L

L1-Distance

►Manhattan Distance

Label

A label is a target value that is associated with each ►object in ►training data. In ►classification learning, labels are ►classes. In ►regression, labels are numeric.

Labeled Data

Labeled data are ►data for which each ►object has an identified target value, the ►label. Labeled data are used in ►supervised learning. They stand in contrast to *unlabeled data* that are used in ►unsupervised learning.

Language Bias

Definition

A learner's language bias is the set of hypotheses that can be expressed using the hypothesis language employed by the learner.

This language bias can be implicit, or it can be defined explicitly, using a bias specification language (see ►Bias Specification Language).

Cross References

►Learning as Search

Laplace Estimate

►Rule Learning

Latent Class Model

►Mixture Model

Latent Factor Models and Matrix Factorizations

Definition

Latent Factor models are a state of the art methodology for model-based ►collaborative filtering. The basic assumption is that there exist an unknown low-dimensional representation of users and items where user-item affinity can be modeled accurately. For example, the rating that a user gives to a movie might be assumed to depend on few implicit factors such as the user's taste across various movie genres. Matrix factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations to the user-item matrix, where weights are used to hold out missing entries. There is a large family of matrix factorization models based on choice of loss function to measure approximation quality, regularization terms to avoid overfitting, and other domain-dependent formulations.

Lazy Learning

Geoffrey I. Webb
Monash University, Victoria, Australia

Definition

The computation undertaken by a learning system can be viewed as occurring at two distinct times, ►training time and ►consultation time. Consultation time is the time between when an ►object is presented to a system for an inference to be made and the time when the

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

inference is completed. Training time is the time prior to consultation time during which the system makes inferences from training data in preparation for consultation time. *Lazy learning* refers to any machine learning process that defers the majority of computation to consultation time. Two typical examples of lazy learning are ►instance-based learning and ►Lazy Bayesian Rules. Lazy learning stands in contrast to ►eager learning in which the majority of computation occurs at training time.

Discussion

Lazy learning can be computationally advantageous when predictions using a single ►training set will only be made for few objects. This is because only the immediate sections of the instance space that are occupied by objects to be classified need be modeled. In consequence, no computation is expended in the modeling areas of the instance space that are irrelevant to the predictions that need to be made. This can also be an advantage when a training set is frequently updated, as can be the case in ►online learning, as only the applicable portions of each model are created.

Lazy learning can help improve prediction ►accuracy by allowing a system to concentrate on deriving the best possible decision for the exact points of the instance space for which predictions are to be made. In contrast, eager learning can sometimes result in suboptimal predictions for some specific parts of the instance space as a result of trade-offs during the process of deriving a single model that seeks to minimize average error over the entire instance space.

Cross References

►Eager Learning
►Instance-Based Learning
►Locally Weighted Regression for Control
►Online Learning

Learning as Search

Claude Sammut
The University of New South Wales, Sydney NSW, Australia

Definition

Learning can be viewed as a search through the space of all sentences in a concept description language for a sentence that best describes the data. Alternatively, it can be viewed as a search through all hypotheses in a ►hypothesis space. In either case, a generality relation usually determines the structure of the search space.

Background

The input to a learning program consists of descriptions of objects from the universe (the ►training set) and, in the case of ►supervised learning, an output value associated with the example. A program is limited in the concepts that it can learn by the representational capabilities of both the ►observation language (i.e., the language used to describe the training examples) and ►hypothesis language (the language used to describe the concept). For example, if an attribute/value list is used to represent examples for an induction program, the measurement of certain attributes and not others places limits on the kinds of patterns that the learner can find. The learner is said to be *biased* by its observation language. The hypothesis language also places constraints on what may and may not be learned. For example, in the language of attributes and values, relationships between objects are difficult to represent. Whereas, a more expressive language, such as first-order logic, can easily be used to describe relationships. These biases are collectively referred to as *representation bias*.

Representational power comes at a price. Learning can be viewed as a search through the space of all sentences in a language for a sentence that best describes the data. The richer the language, the larger the search space. When the search space is small, it is possible to use "brute force" search methods. If the search space is very large, additional knowledge is required to reduce the search. Notions of generality and specificity are important for ordering the search (see ►Generalization and ►Specialization).

Representation

The representation of instances and concepts affects the way a learning system searches for concept representations.

The input to a learning program may take many forms, for example, records in a database, pages of text, images, audio, and other signals of continuous data. Very often, the raw data are transformed into *feature vectors* or *attribute/value lists*. The values of the attributes or features may be continuous or discrete.

