

Next, the argmin is estimated by minimizing a sample-based estimate of the $F(\theta)$'s. More precisely, we are given a "training set" of samples of $p(y \mid x)\, p(x)$, $\{(x^{(i)}, y^i) i = 1, \ldots, m\}$. This training set provides a set of associated estimates of $F(\theta)$:

$$\hat{F}(\theta) = \frac{1}{m} \sum_{i=1}^{m} l_\theta(x^{(i)}).$$

These are used to estimate $\arg\min_\theta F(\theta)$, exactly as in MCO. In particular, one could estimate the minimizer of $F(\theta)$ by finding the minimum of $\hat{F}(\theta)$, just as in natural MCO. As mentioned above, this MCO algorithm can perform very poorly in practice. In PL, this poor performance is called "overfitting the data."

There are several formal approaches that have been explored in PL to try to address this "overfitting the data." Interestingly, none are based on direct consideration of the random variable $F(\hat{\theta}^\star(\hat{\underline{F}}))$ and the ramifications of its distribution for expected loss (cf. (7)). In particular, no work has applied the mathematics of extrema of multiple random variables to analyze the bias-variance-covariance trade-offs encapsulated in (7).

The PL approach that perhaps comes closest to such direct consideration of the distribution of $F(\hat{\underline{\theta}}^\star)$ is uniform convergence theory, which is a central part of computational learning theory (see Angluin, 1992). Uniform convergence theory starts by crudely encapsulating the quadratic loss formula for expected loss under natural MCO (7). It does this by considering the worst-case bound, over possible $p(x)$ and $p(y \mid x)$, of the probability that $F(\underline{\theta}^\star)$ exceeds $\min_\theta F(\theta)$ by more than κ. It then examines how that bound varies with κ. In particular, it relates such variation to characteristics of the set of functions $\{f_\theta : \theta \in \Theta\}$, e.g., the "VC dimension" of that set (see Vapnik, 1982, 1995).

Another, historically earlier approach, is to apply bias-plus-variance considerations to the *entire* PL algorithm $\hat{\underline{\theta}}^\star$, rather than to each $\hat{\underline{F}}(\theta)$ separately. This approach is applicable for algorithms that do not use natural MCO, and even for non-parametric supervised learning. As formulated for parameteric supervised learning, this approach combines the formulas in (8) to write

$$F(\theta) = \int \mathrm{d}x\, \mathrm{d}y\, p(x) p(y \mid x) (z_\theta(x) - y)^2.$$

This is then substituted into (6), giving

$$\begin{aligned} \mathbb{E}[L] &= \int \mathrm{d}\hat{\theta}^\star \mathrm{d}x\, \mathrm{d}y\, p(x)\, p(y \mid x)\, p(\hat{\theta}^\star) (z_{\hat{\theta}^\star}(x) - y)^2 \\ &= \int \mathrm{d}x\, p(x) \Big[\int \mathrm{d}\hat{\theta}^\star \mathrm{d}y\, p(x) p(y \mid x) p(\hat{\theta}^\star) \\ &\quad (z_{\hat{\theta}^\star}(x) - y)^2 \Big]. \end{aligned} \tag{9}$$

The term in square brackets is an x-parameterized expected quadratic loss, which can be decomposed into

a bias, variance, etc., in the usual way. This formulation eliminates any direct concern for issues like the distribution of extrema of multiple random variables, covariances between $\underline{\hat{F}}(\theta)$ and $\underline{\hat{F}}(\theta')$ for different values of θ, and so on.

There are numerous other approaches for addressing the problems of natural MCO that have been explored in PL. Particularly important among these are Bayesian approaches, e.g., Buntine and Weigend (1991), Berger (1985), and Mackay (2003). Based on these approaches, as well as on intuition, many powerful techniques for addressing data-overfitting have been explored in PL, including regularization, cross-validation, stacking, bagging, etc. Essentially all of these techniques can be applied to *any* MCO problem, not just PL problems. Since many of these techniques can be justified using (9), they provide a way to exploit the bias-variance trade-off in other domains besides PL.

PLMCO

In this section, we illustrate how PL techniques that exploit the bias-variance decomposition of (9) can be used to improve an MCO algorithm used in a domain outside of PL. This MCO algorithm is a version of adaptive importance sampling, somewhat similar to the CE method (Rubinstein & Kroese, 2004), and is related to function smoothing on continuous spaces. The PL techniques described are applicable to any other MCO problem, and this particular one is chosen just as an example.

MCO Problem Description The problem is to find the θ-parameterized distribution q_θ that minimizes the associated expected value of a function $G: \mathbb{R}^n \to \mathbb{R}$, i.e., find

$$\arg\min_\theta \mathbb{E}_{q_\theta}[G].$$

We are interested in versions of this problem where we do not know the functional form of G, but can obtain its value $G(x)$ at any $x \in \mathcal{X}$. Similarly we cannot assume that G is smooth, nor can we evaluate its derivatives directly. This scenario arises in many fields, including blackbox optimization (see Wolpert, Strauss, & Rajnarayan, 2006), and risk minimization (see Ermoliev & Norkin, 1998).

We begin by expressing this minimization problem as an MCO problem. We know that

$$\mathbb{E}_{q_\theta}[G] = \int_{\mathcal{X}} dx\, q_\theta(x) G(x)$$

Using MCO terminology, $f_\theta(x) = q_\theta(x)G(x)$ and $F(\theta) = \mathbb{E}_{q_\theta}[G]$. To apply MCO, we must define a vector-valued random variable $\underline{\hat{F}}$ with components indexed by θ, and then use a sample of $\underline{\hat{F}}$ to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$. In particular, to apply naive MCO to estimate $\arg\min_\theta \mathbb{E}_{q_\theta}(G)$, we first i.i.d. sample a density function $h(x)$. By evaluating the associated values of $G(x)$ we get a data set

$$\begin{aligned}\mathcal{D} &\equiv (\mathcal{D}_{\mathcal{X}}, \mathcal{D}_G) \\ &= (\{x^{(i)} : i = 1, \ldots, m\}, \{G(x^{(i)}) : i = 1, \ldots, m\}).\end{aligned}$$

The associated estimates of $F(\theta)$ for each θ are

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{q_\theta(x^{(i)}) G(x^{(i)})}{h(x^{(i)})}. \tag{10}$$

The associated naive MCO estimate of $\arg\min_\theta \mathbb{E}_{q_\theta}[G]$ is

$$\hat{\theta}^\star \equiv \arg\min_\theta \hat{F}(\theta).$$

Suppose Θ includes all possible density functions over x's. Then the q_θ minimizing our estimate is a delta function about the $x^{(i)} \in \mathcal{D}_{\mathcal{X}}$ with the lowest associated value of $G(x^{(i)})/h(x^{(i)})$. This is clearly a poor estimate in general; it suffers from "data-overfitting." Proceeding as in PL, one way to address this data-overfitting is to use regularization. In particular, we can use the entropic regularizer, given by the negative of the Shannon entropy $S(q_\theta)$. So we now want to find the minimizer of $\mathbb{E}_{q_\theta}[G(x)] - TS(q_\theta)$, where T is the regularization parameter. Equivalently, we can minimize $\beta \mathbb{E}_{q_\theta}[G(x)] - S(q_\theta)$, where $\beta = 1/T$. This changes the definition of $\hat{F}$ from the function given in (10) to

$$\hat{F}(\theta) \triangleq \frac{1}{m} \sum_{i=1}^{m} \frac{\beta\, q_\theta(x^{(i)}) G(x^{(i)})}{h(x^{(i)})} - S(q_\theta).$$

Solution Methodology Unfortunately, it can be difficult to find the θ globally minimizing this new $\hat{F}$ for an arbitrary $\mathcal{D}$. An alternative is to find a close approximation

to that optimal θ. One way to do this is as follows. First, we find minimizer of

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\beta\, p(x^{(i)})G(x^{(i)})}{h(x^{(i)})} - S(p) \tag{11}$$

over the set of *all* possible distributions $p(x)$ with domain $\mathcal{X}$. We then find the q_θ that has minimal Kullback–Leibler (KL) divergence from this p, evaluated over $\mathcal{D}_\mathcal{X}$. That serves as our approximation to $\arg\min_\theta \hat{F}(\theta)$, and therefore as our estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$.

The minimizer p of (11) can be found in closed form; over $\mathcal{D}_\mathcal{X}$ it is the Boltzmann distribution $p^\beta(x^{(i)}) \propto \exp(-\beta\, G(x^{(i)}))$. The KL divergence in $\mathcal{D}_\mathcal{X}$ from this Boltzmann distribution to q_θ is

$$F(\theta) = \mathrm{KL}(p^\beta \| q_\theta) = \int_\mathcal{X} \mathrm{d}x\, p^\beta(x) \log\left(\frac{p^\beta(x)}{q_\theta(x)}\right).$$

The minimizer of this KL divergence is given by

$$\theta^\dagger = \arg\min_\theta -\sum_{i=1}^{m}\frac{\exp(-\beta G(x^{(i)}))}{h(x^{(i)})}\log(q_\theta(x^{(i)})). \tag{12}$$

This approach is an approximation to a regularized version of the naive MCO estimate of the θ that minimizes $\mathbb{E}_{q_\theta}(G)$. The application of the technique of regularization in this context has the same motivation as it does in PL: to reduce bias plus variance.

Log-Concave Densities If q_θ is log-concave in its parameters θ, then the minimization problem in (12) is a convex optimization problem, and the optimal parameters can be found closed-form. Denote the likelihood ratios by $s^{(i)} = \exp(-\beta G(x^{(i)}))/h(x^{(i)})$. Differentiating (12) with respect to the parameters μ and Σ^{-1} and setting them to zero yields

$$\mu^\star = \frac{\sum_\mathcal{D} s^{(i)} x^{(i)}}{\sum_\mathcal{D} s^{(i)}}$$
$$\Sigma^\star = \frac{\sum_\mathcal{D} s^{(i)}(x^{(i)} - \mu^\star)(x^{(i)} - \mu^\star)^T}{\sum_\mathcal{D} s^{(i)}}$$

Mixture Models The single Gaussian is a fairly restrictive class of models. Mixture models (see ▶Mixture Modeling) can significantly improve flexibility, but at the cost of convexity of the KL distance minimization problem. However, a plethora of techniques from supervised learning, in particular the expectation maximization (EM) algorithm, can be applied with minor modifications.

Suppose q_θ is a mixture of M Gaussians, that is, $\theta = (\mu, \Sigma, \phi)$ where ϕ is the mixing p.m.f, we can view the problem as one where a hidden variable z decides which mixture component each sample is drawn from. We then have the optimization problem

$$\text{minimize } -\sum_\mathcal{D}\frac{p(x^{(i)})}{h(x^{(i)})}\log\left(q_\theta(x^{(i)}, z^{(i)})\right).$$

Following the standard EM procedure, we get the algorithm described in (13). Since this is a nonconvex problem, one typically runs the algorithm multiple times with random initializations of the parameters.

E-step: For each i, set $Q_i(z^{(i)}) = p(z^{(i)}|x^{(i)})$,
that is, $w_j^{(i)} = q_{\mu,\Sigma,\phi}(z^{(i)} = j|x^{(i)}),\quad j = 1,\ldots,M.$

M-step: Set
$$\mu_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}\, x^{(i)}}{\sum_\mathcal{D} w_j^{(i)} s^{(i)}},$$
$$\Sigma_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}\,(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_\mathcal{D} w_j^{(i)} s^{(i)}},$$
$$\phi_j = \frac{\sum_\mathcal{D} w_j^{(i)} s^{(i)}}{\sum_\mathcal{D} s^{(i)}}.$$

Test Problems To compare the performance of this algorithm with and without the use of PL techniques, we use a couple of very simple academic problems in two and four dimensions – the Rosenbrock function in two dimensions, given by

$$G_R(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

and the Woods function in four dimensions, given by given by

$$\begin{aligned} G_{\text{Woods}}(x) = {} & 100(x_2 - x_1)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 \\ & + (1 - x_3)^2 \\ & + 10.1[(1 - x_2)^2 + (1 - x_4)^2] \\ & + 19.8(1 - x_2)(1 - x_4). \end{aligned}$$

For the Rosenbrock, the optimum value of 0 is achieved at $x = (1,1)$, and for the Woods problem, the optimum value of 0 is achieved at $x = (1,1,1,1)$.

Application of PL Techniques As mentioned above, there are many PL techniques beyond regularization that are designed to optimize the trade-off between bias and variance. So having cast the solution of $\arg\min_{q_\theta} \mathbb{E}(G)$ as an MCO problem, we can apply those other PL techniques instead of (or in addition to) entropic regularization. This should improve the performance of our MCO algorithm, for the exact same reason that using those techniques to trade off bias and variance improves performance in PL. We briefly mention some of those alternative techniques here.

The overall MCO algorithm is broadly described in Algorithm 1. For the Woods problem, 20 samples of x are drawn from the updated q_θ at each iteration, and for the Rosenbrock, 10 samples. For comparing various methods and plotting purposes, 1,000 samples of $G(x)$ are drawn to evaluate $\mathbb{E}_{q_\theta}[G(x)]$. Note: in an actual optimization, we will not be drawing these test samples! All the performance results in Fig. 1 are based on 50 runs of the PC algorithm, randomly initialized each time. The sample mean performance across these runs is plotted along with 95% confidence intervals for this sample mean (shaded regions).

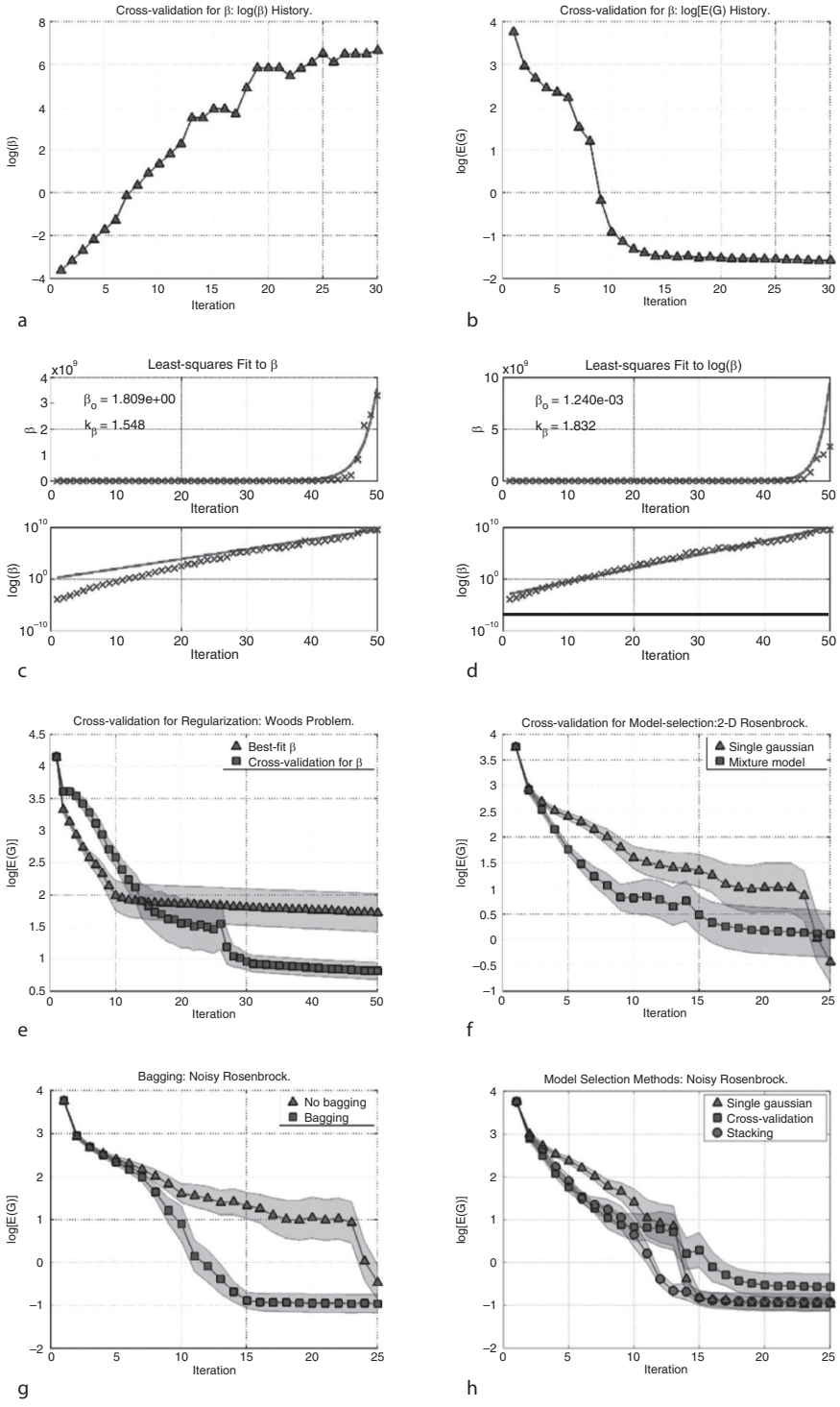
►*Cross-Validation for Regularization:* We note that we are using regularization to reduce variance, but that regularization introduces bias. As is done in PL, we use standard k-fold cross-validation to tradeoff this bias and variance. We do this by partitioning the data into k disjoint sets. The held-out data for the ith fold is just the ith partition, and the held-in data is the union of all other partitions. First, we "train" the regularized algorithm on the held-in data $\mathcal{D}_t$ to get an optimal set of parameters $\theta^\star$, then "test" this $\theta^\star$ by considering unregularized performance on the held-out data $\mathcal{D}_v$. In our context, "training" refers to finding optimal parameters by KL distance minimization using the held-in data, and "testing" refers to estimating $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data using the following formula (Robert & Casella, 2004).

$$\widehat{g}(\theta) = \frac{\sum_{\mathcal{D}_v} \frac{q_\theta(x^{(i)})G(x^{(i)})}{h(x^{(i)})}}{\sum_{\mathcal{D}_v} \frac{q_\theta(x^{(i)})}{h(x^{(i)})}}.$$

We do this for several values of the regularization parameter β in the interval $k_1\beta < \beta < k_2\beta$, and choose the one that yield the best held-out performance, averaged over all folds. For our experiments, $k_1 = 0.5, k_2 = 3$, and we use five equally-spaced values in this interval. Having found the best regularization parameter in this range, we then use *all* the data to minimize KL distance using this optimal value of β. Note that all cross-validation is done *without* any additional evaluations of $G(x)$. Cross-validation for β in PC is similar to optimizing the annealing schedule in simulated annealing. This "auto-annealing" is seen in Fig. 1a, which shows the variation of β with iterations of the Rosenbrock problem. It can be seen that β value sometimes decreases from one iteration to the next. This can never happen in any kind of "geometric annealing schedule," $\beta \leftarrow k_\beta\beta$, $k_\beta > 1$, of the sort that is often used in most algorithms in the literature. In fact, we ran 50 trials of this algorithm on the Rosenbrock and then computed a best-fit geometric variation for β, that is, a nonlinear least squares fit to variation of β, and a linear least squares fit to the variation of $\log(\beta)$. These are shown in Fig. 1c and d. As can be seen, neither is a very good fit. We then ran 50 trials of the algorithm with the fixed update rule obtained by best-fit to $\log(\beta)$, and found that the adaptive setting of β using cross-validation performed an order of magnitude better, as shown in Fig. 1e.

Algorithm 1 Overview of pq minimization using Gaussian mixtures

1: Draw uniform random samples on X
2: Initialize regularization parameter β
3: Compute $G(x)$ values for those samples
4: **repeat**
5: Find a mixture distribution q_θ to minimize sampled pq KL distance
6: Sample from q_θ
7: Compute $G(x)$ for those samples
8: Update β
9: **until** Termination
10: Sample final q_θ to get solution(s).



Bias-Variance Trade-offs: Novel Applications. Figure 1. Various PL techniques improve MCO performance

Cross-Validation for Model Selection: Given a set Θ (sometimes called a model class) to choose θ from, we can find an optimal $\theta \in \Theta$. But how do we choose the set Θ? In PL, this is done using cross-validation. We choose that set Θ such that $\arg\min_{\theta\in\Theta} \hat{F}(\theta)$ has the best held-out performance. As before, we use that model class Θ that yields the lowest estimate of $\mathbb{E}_{q_\theta}[G(x)]$ on the held-out data. We demonstrate the use of this PL technique for minimizing the Rosenbrock problem, which has a long curved valley that is poorly approximated by a single Gaussian. We use cross-validation to choose between a Gaussian mixture with up to four components. The improvement in performance is shown in Fig. 1d.

Bagging: In bagging (Breiman, 1996a), we generate multiple data sets by resampling the given data set with replacement. These new data sets will, in general, contain replicates. We "train" the learning algorithm on each of these resampled data sets, and average the results. In our case, we average the q_θ got by our KL divergence minimization on each data set. PC works even on stochastic objective functions, and on the noisy Rosenbrock, we implemented PC with bagging by resampling ten times, and obtained significant performance gains, as seen in Fig. 1g.

Stacking: In bagging, we combine estimates of the same learning algorithm on different data sets generated by resampling, whereas in stacking (Breiman, 1996b; Smyth & Wolpert, 1999), we combine estimates of different learning algorithms on the same data set. These combined estimated are often better than any of the single estimates. In our case, we combine the q_θ obtained from our KL divergence minimization algorithm using multiple models Θ. Again, Fig. 1h shows that cross-validation for model selection performs better than a single model, and stacking performs slightly better than cross-validation.

Conclusions

The conventional goal of reducing bias plus variance has interesting applications in a variety of fields. In straightforward applications, the bias-variance trade-offs can decrease the MSE of estimators, reduce the generalization error of learning algorithms, and so on. In this article, we described a novel application of bias-variance trade-offs: we placed bias-variance trade-offs in the context of MCO, and discussed the need for higher moments in the trade-off, such as a bias-variance-covariance trade-off. We also showed a way of applying just a bias-variance trade-off, as used in Parametric Learning, to improve the performance of MCO algorithms.

Recommended Reading

Angluin, D. (1992). Computational learning theory: Survey and selected bibliography. In *Proceedings of the twenty-fourth annual ACM symposium on theory of computing*. New York: ACM.

Berger, J. O. (1985). *Statistical decision theory and bayesian analysis*. New York: Springer.

Breiman, L. (1996a). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (1996b). Stacked regression. *Machine Learning, 24*(1), 49–64.

Buntine, W., & Weigend, A. (1991). Bayesian back-propagation. *Complex Systems, 5*, 603–643.

Ermoliev, Y. M., & Norkin, V. I. (1998). *Monte carlo optimization and path dependent nonstationary laws of large numbers*. Technical Report IR-98-009. International Institute for Applied Systems Analysis, Austria.

Lepage, G. P. (1978). A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics, 27*, 192–203.

Mackay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge, UK: Cambridge University Press.

Robert, C. P., & Casella, G. (2004). *Monte Carlo statistical methods*. New York: Springer.

Rubinstein, R., & Kroese, D. (2004). *The cross-entropy method*. New York: Springer.

Smyth, P., & Wolpert, D. (1999). Linearly combining density estimators via stacking. *Machine Learning, 36*(1–2), 59–83.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data*. New York: Springer.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. New York: Springer.

Wolpert, D. H. (1997). On bias plus variance. *Neural Computation, 9*, 1211–1244.

Wolpert, D. H., & Rajnarayan, D. (2007). Parametric learning and monte carlo optimization. arXiv:0704.1274v1 [cs.LG].

Wolpert, D. H., Strauss, C. E. M., & Rajnarayan, D. (2006). Advances in distributed optimization using probability collectives. *Advances in Complex Systems, 9*(4), 383–436.

Bias-Variance Trade-offs

►Bias-Variance

Bias-Variance-Covariance Decomposition

The bias-variance-covarianc delcomposition is a theoretical result underlying ►ensemble learning algorithms. It is an extension of the ►bias-variance decomposition, for linear combinations of models. The expected squared error of the ensemble $\bar{f}(\mathbf{x})$ from a target d is:

$$\mathcal{E}_{\mathcal{D}}\{(\bar{f}(\mathbf{x}) - d)^2\} = \overline{\text{bias}}^2 + \frac{1}{T}\overline{\text{var}} + \left(1 - \frac{1}{T}\right)\overline{\text{covar}}.$$

The error is composed of the average bias of the models, plus a term involving their average variance, and a final term involving their average *pairwise covariance*. This shows that while a single model has a two-way bias-variance tradeoff, an ensemble is controlled by a three-way tradeoff. This ensemble tradeoff is often referred to as the *accuracy-diversity* dilemma for an ensemble. See ►ensemble learning for more details.

Bilingual Lexicon Extraction

Bilingual lexicon extraction is the task of automatically identifying a terms in a first language and terms in a second language which are translation f one another. In this context, a term can be either a single word or an expression composed of several words the full meaning of which cannot be derived compositionally from the meaning of the individual words. Bilingual lexicon extraction is itself a form of ►cross-lingual text mining and is an essential preliminary step in many approaches for performing other ►cross-lingual text mining tasks.

Binning

►Discretization

Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity

Wulfram Gerstner
Brain Mind Institute, Lausanne EPFL, Switzerland

Synonyms

Correlation-based learning; Hebb rule; Hebbian learning

Definition

The brain of humans and animals consists of a large number of interconnected neurons. Learning in biological neural systems is thought to take place by changes in the connections between these neurons. Since the contact points between two neurons are called synapses, the change in the connection strength is called synaptic plasticity. The mathematical description of synaptic plasticity is called a (biological) learning rule. Most of these biological learning rules can be categorized in the context of machine learning as unsupervised learning rules, and the remaining ones as reward-based or reinforcement learning. The Hebb rule is an example of an unsupervised correlation-based learning rule formulated on the level of neuronal firing rates. Spike-timing-dependent plasticity (STDP) is an unsupervised learning rule formulated on the level of spikes. Modulation of learning rates in a Hebb rule or STDP rule by a diffusive signal carrying reward-related information yields a biologically plausible form of a reinforcement learning rule.

Motivation and Background

Humans and animals can adapt to environmental conditions and learn new tasks. Learning becomes measurable by changes in the behavior: humans and animals get better at seeing and distinguishing visual objects with experience; animals can learn to go to a target location; humans can memorize a list of words and recall the items 2 days later. How learning is implemented in the biological substrate is only partially known.

The brain consists of billions of neurons. Each neuron has long wire-like extensions and makes contacts with thousands of other neurons. This network of neurons is not fixed but constantly changes. Connections

can be formed or can disappear, and existing connections can be strengthened or weakened. Neuroscientists have shown in numerous experiments that changes can be induced by stimulating neuronal activity in an appropriate fashion. Moreover, changes in synaptic connections that have been induced in one or a few seconds can persist for hours or days, an effect called long-term potentiation (LTP) or long-term depression (LTD) of synapses.

The question arises of whether such long-lasting changes in connections are useful for learning. To answer this question, research in theoretical and computational neuroscience needs to solve two problems: First, develop a compact but realistic description of the phenomenon of synaptic plasticity observed in biology, i.e., extract learning rules from the biological data; and second, study the functional consequences of these learning rules. An important insight from experiments on LTP is that the activation of a synaptic connection alone does not lead to a long-lasting change; however, if the activation of the synapses by presynaptic signals is combined with some activation of the postsynaptic neuron, then a long-lasting change of the synapse may occur. The coactivation of presynaptic and postsynaptic neurons as a condition for learning is the key ingredient of Hebbian learning rules. Here, activation of the presynaptic neuron means that it fires one or several action potentials; activation of the postsynaptic neuron can be represented by high firing rates, a few well-timed action potentials or input from other neurons that lead to an increase in the membrane voltage.

Structure of the Learning System

The Hebb Rule

Hebbian learning rules are local, i.e., they depend only on the state of the presynaptic and postsynaptic neurons plus possibly the current value of the synaptic weight itself. Let w_{ij} denotes the weight between a presynaptic neuron j and a postsynaptic neuron i, and let us describe the activity (e.g., the firing rate) of each neuron by a continuous variable v_j and v_i, respectively. Mathematically, we may therefore write for a local learning rule

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}; v_i, v_j) \tag{1}$$

where F is an unknown function. In addition to locality, Hebbian learning requires some kind of cooperation or correlation between the activity of the presynaptic neuron and that of the postsynaptic neuron. At the moment we restrict ourselves to the requirement of *simultaneous activity* of presynaptic and postsynaptic neurons. Since F is a function of the rates v_i and v_j, we may expand F about $v_i = v_j = 0$. An expansion to second order of the rates yields

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) \approx\ & c_0(w_{ij}) + c_1^{\mathrm{pre}}(w_{ij})\, v_j + c_1^{\mathrm{post}}(w_{ij}) v_i \\ & + c_2^{\mathrm{corr}}(w_{ij})\, v_i\, v_j + c_2^{\mathrm{post}}(w_{ij})\, v_i^2 \\ & + c_2^{\mathrm{pre}}(w_{ij})\, v_j^2 + O(v^3).\end{aligned} \tag{2}$$

Here, v_i and v_j are functions of time, i.e., $v_i(t)$ and $v_j(t)$ and so is the weight w_{ij}. The bilinear term $v_i(t)\, v_j(t)$ is sensitive to the instantaneous *correlations* between presynaptic and postsynaptic activities. It is this term that makes Hebbian learning a useful concept. The simplest implementation of Hebbian plasticity would be to require $c_2^{\mathrm{corr}} > 0$ and set all other parameters in the expansion (2) to zero

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_2^{\mathrm{corr}}(w_{ij})\, v_i\, v_j\,. \tag{3}$$

Equation (3) with fixed parameter $c_2^{\mathrm{corr}} > 0$ is the prototype of Hebbian learning. However, since the activity variables v_i and v_j are always positive, such a rule will lead eventually to an increase of all weights in a network. Hence, some of the other terms (e.g., c_0 or c_1^{pre}) need to have a negative coefficient to make Hebbian learning stable. In passing we note that a learning rule with $c_2^{\mathrm{corr}} < 0$ is usually called anti-Hebbian.

Oja's rule. A particular interesting case is a model with coefficients $c_2^{\mathrm{corr}} > 0$ and $c_2^{\mathrm{post}} < 0$, since it guarantees the normalization of the set of weights $w_{i1}, \ldots w_{iN}$ converging onto the same postsynaptic neuron i.