These representation by attribute/value lists is the observation language.

The representation of the concept varies considerably, depending on the approach taken for learning. In ►instance-based learning, concepts are represented by a set of prototypical instances of the concept, so abstract representations are not constructed at all. This kind of representation is said to be *extensional*. Instance-based learning is also called ►lazy learning because the learner does little work at the time that training instances are presented. Rather, at classification time, the system must find the most similar instances to the new example. See Fig. 1.

When instances are represented as feature vectors, we can treat each feature or attribute as one dimension in a multi-dimensional space. The supervised learning problem can then be characterized as the problem of finding a surface that separates objects that belong to different classes into different regions. In the case of unsupervised learning, the problem becomes one of the finding clusters of instances in the multi-dimensional space.

Learning methods differ in the way they represent and create the discrimination surfaces. In *function approximation*, the learner searches for functions that describes the surface (Fig. 2). Function approximation methods can often produce accurate classifiers because they are capable of construction complex decision surfaces. However, the concept description is stored as a set of coefficients. Thus, the results of learning are not easily available for inspection by a human reader.

Rather than searching for discriminant functions, symbolic learning systems find expressions equivalent to sentences in some form of logic. For example, we may distinguish objects according to two attributes: size and color. We may say that an object belongs to class 3 if its color is red and its size is very small to medium. Following the notation of Michalski (1983), the classes in Fig. 3 may be written as:

$$class1 \leftarrow size = large \wedge color \in \{red, orange\}$$

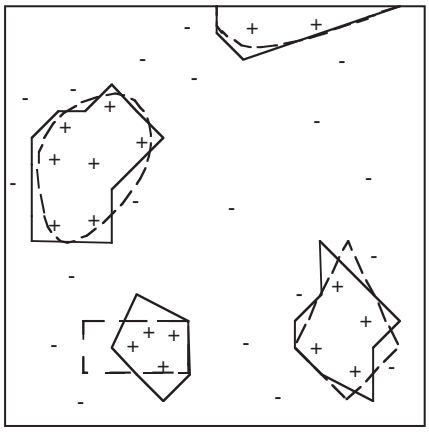
$$class2 \leftarrow size \in \{small, medium\} \wedge color \in \{oragne, yellow\}$$

$$class3 \leftarrow size \in \{v\_small \ldots medium\} \wedge color = blue$$

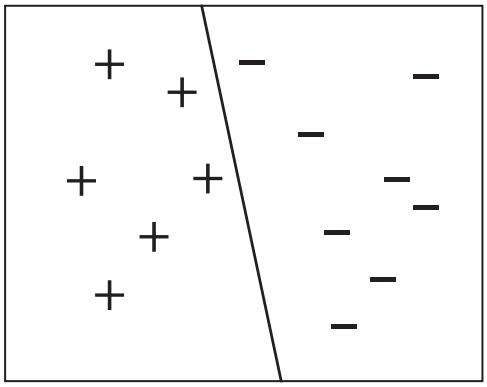
Note that this kind of description partitions the universe with axis-orthogonal surfaces, unlike the function approximation methods that find smooth surfaces to discriminate classes (Fig. 4).

Useful insights into induction can be gained by visualizing it as searching for a discrimination surface in a multi-dimensional space. However, there are limits to this geometric interpretation of learning. If we wish to learn concepts that describe complex objects and relationships between the objects, it is often useful to rely on reasoning about the concept description language itself.

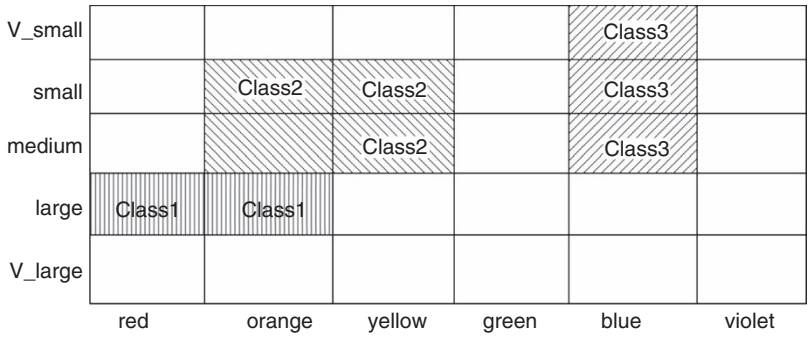
As we saw, the concepts in Fig. 3 can be expressed as clauses in propositional logic. We can establish a correspondence between sentences in the concept des-