BCM rule. The Bienenstock–Cooper–Munro learning rule (also called BCM rule) with

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = a(w_{ij})\Phi(v_i - \vartheta)\, v_j \tag{4}$$

where Φ is some nonlinear function with $\Phi(0) = 0$ is a special case of (1). The parameter ϑ depends on the average firing rate.

Temporally asymmetric Hebbian learning. In the Taylor expansion (2) we focused on *instantaneous* correlations. More generally, we can use a Volterra expansion so as to also include temporal correlations with

nonzero time lag. With the additional assumptions that changes are instantaneous, a Volterra expansion generates terms of the form

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} \propto \int_0^\infty [W_+(s) v_i(t)\, v_j(t-s) + W_-(s) v_j(t)\, v_i(t-s)]\mathrm{d}s \qquad (5)$$

with some functions W_+ and W_-. For reasons of causality, W_+ and W_- must vanish for $s < 0$. Since $W_+(s) \neq W_-(s)$, learning is asymmetric in time so that learning rules of the form (5) are called temporally asymmetric Hebbian learning. In the special case $W_+(s) = -W_-(s)$, we have antisymmetric Hebbian learning. The functions W_+ and W_- may depend on the present weight value.

STDP rule. STDP is a form of Hebbian learning with increased temporal resolution. In contrast to rate-based Hebb models, neuronal activity is described by the firing times of the neuron, i.e., the moments when the presynaptic and postsynaptic neurons emit action potentials. Let t_j^f denote the fth spike of the presynaptic neuron j and t_i^n the nth spike of the postsynaptic neuron i. The weight change in an STDP rule depends on the exact timing of presynaptic and postsynaptic spikes

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = \sum_n \sum_f [A(w_{ij}; t - t_j^f)\delta(t - t_i^n) + B(w_{ij}; t - t_i^f)\delta(t - t_j^f)] \qquad (6)$$

where $A(x)$ and $B(x)$ are some real-valued functions with $A(w_{ij}, x) = B(w_{ij}, x) = 0$ for $x < 0$. Thus, at the moment of a postsynaptic spike the synaptic weight is updated by an amount that depends on the time $t_i^f - t_j^f$ since a previous presynaptic spike t_j^f. Similarly, at the moment of a presynaptic spike the synaptic weight is updated by an amount that depends on the time $t_j^f - t_i^f$ since a previous postsynaptic spike t_i^f. The dependence on the present value w_{ij} can be used to keep the weight in a desired range $0 < w_{ij} < w^{\max}$. A standard choice for the functions A and B is $A(w_{ij}); t - t_j^f = A_+(w_{ij}) \exp[-(t - t_j^f)/\tau_+]$ for $t - t_j^f > 0$ and zero otherwise. Similarly, $B(w_{ij}; t - t_i^n) = B_-(w_{ij}) \exp[-(t - t_i^n)/\tau_-]$ for $t - t_i^f > 0$ and zero otherwise. Here, τ_+ and τ_- are time constants in the range of 10–50 ms. The case $A_+(x) = (w^{\max} - x)\, c_+$ and $B_x(x) = -\, c_- x$ is called soft bounds. The choice $A_+(x) = c_+ \Theta(w^{\max} - x)$ and $B_x = -\, c_- \Theta(x)$ is called hard bounds. Here, c_+ and c_- are positive constants. The term proportional to A_+ causes potentiation (weight increase), the one proportional to A_- causes depression (weight decrease) of synapses. Note that the STDP rule (6) can be interpreted as a spike-based form of temporally asymmetric Hebbian learning.

Functional Consequences of Hebbian Learning

Sensitivity to correlations. All Hebbian learning rules are sensitive to the correlations between the activity of the presynaptic neuron j and that of the postsynaptic neuron i. If the activity of the postsynaptic neuron is given by a linear sum of all inputs rates, i.e., $v_i = \gamma \sum_j w_{ij} v_j$, then correlations between presynaptic and postsynaptic activities can be traced back to correlations in the input. A particular clear example of learning driven by correlations in the input is Oja's learning rule applied to a statistical ensemble of inputs with zero mean. In this case, the postsynaptic neuron becomes sensitive to the dominant principal component of the input ensemble. If the neuron model is nonlinear, Hebbian learning extracts the independent components of the statistical input ensemble. These two examples show that learning by a Hebbian learning rule makes neurons adapt to the statistics of the input. While the condition of zero-mean input is biologically not realistic (because neuronal firing rates are always positive), this condition can be relaxed so that the same result is also applicable to biologically plausible learning rules.

Receptive fields and cortical maps. Neurons in the primary visual cortex of cats and monkeys respond to visual stimuli in a localized region of the visual field. This small sensitive zone is called the receptive field of the neuron. Neighboring neurons normally have very similar receptive fields. The exact location and properties of the receptive field are not fixed, but can be influenced by sensory stimulation. Models of unsupervised Hebbian learning can explain the development of receptive fields and the adaptation of cortical maps to the statistics of the ensemble of stimuli.

Beyond the Hebb rule. Standard models of Hebbian learning are formulated on the level of neuronal firing rates, a graded variable characterizing neuronal activity. However, real neurons communicate by spikes, short electrical pulses or "action potentials" with a rather

stereotyped time course. Experiments have shown that the changes of synaptic efficacy depend not only on the mean firing rate of action potentials but on the relative timing of presynaptic and postsynaptic spikes on the level of milliseconds. This Spike-Timing Dependent Synaptic Plasticity (STDP) can be considered a temporally more precise form of Hebbian learning. The STDP rule indicated above supposes that pairs of spikes (one presynaptic and one postsynaptic action potential) within some time window cause a weight change. However, experimentally it was shown that at least three spikes are necessary (one presynaptic and two postsynaptic spikes). Moreover, the voltage of the postsynaptic neuron matters even in the absence of spikes.

In most models of Hebbian learning and STDP, the factors $c_0, c_1^{\text{pre}} \ldots$ are constant or depend only on the synaptic weight. However, in biological context the speed of learning is often gated by neuromodulators. Since some of these neuromodulators contain reward-related information, one can think of learning as a three-factor rule where weight changes depend on presynaptic activity, postsynaptic activity, and the presence of a reward-related factor. A prominent neuromodulator linked to reward information is dopamine. Three factor learning rules fall in the class of reinforcement learning algorithms.

Cross References

►Dimensionality Reduction
►Reinforcement Learning
►Self-Organizing Maps

Recommended Reading

Bliss, T., & Gardner-Medwin, A. (1973). Long-lasting potentation of synaptic transmission in the dendate area of unanaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology, 232*, 357–374.

Bliss, T., Collingridge, G., & Morris, R. (2003). Long-term potentiation: Enhancing neuroscience for 30 years - introduction. *Philosophical Transactions of the Royal Society of London. Series B : Biological Sciences, 358*, 607–611.

Cooper, L., Intrator, N., Blais, B., & Shouval, H. Z. (2004). *Theory of cortical plasticity.* Singapore: World Scientific.

Dayan, P., & Abbott, L. F. (2001). *Theoretical Neuroscience*. Cambridge, MA: MIT Press.

Gerstner, W., & Kistler, W. K. (2002). *Spiking neuron models*. Cambridgess, UK: Cambridge University Press.

Gerstner, W., Kempter, R., van Hemmen, J. L., & Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature, 383*, 76–78.

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

Lisman, J. (2003). Long-term potentiation: Outstanding questions and attempted synthesis. *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences, 358*, 829–842.

Malenka, R. C., & Nicoll, R. A. (1999). Long-term potentiation–a decade of progress? *Science, 285*, 1870–1874.

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postysnaptic AP and EPSP. *Science, 275*, 213–215.

Schultz, W., Dayan, P., & Montague, R. (1997). A neural substrate for prediction and reward. *Science, 275*, 1593–1599.

Biomedical Informatics

C. David Page, Sriraam Natarajan
University of Wisconsin Medical School, Madison, USA

Introduction

Recent years have witnessed a tremendous increase in the use of machine learning for biomedical applications. This surge in interest has several causes. One is the successful application of machine learning technologies in other fields such as web search, speech and handwriting recognition, agent design, spatial modeling, etc. Another is the development of technologies that enable the production of large amounts of data in the time it used to take to generate a single data point (run a single experiment). A third most recent development is the advent of Electronic Medical/Health Records (EMRs/EHRs). The drastic increase in the amount of data generated has led the biologists and clinical researchers to adopt algorithms that can construct predictive models from large amounts of data. Naturally, machine learning is emerging as a tool of choice.

In this article, we will present a few data types and tasks involving such large-scale biological data, where machine learning techniques have been applied. For each of these data types and tasks, we first present the required background, followed by the challenges involved in addressing the tasks. Then, we present the machine learning techniques that have been applied to these data sets. Finally and most importantly, we

present the lessons learned in these tasks. We hope that these lessons will be helpful to researchers who aim to apply machine learning algorithms to biological applications and equip them with useful knowledge when they collaborate with biological scientists.

Some of the data types that we present in this work are:

- Gene expression microarrays
- SNPs and genetic data
- Mass spectrometry and other proteomic data
- High-throughput screening data for drug design
- Electronic Medical Records (EMR) and personalized medicine

Some of the key lessons learned from all these data types include the following: (1) We can often do surprisingly well with far more features than data points if there are many highly predictive features (e.g., predicting cancer from microarray data) and if we use methods that are robust to overfitting such as *Voted Decision Stumps* (Hardin et al., 2004; Waddell et al., 2005) (►Ensemble Learning and ►Decision Stumps), ►Naive Bayes (Golub et al., 1999; Listgarten et al., 2004), or *Linear Support Vector Machines (SVMs)* (see ►Support Vector Machine) (Furey et al., 2000; Hardin et al., 2004). (2) Bayes Net learning (Friedman, 2000) (see ►Bayesian Methods) often does not give us causality, but ►Active Learning and ►Time-Series data help if available (Pe'er, Regev, Elidan, & Friedman, 2001; Ong, Glassner, & Page, 2002; Tucker, Vinciotti, Hoen, Liu, & Famili, 2005; Zou & Conzen, 2005). (3) Multi-relational methods are useful for EMRs or molecular data as the data in these cases are very highly relational (see ►Multi-relational Data Mining). (4) There are more important issues than just increasing the accuracy of the learned model on these data sets. Such issues include how data was created, its comprehensibility (physicians typically want to understand the model that has been learned), and its privacy (some data sets contain private information that cannot be posted on public web sites and cannot even be downloaded off site).

The rest of the paper is organized as follows: First we present gene expression microarrays, followed by SNPs and other genetic data. We then present mass spectrometry (MS) and related proteomic data. Next, we present high-throughput screening data for drug design, followed by EMR data and personalized medicine. For each of these data types, we motivate the problem and survey the different machine learning solutions. Finally, we conclude by outlining the lessons learned from all these data types and presenting some interesting and exciting directions for future research.

Gene Expression Microarrays

This data type was presented in detail in *AI Magazine* (Molla et al., 2004) and hence we will brief it in this section. We encourage the reader to read Molla et al. (2004) for more details on this data type. Genes are contained in the DNA of an organism. The mechanism by which proteins are produced from their corresponding genes is a two-step process. The first step is the *transcription* of a gene into a messenger RNA (mRNA) and in the second step called as *translation*, a protein is built using mRNA as a blueprint.

One property that DNA and RNA have in common is that each is a chain of chemicals called as *bases*. In the case of DNA, these bases are *Adenine*, *Cytosine*, *Guanine*, and *Thymine*, commonly referred to as A, C, G, and T, respectively. RNA has the same set of four bases, except Thymine; RNA has *Uracil*, commonly referred as U. An important characteristic of DNA and RNA is *complementarity*, that is, each base only binds well with its complement: A with T (or U) and G with C. As a result of complementarity, a strand of either DNA or RNA has a strong affinity toward what is known as its *reverse complement*, which is a strand of either DNA or RNA that has bases exactly complementary to the original strand. Complementarity is central to the processes of replication of the DNA and transcription.

In addition, complementarity can be used to detect specific sequences of bases within strands of DNA and RNA. This is done by first synthesizing a *probe*, a piece of DNA that is the complement of a sequence that one wants to detect, and then introducing this probe to a solution containing the genetic material (DNA or RNA) to be searched. This solution of genetic material is called the *sample*. In theory, the probe will bind to the sample if and only if the probe finds its complement in the sample (in reality, this process is often imperfect). The act of binding between a sample and probe is called *hybridization*. Prior to the experiment, a biologist labels the probe using a florescent flag. After the

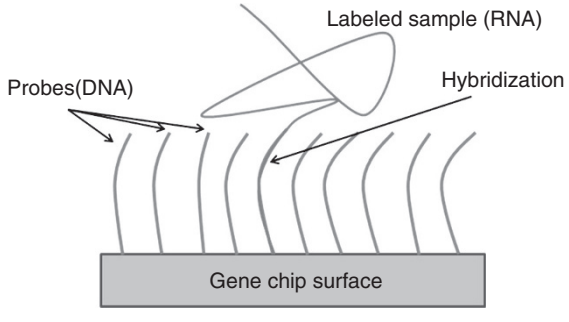
hybridization experiment, one can easily scan to see if the probe has hybridized to its reverse complement in the sample. This allows the molecular biologist to determine the presence or absence of the sequence in the sample.

Gene Chips

DNA probe technology has been adapted for detection of tens of thousands of sequences simultaneously. This has become possible due to the device called a *microarray* or *gene chip*, the working of which is illustrated in Fig. 1. When using the chips it is more common to label (luminescently) the samples than the probe. Thousands of copies of this labeled sample are spread across the probe, followed by washing away any copies that do not remain bound. Since the probes are attached at specific locations on the chip, if a labeled sample is detected at any position in the chip, the probe that is hybridized to its complement can be easily determined. The most common use of these gene chips is to measure the expression levels of various genes in the organism.

Probes are typically on the order of 25-bases long, whereas samples are usually about 10 times, as long, with a large variation due to the process that breaks up long sequences of RNA into small samples (Molla et al., 2004).

To understand about the biology of an organism, say to understand human biology to design new drugs or lower the blood pressure or to cure diabetes, there is a necessity to understand the degree to which different genes get expressed as proteins under different conditions and different cell types. It is much easier to estimate the amount of mRNA for a gene than the protein-production rate. Microarrays provide the measurement of RNAs corresponding to the given gene rather than the amounts of protein.



Biomedical Informatics. Figure 1. Hybridization of sample to probe

In brief, experiments with the microarrays are performed as follows: As can be seen from the figure, probes are DNA strands attached to the gene chip surface. A typical probe length is 25 bases (i.e., 25 letters from *A*, *C*, *G*, *T* to represent a gene). There may be several different subsequences of these 25 bases. Then the mRNA (which is the labeled sample) is passed over the microarrays and the mRNA will bind to the complementary DNA corresponding to the gene better than the other DNA strings. Then the florescence levels of the different gene chips segments are measured, which in turn measures the amount of mRNA on that surface. This mRNA measurement serves as a surrogate to the expression level of the gene.

Machine Learning for Microarrays

The data from microarrays (gene chips) have been analyzed and used by machine learning researchers in two different ways:

1. Data points are genes. This is the case where the examples are genes while the features are the samples (measured expression levels of a single gene under a variety of conditions). The goal of this view is to categorize new genes based on the current set of examples.
2. Data points are samples (e.g., patients). This is the case where the examples are patients and the features are the measured expression levels of genes under one condition.

The problems have been approached in two different ways. In the ►Unsupervised Learning approach, the goal is to cluster the genes according to their expression levels or to cluster the patients (samples) based on their gene expression levels, or both. Hierarchical clustering is especially widely applied. As one of many examples, see Perou et al. (1999). In the ►Supervised Learning setting, the Class labels are the category of the genes or the samples. The latter is the more common supervised task, each sample being mRNA from a different patient (with the same cell type from each patient) or an organism under different conditions to learn a model that accurately predicts the class based on the features. The features could be the patient's expression values for each

gene, while the class labels might be the patient's disease state. We discuss this task further in the subsequent paragraphs.

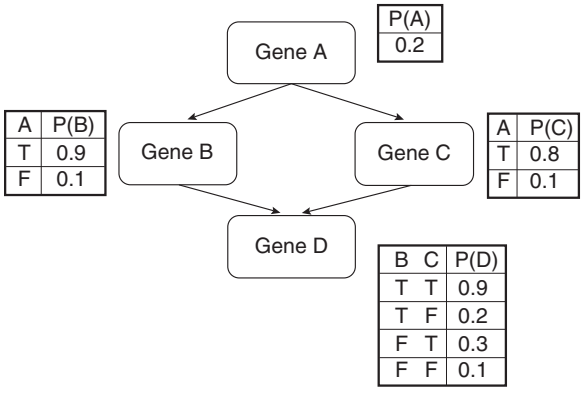
Yet another widely studied supervised learning task is to predict cancer vs. normal for a wide variety of cancer types. One of the significant lessons learned is that it is easy to predict cancer vs. normal in patients based on the gene expression by several machine learning techniques, largely regardless of the type of cancer. The main reason for this is that if cancer is present, many genes in the cancer cells "go haywire" and hence are very predictive of the cancer. The primary challenge in this prediction problem is the noise in the data (impure RNA, cross-hybridization, etc.).

Other related tasks that have been addressed include distinguishing related cancer types and distinguishing cancer from a related benign condition. An early success was a work by Golub et al. (1999), distinguishing acute myeloid leukemia and acute lymphoblastic leukemia (ALL). They used a weighted voting algorithm similar to Naive Bayes and achieved a very high accuracy. This result has been repeated on this data with many other machine learning (ML) approaches. Other work examined multiple myeloma vs. benign condition. This task is challenging because the benign condition is very similar to the cancer, and hence the machine learning algorithms had a difficult time predicting accurately. We refer to Hardin et al. (2004) for more details on the experiments.

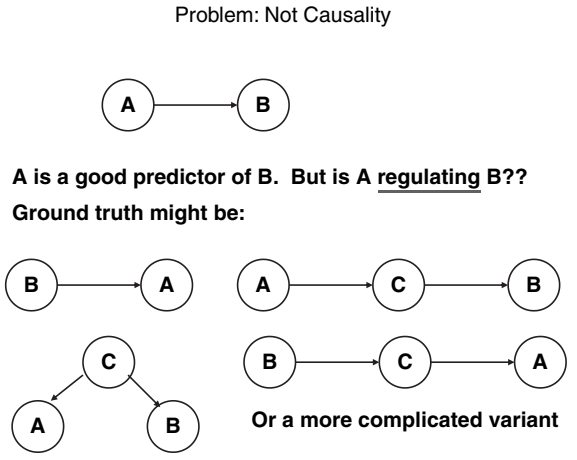
Another important lesson for machine learning researchers from this data type is that the biologists often do not want one predictive model, but a rank-ordered list of genes that a biologist can explore further with additional lab tests on certain genes. Hence, there is a need to present a small set of highly interesting genes to perform follow-up experiments on. Toward this end, statisticians have used mutual information or a t-test to rank the genes. When using a t-test, they check if the mean expression levels are different under the two conditions (cancer vs. normal), yielding a p-value. But the issue is that when working with a large number of genes (typically in the order of 30,000), there could be some genes with lower p-value by chance. This is known as the "multiple comparisons problem." One solution is to do a Bonferoni correction (multiply p-values by the number of genes), but this can be a drastic step and may eliminate all the genes. There are other methods such as false discovery rate (Storey & Tibshirani, 2003) that uses the notion of q-values. We do not go into detail of this method. But the key recommendation we make is that such a method should be used along with the supervised learning method, as the biological collaborators might be interested in the ranking of genes.

One of the most important research directions for the use of microarray data lies in the prognosis and treatment. The features are the same as those of diagnosis, but the class value becomes life expectancy for a given treatment (or a positive response vs. no response to a given treatment). The goal is to use the person's genes to make these predictions. An example of this is the breast cancer prognosis study (Van't Veer et al., 2002), where the goal is to predict good prognosis (no metastastis within 5 years of initial diagnosis) vs. poor prognosis. They used an ensemble of voting algorithms and obtained very good results. Nevertheless, an important lesson learned from this experiment and others was that when using ▶cross-validation, there is a need to tune parameters and perform feature selection *independently* on *each fold* of the cross-validation. There can be a large number of features, and it is natural to want to reduce the size of the data set before working with it. But reducing the number of features by some measure of correlation with the class, such as information gain, using the entire data set means that on each fold of cross-validation, information has leaked from the labeled test set into the training process – labels of test cases were used to eliminate many features from the training set. Hence, selecting features by looking at the entire data set can partially negate the effect of cross-validation, sometimes yielding accuracy estimates that are more than 10% points overly optimistic. Hence the entire training process of selecting features, tuning parameters, and learning a model must be repeated for every fold in cross-validation by looking only at the training data for that fold.

An important use of microarrays for prognosis and therapy is in the area of predictive personalized medicine (PPM). While we present the idea of PPM later in the paper, it must be mentioned that combining gene expression data with clinical trials of the patients to recommend the best treatment for the patients is a very exciting problem with promising impact in the area of PPM.



Biomedical Informatics. Figure 2. A simple Bayes net. The actual learning task typically involves thousands of variables



Biomedical Informatics. Figure 3. Why a learned Bayesian network may not be representing regulation of one gene by another

Bayesian Networks for Regulatory Pathways: ▶Bayesian Networks have been one of the successful machine learning methods used for the analysis of microarray data. Recall that a Bayes net is a directed acyclic graph, such as the one shown in Fig. 2 that defines a joint distribution over the variables using a set of conditional distributions. Friedman and Halpern (Friedman & Halpern, 1999) were the first to use Bayes nets for the microarrays data type. In particular, the problem that was considered was finding regulatory pathways in genes. This problem can be posed as a supervised learning task as follows:

- *Given*: A set of microarray experiments for a single organism under different conditions.
- *Do*: Learn a graphical model that accurately predicts expression of some genes in terms of others.

Friedman and Halpern showed that using statistical methods, a Bayes net representing the observations (expression levels of different genes) can be learned automatically. A main advantage of Bayes nets is that they can (potentially) provide insight into the interaction networks within cells that regulate the expression of genes. But one has to exercise caution, interpreting the arcs of a learned Bayes net as representing causality. For example in Fig. 2, one might interpret the network to mean that gene A causes gene B and gene C to be expressed, in turn influencing gene D. Note that however, the Bayes net in this case just denotes the correlation and not the causality, that is, the direction of an arc merely represents the fact that one variable is a good predictor of the other, as illustrated in Fig. 3.

One possible method of learning causality is to use *knock-out* methods [Pe'er, Regev, Elidan, & Friedman, 2001], where for 300 of the genes in *S. cerevisiae* (bakers' yeast), biologists have created a *knock-out mutant* or a genetic mutant lacking that gene. If the parent of a gene in the Bayes net is knocked out and the child's status remains unchanged, then it is unlikely that the arc from the parent to the child captures causality. A key limitation is that the mutants are not available for many organisms. Some other approaches such as RNAi have been proposed for more efficiently doing knock-outs, but a limitation is that RNAi typically reduces rather than eliminates expression of a gene.

Ong, Glassner, and Page (2002) used time-series data (data from the same organism at various time points) to partially address the issue of causality. They used these data to learn dynamic Bayesian networks in order to infer temporal direction for gene interactions, thereby getting a potentially better handle on causality. DBNs have been employed by other researchers for time-series gene expression data, and the approach has been extended to learn DBNs with continuous variables (Segal, Pe'er, Regev, Koller, & Friedman, 2005).

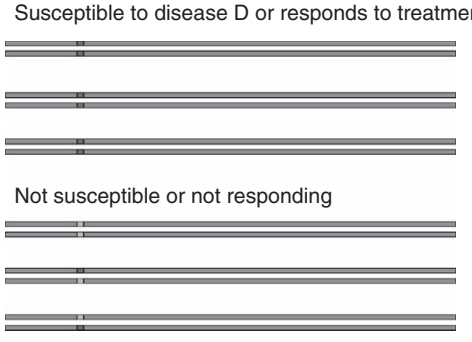
Single Nucleotide Polymorphisms

Single-Nucleotide Polymorphisms (SNPs) are individual base positions (i.e., single-nucleotide positions)

in DNA, where people (or the organism of interest) vary. Most of the variation in human DNA is due to SNPs variations. (There are other variations such as copy number, insertions and deletions that we do not consider in this article.) There are well over three million known SNPs in humans. Technologies such as *Illumina* or *Affymetrix whole-genome* scan can measure a million SNPs in short time. The measurement of these variations is an order of magnitude faster, easier, and cheaper than sequencing all the genes of the person.

It is believed that in the next decade, it will be possible to obtain the entire genome sequence for an individual human for under $1,000 (Mardis, 2006). If we had every human's entire sequence, it could be used to predict the susceptibility of diseases for humans or the adverse reactions to drugs for a certain subset of patients. The idea is illustrated in Fig. 4. Suppose the red dots in the figure are two copies of nucleotide *A*, and the green dots denote a different nucleotide, say *C*. As can be seen from the figure, people who respond to a treatment *T* (top half of the figure) have two copies of *A* (for instance, these could be the positive examples), while the people who do not respond to the treatment have at most one copy of *A* (negative examples and are presented in the bottom half of the figure). Now, we can imagine modeling the sequence to predict the susceptibility to a disease or responsiveness to a treatment.

SNP data can serve as a surrogate for the above problem. SNPs allow us to detect the variations among humans. An example of SNP data is presented in Fig. 5



Biomedical Informatics. Figure 4. Example application of sequencing human genes. The top half is the case, where patients respond to a treatment and the bottom is the case, where three patients do not respond to the treatment

for the prediction of *myeloma* cancer that is common with older people (with age > 70) and is very rare in younger people (age < 40). This data set consists of 40 people diagnosed with myeloma at young age and 40 people who weren't diagnosed till they were 70 when the disease is more common. Most SNP positions represent a pair of nucleotides and are typically restricted in the combinations of values they may assume. For example, in the figure, SNP 1 can take values from the three possible combinations $< C\ T,\ C\ C,\ T\ T >$ for its two positions. The goal is to use the feature values of the different SNPs to predict the class label which could be the susceptibility. That is, the goal is to determine genetic difference between people who got the disease at a young age vs. people who did not until they were old.

There is also the possibility of two patients having the same SNP pattern in the data but not the identical DNA. Patients 1 and 2 may have CT for the SNP1 and GA for SNP2, where both SNPs are on chromosome 1. But, Patient 1 has *C* on SNP1 in the same copy of chromosome 1 as the *G* in SNP2, whereas Patient 2 has *C* on the same copy as an *A*. Hence, while they have the same SNP pattern of CT and GA, they do not have identical DNA. The process of converting the data from the form in the Figure 5 below to the form above is called *Phasing*. From a machine learning perspective, there is a choice of either working with the *unphased* data or to use an algorithm for phasing. It turns out that phasing is very difficult and is an active research area. If there are a number of unrelated patients phasing is very hard. Hence many machine learning researchers work mainly with unphased data. Admittedly, there is a small loss of information with the unphased data that compensates for the difficulty of phasing.

Most biologists and statisticians using SNP data perform genome-wide associations studies (GWAS). The goal in this work is to find individual SNPs that are significantly associated with disease, that is, such that one of the SNP values, or alleles, raises the risk of disease. This is typically measured by "relative risk" or by "odds ratio," and significance is typically measured by statistical tests such as Wald test, Score test, or LRLR (►logistic regression log likelihood, where each SNP is used individually to predict disease, and log likelihood of the predictive model is compared to guessing under the null hypothesis that the SNP is not associated).

Person ▼ SNP ►	1		2		3		...	Class
Person 1	C	T	A	G	T	T	...	Old
Person 2	C	C	A	G	C	T	...	Young
Person 3	T	T	A	A	C	C	...	Old
Person 4	C	T	G	G	T	T	...	Young
. . .	.	.	.	...	.			
. . .	.	.	.	...	.			
. . .	.	.	.	...	.			

Biomedical Informatics. Figure 5. Example of SNP data

One of many examples is the use of SNPs to predict susceptibility to breast cancer (Easton et al., 2007).

The advantages of SNP data compared to microarray data are the following: (1) Because SNP analysis is typically performed on DNA from saliva or peripheral blood cells, a person's SNP pattern does not change with time or disease. If the SNPs are collected from a blood sample of a person aged 40 years, the SNP patterns are probably the same as when they were born. This gives more insight to the susceptibility of the person to many diseases. Hence, we do not see the widespread changes in SNP pattern with cancer, for example, that we see in microarray data from tumor samples. (2) It is easier to collect the samples. These can be obtained from the blood samples as against obtaining say, the biopsy of other tissue types.

The challenges of SNP data are as follows: (1) As explained earlier, the data is unphased. Algorithms exist for phasing (haplotyping), but they are error prone and do not work well with unrelated patient samples. They require the data to consist of related individuals in order to have a dense coverage. (2) ►Missing Values are more common than in microarray data. The good news is that the amount of missing values is decreasing substantially (down from 30–40% a few years ago to 1–2%). (3) The sheer volume of measurements – currently, it is possible to measure a million SNPs out of over three million SNPs in the human genome. While this provides a tremendous amount of potential information, the resulting high dimensionality causes problems for machine learning. As with gene expression microarray data, we have a multiple comparisons problem, so approaches such as Bonferoni correction or q-values from False Discovery Rate can again be applied. But even when a significant SNP is found, it usually only increases our accuracy at predicting disease by 2% or 3% points, because a single SNP typically either has a small effect or small penetrance (the variation is fairly rare – one value of the SNP is strongly predominant). So GWAS are missing a major opportunity to build predictive models by combining multiple SNPs with small effects – this is an exciting opportunity for machine learning.

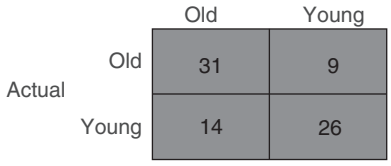
The supervised learning task can be defined as follows:

- *Given*: A set of SNP profiles each from a different patient.

  Phased: Nucleotides at each SNP position on each copy of *each chromosome* constitute the *features* and patient's *disease susceptibility* or *drug response* constitutes the *class*.

  Unphased: Unordered pair of nucleotides at each SNP position constitutes the *features* and patient's *disease susceptibility* or *drug response* constitutes the *class*.
- *Do*: Learn a model to predict the class based on the features.

We now briefly present one example of supervised learning from SNP data. (Waddell, Page, and Shaughnessy (2005)) found that there was evidence of a genetic component in predicting the blood cancer *multiple myeloma* as it was possible to distinguish the two cases significantly better than chance (71% accuracy). The results from using Support Vector Machines (SVMs) are



Biomedical Informatics. Figure 6. Results on predicting multiple myeloma, young (susceptible) vs. old (less susceptible), 3,000 SNPs

presented in Fig. 6. Similar results were obtained using a Naive Bayes model as well. Listgarten et al. (2004) also used the SNP data with the goal of predicting *lung cancer*. The accuracy of 69% obtained by them was remarkably similar to the task of predicting *multiple myeloma*. The best models for predicting lung cancer were also Naive Bayes and SVMs. There is a striking similarity between the two experiments on unrelated tasks using SNPs. When only the individual SNPs were considered, the accuracy for both the experiments fell to 60%.

The lessons learned from SNP data are the following: (1) ►Supervised learning algorithms such as ►Naive Bayes and ►SVM that can handle large number of features in the presence of smaller number of training examples can predict disease susceptibility at rates better than chance and better than individual SNPs. (2) Accuracies are much lower than the ones with microarray data. This is mainly due to the fact that we are predicting the susceptibility to the diseases (or the response to a drug) as against predicting whether a person already has the disease (as with the microarray data). While we are predicting using the genetic component, there are also many environmental components that are responsible for the diseases and the response. We are not considering such components in our model and hence the accuracies are often not very high. In spite of relatively lower accuracies, they give a different valuable insight to the human gene.

We now briefly outline a couple of exciting future directions for the use of SNP data. *Pharmacogenetics* is the problem of predicting drug response from SNP profile and has been gaining momentum over the past few years. This includes predicting drug efficacy and adverse reactions to certain drugs, given a person's SNP profile. A recent New England Journal of Medicine article showed that the analysis of SNPs can significantly improve the dosing model for the most widely used orally available blood thinner, Warfarin (IWPC, 2009). Another exciting direction is the combination of SNP data with other data types such as clinical data that includes the history of the patient and the lab tests and microarray data. The combination of these different data sets will not only improve the accuracy of the learned model but also provide a deeper insight to the different kinds of interactions that occur within a human, such as gene interactions with other drugs.

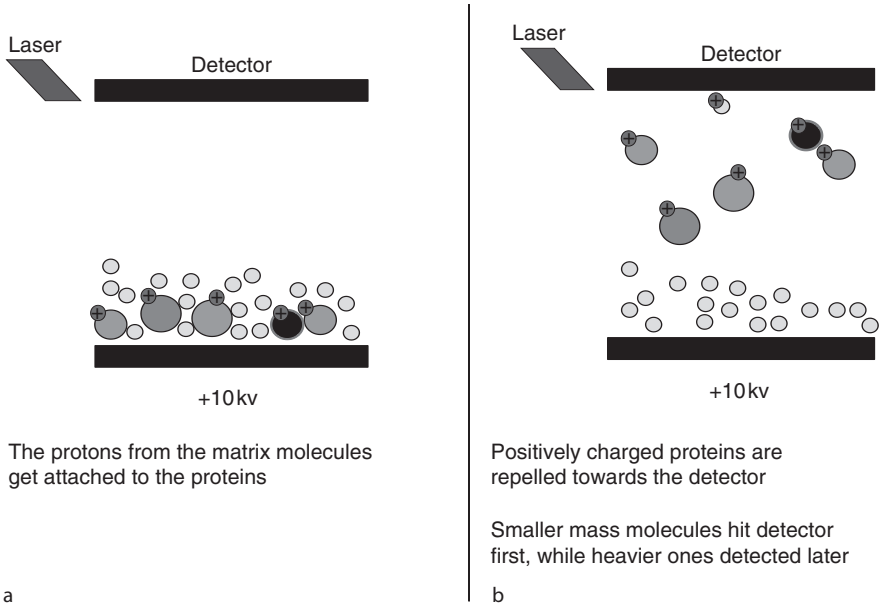
It should be mentioned that other genetic data types are becoming available and may be useful for supervised learning as well. These data types can provide additional information about DNA sequence beyond SNPs but without the expense of full genome sequencing. They include copy-number variations and exon-sequencing.

Mass Spectrometry and Proteomics

Microarrays are useful primarily because mRNA concentrations can serve as surrogates for protein concentrations and they are easier to measure. Though measuring protein concentrations directly is possible, it cannot be done in the same high-throughput manner as measuring mRNA. Recently, techniques such as *Mass Spectrometry* (MS or mass spec) have been successful in high-throughput measuring of proteins. Mass spec still does not given the complete coverage that microarrays provide, nor as good a quantitation.

Mass spectometry is improving on many fronts, using many technologies. As one example, we present *Time-Of-Flight (TOF) Mass Spectometry* illustrated in Fig. 7. This measures the time required for an ionized particle starting from the sample plate (bottom of the figure) to hit the detector. The key idea is to place some proteins (indicated as larger circles) into a matrix (smaller circles are the matrix molecules). Because of mass spec limitations, the proteins typically are digested (broken into smaller peptides), for example, by the compound trypsin. When struck by a laser, the matrix molecules release protons that attach themselves to the peptides or protein fragments (shown in (a)). Note that the plate where the peptides are present is positively charged. This causes the peptides to migrate toward the detector.

As can be seen in (b) of the figure, the molecules with smaller mass move faster toward the detector. The idea is to detect the number of molecules that hit the



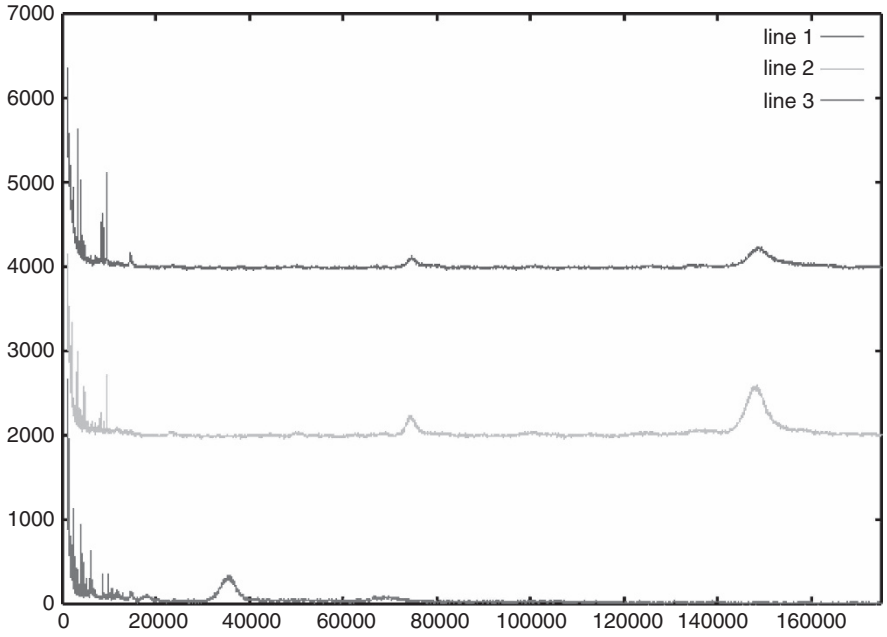
Biomedical Informatics. Figure 7. Time-Of-Flight mass spectrometry

detector at any given time. This makes it possible to use time as a surrogate for mass of the protein. The experiment is repeated a number of times, counting frequencies of "flight-times." Plotting time vs. the number of particles hitting the detector yields a spectrum as presented in Fig. 8. The figure shows three different fractions from the same sample. These kinds of spectra provide us an insight about the different types of proteins in a given sample. A technical detail is that sometimes molecules receive additional charge (additional protons) and hence fly faster. Therefore, the horizontal mass axis in a spectrum is actually a mass/charge ratio.

The main issues for machine learning researchers working with mass spectrometry data compared to microarray data are as follows: (1) There is a lot of ►Noise in the data. The noise is due to extra peaks from handling of sample, from machine and environment (e.g., electrical noise). Also the mass to charge values may not exactly align across the spectra; the accuracy of the mass/charge values is the resolution of the mass spec. (2) Intensities (peak heights) are not calibrated across the spectra, making quantification difficult. This is to say that if one spectrum is compared to another, and if one of them has more intensity at a particular mass/charge, it does not necessarily mean that the levels of the peptide at that mass/charge are higher in that spectrum. (3) Another issue is that the mass spectrometry data is not as comprehensive as microarray data, in that it is not possible to measure all peptides (typically only several hundred of them can be obtained). To get the best results, there is a need to fractionate the sample beforehand, getting different groups of proteins in different subsamples (fractions). (4) As already mentioned, the proteins themselves typically must be broken down (digested) into smaller peptides in order to get accurate readings from the mass spec. But this means processing is needed afterward not only to determine from a spectrum which peptides are present but also from that determination which proteins are present. It is worth noting that some of these challenges are being partially addressed by ongoing improvements in mass spectrometry technologies, including the use of "tandem mass spectrometry."

This data type opens up a lot of possibilities for machine learning research. Some of the learning tasks include:

- Learn to predict proteins from spectra, when the organism's proteome (full set of proteins) is known.
- Learn to identify isotopic distributions (combinations of multiple peaks for a given molecule



Biomedical Informatics. Figure 8. Example spectra from a competition by Lin et al.

arising from different isotypes of carbon, nitrogen. and oxygen).

- Learn to predict disease from either proteins, peaks or isotopic distributions as features.
- Construct pathway models.

We will now present one case study that was successful and generated a lot of interest – *Early Detection of Ovarian Cancer* (Petricoin et al., 2002). Ovarian cancer is difficult to detect early, often leading to poor prognosis. The goal of this work was to predict ovarian cancer from blood samples. To this effect, the researchers trained and tested on mass spectra from blood serum. They used 100 training cases (50 positive) and used a held-out test set of 116 cases (50 positive). The results were extremely impressive (100% sensitivity, 95% specificity).

While the results were extremely impressive and while the machine learning methodology seemed very sound, it turns out that the preprocessing stage of the data may have introduced errors (Baggerly, Morris, & Combes, 2004). Mass spectrometry is very sensitive to the external factors as well. For instance, if we run cancer samples on Monday and normal samples on Wednesday, it is possible that we could get differences from variations in the machine or nearby electrical equipment that is running on Monday but not Wednesday. Hence, one of the important lessons learned from this data type is the need for *careful randomization* of the data samples. This is to say that we should sample the positive and negative samples under identical conditions. It should not be the case that the positive examples are run through the machine on one day and the negatives on the other day. Any preprocessing of the data must be performed similarly.

While mass spectrometry is a widely used type of high-throughput proteomic data, other types of data are also important and are briefly covered next.

Protein Structures

X-ray crystallography and nuclear magnetic resonance are widely used to determine the three-dimensional structures of proteins. Predicting protein structures has been a very fertile field for machine learning research for several decades.

While the amino acid sequence of a protein is called its primary structure, it is more difficult to determine secondary structure and tertiary (3D) structure. Secondary structure maps subsequences of the primary

structure in the three classes of alpha helix (helical structures akin to a telephone cord, often denoted by A), beta strand (which comes together with other strand sections to form planar structures called beta sheets, often denoted by B), and less descript regions referred to as coil, or loop regions, often denoted by C.

Predicting secondary structure and tertiary structure has been a popular topic for machine learning for many years, because training data exists yet it is difficult and expensive to experimentally determine structures. We will not attempt to survey all the work in this area. Waltz and colleagues (Zhang, Mesirov, & Waltz, 1992) showed the benefit of applying neural networks to the task of secondary structure prediction, and the best secondary structure predictors (e.g., Rost & Sander, 1993) have continued to be constructed by machine learning over the years. Approaches for predicting the tertiary structure have also relied heavily on machine learning and include ab initio prediction (e.g., Bonneau & Baker, 2001), prediction aided by crystallography data (e.g., DiMaio et al., 2007), and homology-based prediction (by finding similar proteins). For over a decade, there has been a regular competition in the prediction of protein structures (Critical Assessment of Structure Prediction [CASP]).

Protein–Protein Interactions

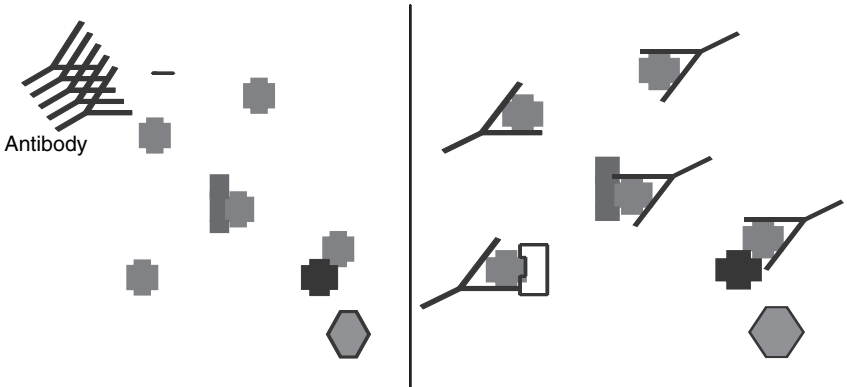
Another proteomics data type is protein–protein interactions. This is illustrated in Fig. 9. The idea is to identify proteins that interact with the current protein say *P*. Generally, this is performed as follows: In the sample, there are some proteins of type *X* (shown in pink in the figure) and other types of proteins. Proteins that interact with *X* are bonded to *X*. Then antibodies (shown as Y-shaped green objects) are introduced in the sample. The idea of antibodies is to collect the proteins of type *X*. Once the antibodies have collected all protein *X*'s in the sample, they can be analyzed through mass spectrometry presented earlier.

A particularly high-throughput way of measuring protein–protein interactions is through "ChIP-chip" data. The supervised learning tasks for this task include:

- Learn to predict protein–protein interactions: Protein three-dimensional structures may be critical.
- Use protein–protein interactions in construction of pathway models.
- Learn to predict protein function from interaction data.

Related Data Types

- *Metabolomics* measures concentration of each low-molecular-weight molecule in sample. These typically are *metabolites*, or small molecules produced or consumed by reactions in biochemical pathways. These reactions are typically catalyzed by proteins (specifically, enzymes). This data typically uses mass spectrometry.



Biomedical Informatics. Figure 9. Schematic of antibody-based identification of protein–protein interactions

- *ChIP-chip* data measures protein–DNA interactions. For example, transcription factors are proteins that interact with DNA in specific locations to alter transcription of a nearby gene.
- *Lipomics* is analogous to metabolomics, but measuring concentrations of Lipids rather than metabolites. These potentially help induce biochemical pathway information or to help disease diagnosis or treatment choice.

High-Throughput Screening Data for Drug Design

The typical steps in designing a drug are: (1) Identifying a target protein – for example, while developing an antibiotic, it will be useful to find a protein that belongs to the bacteria that we are interested in and find a small molecule that will bind to that protein. In order to perform this, we need the knowledge of proteome/genome and the relevant biological path ways. (2) Determining the target site structure once the protein has been identified – this is typically performed using crystallography. (3) Finding a molecule that will bind to the target site. These steps are presented in Fig. 10.

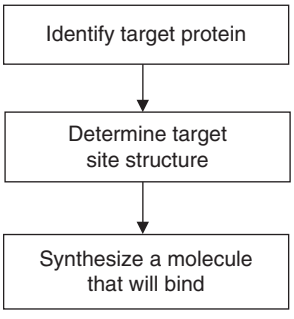
The molecules that bind to the target may have a number of other problems and hence they cannot directly be used as a drug. Some common problems are as follows: (1) They may bind too tightly or not tightly enough. (2) They may be toxic. (3) They may have unanticipated side effects in the body. (4) They may break down as soon as they get into the body or may not leave the body soon enough. (5) They may not get to the right target in the body (e.g., cross blood–brain barrier). (6) They may not diffuse from gut to bloodstream. Also, since the organisms are different, even if a molecule works in the test tube and in animal studies, it may fail in clinical trials. Also while a molecule may work for some people, it may not work for others. Conversely, while some molecules may cause harmful side effects in some people, they may not do so in others.

Often pharmaceutical companies will use robotic high-throughput screening assays to test many thousands of molecules to see if they bind to the target protein, and then computational chemists will work to determine the commonalities that allow them to bind to the target as often the structure of the target protein cannot be determined. The process of discovering the commonalities across the different molecules presents a great opportunity for machine learning research. The first study of this task using machine learning was by Dietterich, Lathrop, and Lozano-Perez and led to the formulation of Multi-Instance Learning. Yet, another machine learning task could be to predict the reactions of the patients to the drugs.

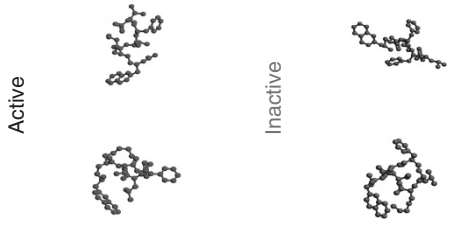
High-Throughput Screening: When the target structure is unknown, it is a common practice to test many molecules (1,000,000) to find some that bind to the target. This is called as *High-Throughput Screening*. Hence, it is important to infer the shape of the target from three-dimensional structural similarities. The shared three-dimensional structure is called as *pharmacophore*. This is a perfect example of a machine learning task with a spatial target and is presented in Fig. 11.

Given: A set of molecules, each labeled by activity (binding affinity for a target protein) and a set of low-energy conformers for each molecule

Do: Learn a model that accurately predicts the activity (may be Boolean or real valued).



Biomedical Informatics. Figure 10. Steps involved in drug design



Biomedical Informatics. Figure 11. An example of structure learning

The common machine learning approaches taken toward solving this problem are:

1. Representing a molecule by thousands to millions of features and use standard techniques (KDD, 2001)
2. Representing each low-energy conformer by feature vector and use multiple-instance learning (Jain et al., 1994)
3. Relational learning – using either Inductive Logic Programming techniques (Finn, Muggleton, Page, & Srinivasan, 1998) or Graph Mining

Thermolysin Inhibitors: We present some results of relational learning algorithms on thermolysin inhibitors data set (Davis, 2007a). Thermolysin belongs to the family of metalloproteases and plays roles in physiological processes such as digestion and blood pressure regulation. The molecules in the data set are known inhibitors of thermolysin. Activity for these molecules is measured in $pKi = -log\ Ki$, where Ki is a dissociation constant, measuring the ratio of the concentrations of bound product to unbound constituents. A higher value indicates a stronger affinity for binding. The data set that was used had the ten lowest energy conformations (as computed by the SYBYL software package [www.tripos.com]) for each of 31 thermolysin inhibitors along with their activity levels.

The key results for this data set using the relational algorithm SAYU (Davis, 2007b) were:

- Ten five-point pharmacophore identified, falling into two groups (7/10 molecules):
  - Three "acceptors," one hydrophobe, and one donor
  - Four "acceptors," and one donor
- Common core of Zn ligands, Arg203, and Asn112 interactions identified
- Correct assignments of functional groups
- Correct geometry to 1 Å tolerance
- Increasing tolerance to 1.5 Å finds common six-point pharmacophore including one extra interaction

Antibacterial Peptides: This is a data set of 11 pentapeptides showing activity against Pseudomonas aeruginosa (Spatola, Page, Vogel, Blondell, & Crozet, 1999). There are six active pharmacophores with < 64 μg/ml of IC_{50} and five inactives. The pharmacophore that has been identified is presented in Table 1.

Biomedical Informatics. Table 1 Identified Pharmacophore

A molecule M is active against *Pseudomonas aeruginosa* if it has a conformation B such that
M has a hydrophobic group C
M has a hydrogen acceptor D
The distance between C and D in conformation B is 11.7 Å
M has a positively charged atom E
The distance between C and E in conformation B is 4 Å
The distance between D and E in conformation B is 9.4 Å
M has a positively charged atom F
The distance between C and F in conformation B is 11.1 Å
The distance between D and F in conformation B is 12.6 Å
The distance between E and F in conformation B is 8.7 Å
Tolerance 1.5 Å

Dopamine Agonists: The last data set that we present here consists of dopamine agonists (Martin et al., 1993). Dopamine works as a neurotransmitter in the brain, where it plays a major role in the movement control. Dopamine agonists are molecules that function like dopamine and produce dopamine-like effects and can potentially be used to treat diseases such as Parkinson's disease. The data set had 23 dopamine agonists along with their activity levels. The pharmacophore identified using Inductive Logic Programming is presented in Table 2.

Electronic Medical Records (EMR) and Personalized Medicine

Predictive personalized medicine (PPM) is a vision of the future, whose parts are beginning to come into place now. Under this vision, physicians can construct safer and more effective prevention and treatment plans for

each patient. This is rendered possible by predicting the impact of treatments on patients – their effectiveness for different classes of patients, adverse reactions of certain drugs that are prescribed to the patients, and susceptibility of different types of patients to diseases. PPM can become a reality due to three reasons: The first is the widespread use by many clinics of *Electronic Medical Records* (EMR also called as *Electronic Health Records* – EHR). The second is that whole-genome scan technology makes it possible in one experiment, for well under $1,000, to measure for one patient a half million to one million SNPs, or individual positions in the DNA where humans vary. The third key reason is the advancement of statistical modeling (machine learning) methods in the past decade that can handle large relational longitudinal databases with significant amount of noise. The first two reasons make it possible for the clinics to have a relational database of the form presented in Fig. 12.

Biomedical Informatics. Table 2 Pharmacophore Identified for Dopamine Agonists

Molecule A has the desired activity if
• In conformation B molecule A contains a hydrogen acceptor at C
• In conformation B molecule A contains a basic nitrogen group at D
• The distance between C and D is 7.05966 ± 0.75 Å
• In conformation B molecule A contains a hydrogen acceptor at E
• The distance between C and E is 2.80871 ± 0.75 Å
• The distance between D and E is 6.36846 ± 0.75 Å
• In conformation B molecule A contains a hydrophobic group at F
• The distance between C and F is 2.68136 ± 0.75 Å
• The distance between D and F is 4.80399 ± 0.75 Å
• The distance between E and F is 2.74602 ± 0.75 Å

Given such a database, it is conceivable to use existing machine learning algorithms for achieving the goal of PPM. These algorithms could focus on predicting which patients are at risk (pos and neg examples). Another task is predicting which patients will respond to a specific treatment – a set of patients who have undergone specific treatments in order to learn predictive models that could be extended to similar patients of the population. Similarly, it is possible to focus on certain drugs and their adverse reactions and use them to predict the adverse reactions of similar drugs that are released in the market. In this work, we focus on the machine learning solutions to predicting adverse drug reactions for different drugs.

There are actually at least three different tasks for machine learning in predicting Adverse Drug Events (ADEs).

Patient ID	Gender	Birthdate
P1	M	3/22/63

Patient ID	Date	Physician	Symptoms	Diagnosis
P1	1/1/01	Smith	Palpitations	Hypoglycemic
P1	2/1/03	Jones	Fever, Aches	influenza

Patient ID	Date	Lab Test	Result
P1	1/1/01	blood glucose	42
P1	1/9/01	blood glucose	45

Patient ID	SNP1	SNP2	⋯	SNP500K
P1	AA	AB		BB
P2	AB	BB		AA

Patient ID	Date Prescribed	Date Filled	Physician	Medication	Dose	Duration
P1	5/17/98	5/18/98	Jones	Prilosec	10 mg	3 months

Biomedical Informatics. Figure 12. Electronic Health Records (dramatically simplified) – most data currently do not include SNP information but are anticipated in the future

Task 1:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Construct a model to predict a minimum efficacious dose of drug D, because a minimum dose is less likely to induce an ADE.

An example of this task is predicting the "stable dose" of the blood-thinner Warfarin (Coumadin) for a patient (McCarty, Wilke, Giampietro, Wesbrook, & Caldwell, 2005). A stable dose of Warfarin yields the desired degree of anticoagulation, whereas a higher dose can lead to bleeding ADEs; the stable dose for a patient is currently found by trial and error, modifying the dose and measuring the degree of anticoagulation. The cited study shows that a learned dosing model can predict a significantly better starting dose (significantly closer to the final "stable dose") than the 5 mg/day starting dose currently used in many clinics.

Task 2:

Given: Patient data (from claims databases and/or EMRs), a drug D, and an adverse event E

Do: Construct a model to predict which patients are likely to suffer the adverse event E if they take D.

In this second task, we assume that the association between D and E already has been hypothesized. We seek to construct models that can predict who will suffer a given event if they take the drug. Here, whether the patient will suffer adverse event *E* is the class variable to be predicted. This task is important for *personalized medicine*, as accurate models for this task can be used to identify patients who should not be given a particular drug. An earlier study has demonstrated the benefit of a Statistical Relational Learning (SRL) system called SAYU (Davis, 2007b) over standard machine learning approaches with a feature-vector representation of the EHR, for the task of predicting which users of cox2 inhibitors would have an MI.

Task 3:

Given: Patient data (from claims databases and/or EMRs) and a drug D

Do: Determine if evidence exists that associates D with a previously unanticipated adverse event.

This third task is the most challenging because no associated event has been hypothesized. There is a need to identify the response variable to be predicted. In brief, the major approach for this task is to use machine learning "in reverse." We seek a model that can predict which patients are on drug *D* using the data after they start the drug (left censored) and also censoring the indications of the drug. If a model can predict (with accuracy better than chance on held-aside data) which patients are taking the drug, there must be some combination of variable settings more common among patients on the drug. Because we have left censored, in theory, this commonality should not consist of common symptoms, but common effects, presumably from the drug. The model can then be examined by the experts to see if it might indicate a possible new adverse event for the drug.

The preceding use of machine learning "in reverse" actually can be viewed as Subgroup Discovery (Wrobel, 1997; Klösgen, 2002), finding a subgroup of patients on drug *D* who share some subsequent clinical events. The learned model – say an IF-THEN rule – need not correctly identify everyone on the drug but rather merely a subgroup of those on the drug, while not generating many false positives (individuals not on the drug). This task poses several different challenges that traditional ML methods will find difficult to handle.

First, the data is *multi-relational*. There are several objects such as doctors, patients, drugs, diseases, and labs that are connected through relations such as visits, prescriptions, diagnoses, etc. If traditional machine learning (ML) techniques are to be employed on this problem, they require flattening the data into a single table. All known flattening techniques such as computing a join or summary features result in either (1) changes in frequencies on which machine learning algorithms critically depend or (2) loss of information. They also typically result in loss of some correlations between the objects and explosion in database size. Second, the data is *non-i.i.d.*, as there are relationships between the objects and between different rows within a table. Third, there are *arbitrary* numbers of patient visits, diagnoses, and prescriptions for different patients. This is to say that there is no fixed pattern in the diagnoses and prescriptions of the patients. It is incorrect to assume that the patients are diagnosed a fixed number of times or to assume only the last diagnosis is relevant. To predict the adverse reactions to a drug, it is important to consider the other drugs that the patient is prescribed or has been prescribed in the past, as well as past diagnoses and laboratory results. To capture

these interactions, it is critical to explicitly model time since the interactions are highly *temporal*. Some drugs taken at the same time can lead to side effects while in some cases, drugs taken after one another cause side effects. It is important to capture such interactions to be able to make useful predictions for the physicians and the Federal Drug Authority (FDA). In this work, we focus on this hardest task and present the results on two data sets.

Cox2 Inhibitors: Recently, a study was performed to see if there were any unanticipated adverse events that occurred when subjects used cox2 inhibitors (Vioxx, Celebrex, and Bextra). Cox2 inhibitors are a nonsteroidal anti-inflammatory class of drugs that were used to reduce joint pain. Vioxx, Celebrex, and Bextra were approved for use in the late 1990s and were ranked as one of the top therapeutic drugs in the USA. Several clinical trials were conducted, and the APPROVe trial (focused on Vioxx outcomes) showed an increase of adverse events from myocardial infarction, stroke, and vascular thrombosis. The manufacturer withdrew Vioxx from the market shortly after the results were published. The other cox2 inhibitor drugs were discontinued shortly thereafter.

This study utilized the Marshfield Clinic's Personalized Medicine Research Project (McCarty, Wilke, Giampietro, Wesbrook, & Caldwell, 2005) (PMRP) cohort consisting of approximately 19,700+ subjects. The PMRP cohort included adults aged 18 years and older, who reside in the Marshfield Epidemiology Study Area (MESA). Marshfield has one of the oldest internally developed Electronic Medical Records (Cattails MD) in the USA, with coded diagnoses dating back to the early 1960s. Cattails MD has over 13,000 users throughout central and northern Wisconsin.

Since the data is multi-relational, an Inductive Logic Programming (Muggleton & Raedt, 1994) system, *Aleph* (Srinivasan, 2001) was used to learn the models. Aleph learns rules in the form of Prolog clauses and scores rules by positive examples covered (P) minus negative examples covered (N). Seventy-five percent of the data was used for training and rule development, while the remaining 25% was used for testing. There were 14,654 subjects within the PMRP cohort that had medication records. Within this cohort, almost 20% of the subjects indicated use of a cox2 inhibitor, and more specifically, 8.5% indicated the use of Vioxx. Approximately, 3.5% of this cohort had an indicated use of clopidogrel biosulfate (Plavix).

Biomedical Informatics. Table 3 Cox2 Inhibitor Test Data Results

	Actual		
Rule	+	–	
+	438	158	596
–	269	549	818
	707	707	1,414
Accuracy	0.69801		

Aleph generated thousands of rules and selected a subset of the "best" rules that were based on the scoring algorithm. The authors also developed specific hypotheses to test for known adverse events to validate the approach (indicated by # A). This rule was: cox2(A):- diagnoses(A, _,'410'). It states that if finding (A): the subject would have the diagnosis coded as *410* (*myocardial infarction*). Aleph also provided summary statistics on model performance for identifying subjects on cox2 inhibitors, as indicated in Table 3. If we assume that the probability of being on the cox2 inhibitor is greater than. 5 (the common threshold), then the model has a predictive probability of 69% to predict cox2 inhibitor use.

OMOP Challenge: Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated data sets to identify adverse drug events. The simulated data sets are modeled after real observational data sources but are comprised of hypothetical persons with fictional drug exposure and health outcomes occurrence. The data sets are constructed such that the relationships between the fictional drugs and fictional outcomes are well characterized as *true* and *false* associations. That is, hypothetical persons are created and assigned fictional drug exposure periods and instances of health outcomes based on random sampling from probability distributions that define the relationships between the fictional drugs and outcomes. The relationships created within the simulated data sets are contrived but are representative of the types of relationships observed within real observational data sources. OMOP has made a

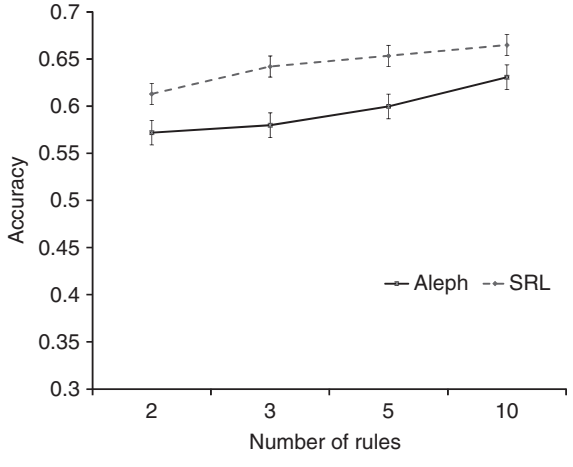
simulated data set and the simulator itself publicly available as part of the OMOP Cup Data Mining Competition (http://omopcup.orwik.com).

Aleph was used to learn rules from a subset of the data (about 10,000 patients). Each patient had a record of drugs and diagnoses (conditions) with dates attached. A few examples of the rules learned by Aleph in this data set are:

```
on_drug(A):-  condition_occurrence(B,C,A,D,
                E,3450,F,G,H)
on_drug(A):-  condition_occurrence(B,C,A,D,E,
                140,F,G,H)
              condition_occurrence(I,J,A,K,L,
                1487,M,N,O)
```

The first rule identifies drug *3450* as interesting, while the second rule identifies two other drugs as interesting when predicting the reaction for person *A*. With about 150 rules, Aleph was able to achieve a 67% coverage. The results were compared against a Statistical Relational Learning technique (SRL) (Getoor & Taskar, 2007) that uses a probability distribution on the rules. The results are presented in Fig. 13. As expected, with a small number of rules, SRL has a better performance than Aleph, but as the number of rules increase, they converge on the same performance.

The leading approaches in the first OMOP Cup include a machine learning approach based on random forests as well as several approaches based on techniques from epidemiology such as disproportionality analysis. At the time of this writing further details, as well as plans for future competitions, are available at http://omopcup.orwik.com/.



Biomedical Informatics. Figure 13. Results of OMOP data

Identifying previously unanticipated ADEs, predicting who is most at risk for an ADE, and predicting safe and efficacious doses of drugs for particular patients are all important needs for society. With the recent advent of "paperless" medical record systems, the pieces are in place for machine learning to help meet these important needs.

Conclusion

In this work, we aim to survey the abundant opportunities in biomedical applications to machine learning researchers by presenting several data types to which machine learning techniques have been applied successfully or showing tremendous promise. One of the most important developments in biology and medicine over the last few years is the availability of technologies that can produce large volumes of data. This in turn has necessitated the need for processing large volumes of data in a reasonable amount of time, presenting the perfect setting for machine learning algorithms to have an impact. We outlined several data types including gene expression microarrays (measuring mRNA), mass spectrometry (measuring proteins), SNP chips (measuring genetic variation), and Electronic Medical/Health Records (EMR/EHRs).

The key lessons learned from all these data types are as follows: (1) Even if the number of features is greater than the number of data points (e.g., predicting cancer from microarray data), we can do well provided the features are highly predictive. (2) Careful randomization of data samples is necessary. (3) It is very easy to overfit the data and hence robust techniques such as voted ▶decision stumps, ▶naive Bayes or linear ▶SVMs are in general very useful tools for such data sets. (4) ▶Bayes nets do not give us causality and hence knock-out experiments (▶active learning) and ▶DBNs with ▶time-series data can help. (5) Multi-relational methods such as SRL and ILP are helpful for predictive personalized medicine due to the relational nature of the data. (6) Mostly, the collaborators are interested in measures other than just accuracy. Comprehensibility, privacy, and ranking are other criteria that are important to biologists.

This chapter is necessarily incomplete because so many exciting tasks and data types exist within biology

and medicine. While we have touched on many of the leading such data types, other related ones also exist. For example, there are many opportunities in analyzing genomic and protein sequences (Learning Models of Biological Sequences). Other opportunities exist within phylogenetics, for example, see work by Heckerman and colleagues on HIV (Carlson et al., 2009). New technologies such as optical mapping are constantly being developed and refined (Ananiev et al., 2008). Machine learning has great potential for developing models for computer-aided diagnosis (CAD), for example, for mammography (Burnside et al., 2009). Data types such as metabolomics and auxotropic growth experiments raise opportunities for active learning and for automatic revision of biological network models, for example, as in the Robot Scientist projects (Jones et al., 2004; Oliver et al., 2009). Incorporation of multiple data types can further help in mapping out the regulatory entities and networks of an organism (Noto & Craven, 2006). It is our hope that this article will encourage some machine learning researchers to delve deeper into these and other related opportunities.

Acknowledgment

We would like to thank Elizabeth Burnside, Michael Caldwell, Mark Craven, Jesse Davis, Lingjun Li, David Madigan, Sean McIlwain, Michael Molla, Irene Ong, Peggy Peissig, Patrick Ryan, Jude Shavlik, Michael Sussman, Humberto Vidaillet, Michael Waddell and Steve Wesbrook.

Cross References

►Learning Models of Biological Sequences

Recommended Reading

Ananiev, G. E., Goldstein, S., Runnheim, R., Forrest, D. K., Zhou, S., Potamousis, K., Churas, C. P., Bergendah, V., Thomson, J. A., & David, C. (2008). Schwartz1. Optical mapping discerns genome wide DNA methylation profiles. *BMC Molecular Biology*, *9*, doi:10.1186/1471-2199-9-68.

Baggerly, K., Morris, J. S., & Combes, K. R. (2004). *Reproducibility of seldi-tof protein patterns in serum: Comparing datasets from different experiments. Bioinformatics*, *20*, 777–785.

Bonneau, R., & Baker, D. (2001). Ab initio protein structure prediction: Progress and prospects. *Annual Review of Biophysics and Biomolecular Structure*, *30*, 173–189.

Burnside, E. S., Davis, J., Chhatwal, J., Alagoz, O., Lindstrom, M. J., Geller, B. M., Littenberg, B., Kahn, C. E., Shaffer, K., & Page, D. (2009). Unique features of hla-mediated hiv evolution in a mexican cohort: A comparative study. *Radiology*, *251*, 663–672.

Carlson, J., Valenzuela-Ponce, H., Blanco-Heredia, J., Garrido-Rodriguez, D., Garcia-Morales, C., Heckerman, D., et al. (2009). Unique features of hla-mediated hiv evolution in a mexican cohort: A comparative study. *Retrovirology*, *6*(72), 39.

Davis, J., Costa, V. S., Ray, S., & Page, D. (2007a). An integrated approach to feature construction and model building for drug activity prediction. In *Proceedings of the 24th international conference on machine learning (ICML)*.

Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., & Costa, V. S. (2007b). Change of representation for statistical relational learning. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI)*.

DiMaio, F., Kondrashov, D., Bitto, E., Soni, A., Bingman, C., Phillips, G., & Shavlik, J. (2007). Creating protein models from electron-density maps using particle-filtering methods. *Bioinformatics*, *23*, 2851–2858.

Easton, D. F., Pooley, K. A., Dunning, A. M., Pharoah, P. D., et al. (2007). Genome-wide association study identifies novel breast cancer susceptibility loci. *Nature*, *447*, 1087–1093.

Finn, P., Muggleton, S., Page, D., & Srinivasan, A. (1998). Discovery of pharmacophores using the inductive logic programming system progol. *Machine Learning, 30*(1, 2), 241–270.

Friedman, N. (2000). Being Bayesian about network structure. In *Machine Learning*, *50*, 95–125.

Friedman, N., & Halpern, J. (1999). Modeling beliefs in dynamic systems. part ii: Revision and update. *Journal of AI Research*, *10*, 117–167.

Furey, T. S., Cristianini, N., Duffy, N., Bednarski, B. W., Schummer, M., & Haussler, D. (2000). Support vector classification and validation of cancer tissue samples using microarray expression. *Bioinformatics*, *16*(10), 906–914.

Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge, MA: MIT Press.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., et al. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, *286*, 531–537.

Hardin, J., Waddell, M., Page, C. D., Zhan, F., Barlogie, B., Shaughnessy, J., et al. (2004). Evaluation of multiple models to distinguish closely related forms of disease using DNA microarray data: An application to multiple myeloma. *Statistical Applications in Genetics and Molecular Biology*, *3*(1).

Jain, A. N., Dietterich, T. G., Lathrop, R. H., Chapman, D., Critchlow, R. E., Bauer, B. E., et al. (1994). Compass: A shape-based machine learning tool for drug design. *Aided Molecular Design*, *8*(6), 635–652.

Jones, K. E., Reiser, F. M., Bryant, P. G. K., Muggleton, C. H., Kell, S., King, D. B., et al. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427, 247–252.

KDD cup (2001). http://pages.cs.wisc.edu/ dpage/kddcup2001/.

Klösgen, W. (2002). *Handbook of data mining and knowledge discovery, chapter 16.3: Subgroup discovery*. New York: Oxford University Press.

Listgarten, J., Damaraju, S., Poulin, B., Cook, L., Dufour, J., Driga, A., et al. (2004). Predictive models for breast cancer

susceptibility from multiple single nucleotide polymorphisms. *Clinical Cancer Research*, 10, 2725–2737.
Mardis, E. R. (2006). Anticipating the 1,000 dollar genome. *Genome Biology*, *7*(7), 112.
Martin, Y. C., Bures, M. G., Danaher, E. A., DeLazzer, J., Lico, I. I., & Pavlik, P. A. (1993). A fast new approach to pharmacophore mapping and its application to dopaminergic and benzodiazepine agonists. *Journal of Computer Aided Molecular Design*, *8*, 751–758.
McCarty, C., Wilke, R. A., Giampietro, P. F, Wesbrook, S. D., & Caldwell, M. D. (2005). Personalized Medicine Research Project (PMRP): Design, methods and recruitment for a large population-based biobank. *Personalized Medicine*, *2*, 49–79.
Molla, M., Waddell, M., Page, D., & Shavlik, J. (2004). Using machine learning to design and interpret gene expression microarrays. *AI Magazine*, *25*(1), 23–44.
Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19*(20), 629–679.
Noto, K., & Craven, M. (2006). A specialized learner for inferring structured cis-regulatory modules. *BMC Bioinformatics*, *7*(528), doi:10.1186/1471-2105-7-528.
Oliver, S. G., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., et al. (2009). The automation of science. *Science*, *324*, 85–89.
Ong, I., Glassner, J., & Page, D. (2002). Modelling regulatory pathways in e.coli from time series expression profiles. *Bioinformatics*, *18*, 241S–248S.
Pe'er, D., Regev, A., Elidan, G., & Friedman, N. (2001). Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, *17*, 215–224.
Perou, C., Jeffrey, S., Van De Rijn, M., Rees, C. A., Eisen, M. B., Ross, D. T., et al. (1999). Distinctive gene expression patterns in human mammary epithelial cells and breast cancers. *Proccedings of National Academy of Science*, *96*, 9212–9217.
Petricoin, E. F., III, Ardekani, A. M., Hitt, B. A., Levine, P. J., Fusaro, V. A., Steinberg, S. M., et al. (2002). Use of proteomic patterns in serum to identify ovarian cancer. *Lancet*, *359*, 572–577.
Rost, B., & Sander, C. (1993). Prediction of protein secondary structure at better than 70 accuracy. *Journal of Molecular Biology*, *232*, 584–599.
Segal, E., Pe'er, D., Regev, A., Koller, D., & Friedman, N. (April 2005). Learning module networks. *Journal of Machine Learning Research*, *6*, 557–588.
Spatola, A., Page, D., Vogel, D., Blondell, S., & Crozet, Y. (1999). Can machine learning and combinatorial chemistry co-exist? In *Proceedings of the American Peptide Symposium*. Kluwer Academic Publishers.
Srinivasan, A. (2001). The aleph manual. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.
Storey, J. D., & Tibshirani, R. (2003). Statistical significance for genome-wide studies. *Proceedings of the National Academy of Sciences*, *100*, 9440–9445.
The International Warfarin Pharmacogenetics Consortium (IWPC) (2009). Estimation of the Warfarin Dose with Clinical and Pharmacogenetic Data. *The New England Journal of Medicine*, 360:753–764.
Tucker, A., Vinciotti, V., Hoen, P. A. C., Liu, X., & Famili, A. F. (2005). Bayesian network classifiers for time-series microarray data. *Advances in Intelligent Data Analysis VI, 3646*, 475–485.
Van't Veer, L. L., Dai, H., van de Vijver, M. M., He, Y., Hart, A., Mao, M., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, *415*, 530–536.
Waddell, M., Page, D., & Shaughnessy, J., Jr. (2005). Predicting cancer susceptibility from single-nucleotide polymorphism data: A case study in multiple myeloma. *BIOKDD'05: Proceedings of the fifth international workshop on bioinformatics*, Chicago, IL.
Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In *European symposium on principles of kdd* (pp. 78–87). Lecture notes in computer science, Springer, Norway.
Zhang, X., Mesirov, J. P., & Waltz, D. L. (1992). Hybrid system for protein secondary structure prediction. *Journal of Molecular Biology*, *225*, 81–92.
Zou, M., & Conzen, S. D. (2005). A new dynamic Bayesian network approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, *21*, 71–79.

Blog Mining

Blog mining is the application of data mining (in particular, Web mining) techniques on blogs, adapted to the content, format, and language of the medium blog. A *blog* is a (more or less) frequently updated publication on the Web, sorted in (usually reverse) chronological order of the constituent blog posts. As in other areas of the Web, mining is applied to the content of blogs, to the various types of links between blogs, and to blog-related behavior. The latter comprises blog authoring including link setting, blog reading and commenting, and querying (often in blog search engines). For more details on blogs and on mining them, see ►text mining for news and blogs analysis.

Boltzmann Machines

Geoffrey Hinton
University of Toronto, ON, Canada

Synonyms

Boltzmann machines

Definition

A Boltzmann machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off. Boltzmann machines have a simple learning algorithm (Hinton &

Sejnowski, 1983) that allows them to discover interesting features that represent complex regularities in the training data. The learning algorithm is very slow in networks with many layers of feature detectors, but it is fast in "restricted Boltzmann machines" that have a single layer of feature detectors. Many hidden layers can be learned efficiently by composing restricted Boltzmann machines, using the feature activations of one as the training data for the next.

Boltzmann machines are used to solve two quite different computational problems. For a search problem, the weights on the connections are fixed and are used to represent a cost function. The stochastic dynamics of a Boltzmann machine then allow it to sample binary state vectors that have low values of the cost function. For a learning problem, the Boltzmann machine is shown a set of binary data vectors and it must learn to generate these vectors with high probability. To do this, it must find weights on the connections so that relative to other possible binary vectors, the data vectors have low values of the cost function. To solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search problems.

Motivation and Background

The brain is very good at settling on a sensible interpretation of its sensory input within a few hundred milliseconds, and it is also very good, over a much longer timescale, at learning the code that is used to express its interpretations. It achieves both the settling and the learning using spiking neurons which, over a period of a few milliseconds, have a state of 1 or 0. These neurons have intrinsic noise caused by the quantal release of vesicles of neurotransmitter at the synapses between the neurons.

Boltzmann machines were designed to model both the settling and the learning, and were based on two seminal ideas that appeared in 1982. Hopfield (1982) showed that a neural network composed of binary units would settle to a minimum of a simple, quadratic energy function provided that the units were updated asynchronously and the pairwise connections between units were symmetrically weighted. Kirkpatrick et al. (1982) showed that systems that were settling to energy minima could find deeper minima if noise was added to the update rule so that the system could occasionally increase its energy to escape from poor local minima.

Adding noise to a Hopfield net allows it to find deeper minima that represent more probable interpretations of the sensory data. More significantly, by using the right kind of noise, it is possible to make the log probability of finding the system in a particular global configuration be a linear function of its energy. This makes it possible to manipulate log probabilities by manipulating energies, and since energies are simple local functions of the connection weights, this leads to a simple, local learning rule.

Structure of Learning System

The learning procedure for updating the connection weights of a Boltzmann machine is very simple, but to understand why it works it is first necessary to understand how a Boltzmann machine models a probability distribution over a set of binary vectors and how it samples from this distribution.

The stochastic Dynamics of a Boltzmann Machine

When unit i is given the opportunity to update its binary state, it first computes its total input, x_i, which is the sum of its own bias, b_i, and the weights on connections coming from other active units:

$$x_i = b_i + \sum_j s_j w_{ij} \tag{1}$$

where w_{ij} is the weight on the connection between i and j, and s_j is 1 if unit j is on and 0, otherwise. Unit i then turns on with a probability given by the logistic function:

$$\text{prob}(s_i = 1) = \frac{1}{1 + e^{-x_i}} \tag{2}$$

If the units are updated sequentially in any order that does not depend on their total inputs, the network will eventually reach a Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, $\mathbf{v}$, is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})} \tag{3}$$

As in Hopfield nets, the energy of state vector $\mathbf{v}$ is defined as

$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij} \tag{4}$$

where $s_i^{\mathbf{v}}$ is the binary state assigned to unit i by state vector $\mathbf{v}$.

If the weights on the connections are chosen so that the energies of state vectors represent the cost of those state vectors, then the stochastic dynamics of a Boltzmann machine can be viewed as a way of escaping from poor local optima while searching for low-cost solutions. The total input to unit i, x_i, represents the difference in energy depending on whether the unit is off or on, and the fact that unit i occasionally turns on even if x_i is negative means that the energy can occasionally increase during the search, thus allowing the search to jump over energy barriers.

The search can be improved by using simulated annealing. This scales down all of the weights and energies by a factor, T, which is analogous to the temperature of a physical system. By reducing T from a large initial value to a small final value, it is possible to benefit from the fast equilibration at high temperatures and still have a final equilibrium distribution that makes low-cost solutions much more probable than high-cost ones. At a temperature of 0, the update rule becomes deterministic and a Boltzmann machine turns into a Hopfield network.

Learning in Boltzmann Machines Without Hidden Units

Given a training set of state vectors (the data), the learning consists of finding weights and biases (the parameters) that make those state vectors good. More specifically, the aim is to find weights and biases that define a Boltzmann distribution in which the training vectors have high probability. By differentiating (3) and using the fact that:

$$\partial E(\mathbf{v})/\partial w_{ij} = -s_i^{\mathbf{v}} s_j^{\mathbf{v}} \tag{5}$$

it can be shown that:

$$\left\langle \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} \right\rangle_{\text{data}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}} \tag{6}$$

where $\langle \cdot \rangle_{\text{data}}$ is an expected value in the data distribution and $\langle \cdot \rangle_{\text{model}}$ is an expected value when the Boltzmann machine samples state vectors from its equilibrium distribution at a temperature of 1. To perform gradient ascent in the log probability that the Boltzmann machine would generate the observed data when sampling from its equilibrium distribution, w_{ij} is incremented by a small learning rate times the RHS of (6). The learning rule for the bias, b_i, is the same as (6), but with s_j omitted.

If the observed data specifies a binary state for every unit in the Boltzmann machine, the learning problem is convex: There are no nonglobal optima in the parameter space. However, sampling from $\langle \cdot \rangle_{\text{model}}$ may involve overcoming energy barriers in the binary state space.

Learning with Hidden Units

Learning becomes much more interesting if the Boltzmann machine consists of some "visible" units whose states can be observed, and some "hidden" units whose states are not specified by the observed data. The hidden units act as latent variables (features) that allow the Boltzmann machine to model distributions over visible state vectors that cannot be modeled by direct pairwise interactions between the visible units. A surprising property of Boltzmann machines is that, even with hidden units, the learning rule remains unchanged. This makes it possible to learn binary features that capture higher-order structure in the data. With hidden units, the expectation $\langle s_i s_j \rangle_{\text{data}}$ is the average, over all data vectors, of the expected value of $s_i s_j$ when a data vector is clamped on the visible units and the hidden units are repeatedly updated until they reach equilibrium with the clamped data vector.

It is surprising that the learning rule is so simple because $\partial \log P(\mathbf{v})/\partial w_{ij}$ depends on all the other weights in the network. Fortunately, the locally available difference in the two correlations in (6) tells w_{ij} everything it needs to know about the other weights. This makes it unnecessary to explicitly propagate error derivatives, as in the backpropagation algorithm.

Different Types of Boltzmann Machine

The stochastic dynamics and the learning rule can accommodate more complicated energy functions (Sejnowski, 1986). For example, the quadratic energy function in (4) can be replaced by an energy function

that has typical term $s_i s_j s_k w_{ijk}$. The total input to unit i that is used in the update rule must then be replaced by

$$x_i = b_i + \sum_{j<k} s_j s_k w_{ijk}. \quad (7)$$

The only change in the learning rule is that $s_i s_j$ is replaced by $s_i s_j s_k$.

Boltzmann machines model the distribution of the data vectors, but there is a simple extension, the "conditional Boltzmann machine" for modeling conditional distributions (Ackley, Hinton, & Sejnowski, 1985). The only difference between the visible and the hidden units is that, when sampling $\langle s_i s_j \rangle_{\text{data}}$, the visible units are clamped and the hidden units are not. If a subset of the visible units are also clamped when sampling $\langle s_i s_j \rangle_{\text{model}}$ this subset acts as "input" units and the remaining visible units act as "output" units. The same learning rule applies, but now it maximizes the log probabilities of the observed output vectors conditional on the input vectors.

Instead of using units that have stochastic binary states, it is possible to use "mean field" units that have deterministic, real-valued states between 0 and 1, as in an analog Hopfield net. Equation (2) is used to compute an "ideal" value for a unit's state, given the current states of the other units, and the actual value is moved toward the ideal value by some fraction of the difference. If this fraction is small, all the units can be updated in parallel. The same learning rules can be used by simply replacing the stochastic, binary values by the deterministic real values (Peterson & Anderson, 1987), but the learning algorithm is hard to justify and the mean field nets have problems in modeling multimodal distributions.

The binary stochastic units used in Boltzmann machines can be generalized to "softmax" units that have more than two discrete values, Gaussian units whose output is simply their total input plus Gaussian noise, binomial units, Poisson units, and any other type of unit that falls in the exponential family (Welling, Rosen-Zvi, & Hinton, 2005). This family is characterized by the fact that the adjustable parameters have linear effects on the log probabilities. The general form of the gradient required for learning is simply the change in the sufficient statistics caused by clamping data on the visible units.

The speed of Learning

Learning is typically very slow in Boltzmann machines with many hidden layers because large networks can take a long time to approach their equilibrium distribution, especially when the weights are large and the equilibrium distribution is highly multimodal, as it usually is when the visible units are unclamped. Even if samples from the equilibrium distribution can be obtained, the learning signal is very noisy because it is the difference of two sampled expectations. These difficulties can be overcome by restricting the connectivity, simplifying the learning algorithm, and learning one hidden layer at a time.

Restricted Boltzmann Machines

A restricted Boltzmann machine (Smolensky, 1986) consists of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections. With these restrictions, the hidden units are conditionally independent given a visible vector, so unbiased samples from $\langle s_i s_j \rangle_{\text{data}}$ can be obtained in one parallel step. To sample from $\langle s_i s_j \rangle_{\text{model}}$ still requires multiple iterations that alternate between updating all the hidden units in parallel and updating all of the visible units in parallel. However, learning still works well if $\langle s_i s_j \rangle_{\text{model}}$ is replaced by $\langle s_i s_j \rangle_{\text{reconstruction}}$ which is obtained as follows:

1. Starting with a data vector on the visible units, update all of the hidden units in parallel.
2. Update all of the visible units in parallel to get a "reconstruction."
3. Update all of the hidden units again.

This efficient learning procedure approximates gradient descent in a quantity called "contrastive divergence" and works well in practice (Hinton, 2002).

Learning Deep Networks by Composing Restricted Boltzmann Machines

After learning one hidden layer, the activity vectors of the hidden units, when they are being driven by the real data, can be treated as "data" for training another restricted Boltzmann machine. This can be repeated to learn as many hidden layers as desired. After learning multiple hidden layers in this way, the whole network can be viewed as a single, multilayer generative model,

and each additional hidden layer improves a lower bound on the probability that the multilayer model would generate the training data (Hinton, Osindero, & Teh, 2006).

Learning one hidden layer at a time is a very effective way to learn deep neural networks with many hidden layers and millions of weights. Even though the learning is unsupervised, the highest level features are typically much more useful for classification than the raw data vectors. These deep networks can be fine-tuned to be better at classification or dimensionality reduction using the backpropagation algorithm (Hinton & Salakhutdinov, 2006). Alternatively, they can be fine-tuned to be better generative models using a version of the "wake-sleep" algorithm Hinton et al. (2006).

Relationships to Other Models

Boltzmann machines are a type of Markov random field (see ►Graphical Models), but most Markov random fields have simple, local interaction weights which are designed by hand rather than being learned. Boltzmann machines are also like Ising models, but Ising models typically use random or hand-designed interaction weights. The search procedure for Boltzmann machines is an early example of Gibbs sampling, a ►Markov chain Monte Carlo method which was invented independently (Geman & Geman, 1984) and was also inspired by simulated annealing.

Boltzmann machines are a simple type of undirected graphical model. The learning algorithm for Boltzmann machines was the first learning algorithm for undirected graphical models with hidden variables (Jordan, 1998). When restricted Boltzmann machines are composed to learn a deep network, the top two layers of the resulting graphical model form an undirected Boltzmann machine, but the lower layers form a directed acyclic graph with directed connections from higher layers to lower layers, Hinton et al. (2006).

Conditional random fields (Lafferty, McCallum, & Pereira, 2001) can be viewed as simplified versions of higher-order, conditional Boltzmann machines in which the hidden units have been eliminated. This makes the learning problem convex, but removes the ability to learn new features.

Recommended Reading

Ackley, D., Hinton, G., & Sejnowski, T. (1985). A Learning algorithm for boltzmann machines. *Cognitive Science, 9*(1), 147–169.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 6*(6), 721–741.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA, 79*, 2554–2558.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation, 14*(8), 1711–1800.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*, 504–507.

Hinton, G. E., & Sejnowski, T. J. (1983). Optimal perceptual inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition, Washington, DC* (pp. 448–453).

Jordan, M. I. (1998). *Learning in graphical models*. Cambridge, MA MIT press.

Kirkpatrick, S., Gelatt, D. D., & Vecci, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th international conference on machine learning* (pp. 282–289). San Francisco, Morgan Kaufmann.

Peterson, C., & Anderson, J. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems, 1*(5), 995–1019.

Sejnowski, T. J. (1986). Higher-order boltzmann machines. *AIP Conference Proceedings, 151*(1), 398–403.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing: Vol. 1: Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems* (vol. 17, pp. 1481–1488). Cambridge, MA: MIT Press.

Boosting

Boosting is a family of ►ensemble learning methods. The Boosting framework is an answer to a question posed on whether two complexity classes of learning problems are equivalent: *strongly learnable*, and *weakly learnable*. The Boosting framework is a proof by construction that the answer is positive, they are equivalent. The framework allows a "weak" model, only slightly

better than random guessing, to be *boosted* into an arbitrarily accurate *strong* model. ►Adaboost is the most well known and successful of the Boosting family, though there exist many variants specialized for particular tasks, such as cost-sensitive and noise-tolerant versions. See ►ensemble learning for full details.

Bootstrap Sampling

Definition

Bootstrap sampling is a process for creating a distribution of datasets out of a single dataset. It is used in the ►ensemble learning algorithm ►Bagging. It can also be used in ►algorithm evaluation to create a distribution of training sets from which to estimate properties of an algorithm.

Recommended Reading

Davison, A. C., & Hinkley, D. (2006). *Bootstrap methods and their applications* (8th ed.). Cambridge: Cambridge Series in Statistical and Probabilistic Mathematics.

Bottom Clause

Synonyms

Saturation; Starting clause

Definition

The bottom clause is a notion from the field of ►inductive logic programming. It is used to refer to the most specific hypothesis covering a particular example when ►learning from entailment. When learning from entailment, a hypothesis H covers an example e relative to the background theory B if and only if $B \wedge H \models e$, that is, B together with H ►entails the example e. The bottom clause is now the most specific clause satisfying this relationship w.r.t the background theory B and a given example e.

For instance, given the background theory B

```
bird :- blackbird.
bird :- ostrich.
```

and the example e:

```
flies :- blackbird, normal.
```

the bottom clause is H

```
flies :- bird, blackbird, normal.
```

The bottom clause can be used to constrain the search for clauses covering the given example because all clauses covering the example relative to the background theory should be more general than the bottom clause. The bottom clause can be computed using ►inverse entailment.

Cross References

►Entailment
►Inductive Logic Programming
►Inverse Entailment
►Logic of Generality

Bounded Differences Inequality

►McDiarmid's Inequality

BP

►Backpropagation

Breakeven Point

More accurately described as *precision–recall* BEP, it is an evaluation measure originally introduced in the field of information retrieval to evaluate retrieval systems that return a list of documents ordered by their supposed relevance to the user's information need (see also ►Document Classification). It can also be used to evaluate any classification model f that addresses a two-class classification problem but outputs real-valued predictions $f(x)$ instead of binary ones. To use such a classifier in practice, one would select a threshold θ and predict an instance x to be positive if $f(x) > \theta$ and negative otherwise. Thus, the ►precision and ►recall of this system depend on the choice of the threshold θ. A lower threshold means higher recall, but usually also lower precision. At some point (when the number of instances predicted to be positive is the same as the actual number

of positive instances), precision and recall are equal; this value of precision and recall is known as the *precision–recall BEP*. It is a useful measure of the quality of our classifier because it gives us guidance into what sort of tradeoffs are available to the user of such a classifier via the choice of threshold: if we want a precision above the BEP, we must accept that our recall will be below the BEP, and vice versa. A different meaning of the term "breakeven point" is sometimes used in ROC (►ROC Analysis), where the *ROC breakeven* is defined as the point where the true positive rate and the false positive rate sum to 1; smaller values of the ROC breakeven are better than larger ones. Informally, the ROC breakeven measures how close the ROC curve gets to the "ROC sweet spot" in the top left corner (where the ►true positive rate is 1 and the ►false positive rate is 0).