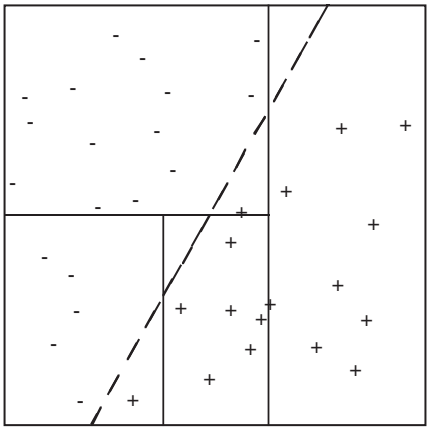
Learning as Search. Figure 1. The extension of an Instance-Based Learning concept is shown in *solid lines*. The *dashed lines* represent the target concept. A sample of positive and negative examples is shown Adapted from Aha, Kibler and Albert (1991)



Learning as Search. Figure 2. A linear discrimination between two classes



Learning as Search. Figure 3. Discrimination on attributes and values



Learning as Search. Figure 4. The dashed line shows the real division of objects in the universe. The solid lines show a decision tree approximation

cription language (the hypothesis language) and a diagrammatic representation of the concept. More importantly, we can create a correspondence between generalization and specialization operations on the sets of objects and generalization and specialization operations on the sentences of the language.

Once we have established the correspondence between sets of objects and their descriptions, it is often convenient to forget about the objects and only consider that we are working with expressions in a language. For example, the clause

$$class2 \leftarrow size = large \wedge color = red \qquad (1)$$

can be generalized to

$$class1 \leftarrow size = large \qquad (2)$$

by dropping one of the conditions. Thus, we can view learning as search through a generalization lattice that is created by applying different syntactic transformations on sentences in the hypothesis language.

Version Spaces and Subsumption

Mitchell (1977, 1982) defines the *version space* for a learning algorithm as the subset of hypotheses consistent with the training examples. That is, the hypothesis language is capable of describing a large, possibly infinite, number of concepts. When searching for the target concept, we are only interested in the subset of sentences in the hypothesis language that are consistent with the training examples, where consistent means that the examples are correctly classified. We can used the *generality* of concepts to help us limit our search to only those hypotheses in the version space.

In the above example, we stated that clause (2) is more general than clause (1). In doing so, we assumed that there is a general-to-specific ordering on the sentences in the hypothesis language. We can formalize the generality relation as follows. A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell, 1997). That is, $h_1 \geq h_2$ if and only if

$$(\forall x)[h_1(x) \rightarrow h_2(x)]$$

A hypothesis, h_1, is strictly more general than h_2, if $h_1 \geq h_2$ and $h_2 \not\leq h_1$.

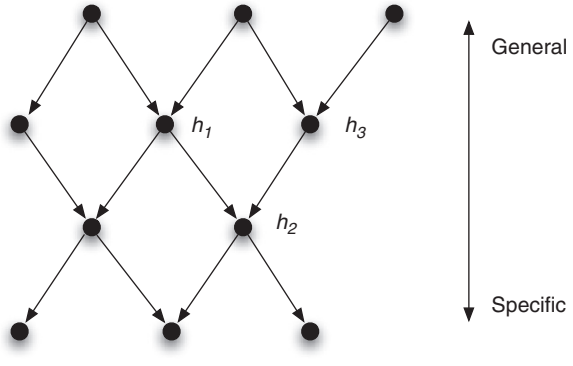
Note that the *more general than* ordering is strongly related to *subsumption* (see ▶subsumption and the

►Logic of Generality). Where the above definition of the generality relation is given in terms of the cover of a hypothesis, subsumption defines a generality ordering on expressions in the hypothesis language.

Learning algorithms can use the *more general than* relation to order their search for the best hypothesis. Because generalizations and specializations may not be unique, this relation forms a lattice over the sentences in the hypothesis language, as illustrated in Fig. 5. A search may start from the set of most specific hypotheses that fit the training data and perform a *specific-to-general* search or it may start from the set of most general hypotheses and perform a *general-to-specific* search. The search algorithm may also be bidirectional, combining both.

In Fig. 5, each node represents a hypothesis. The learning algorithm searches this lattice in an attempt to find the hypothesis that best fits the training data. Like searching in any domain, the algorithm may keep track of one node at a time, as in *depth first* or *best first* searches, or it may create a frontier of nodes as in *breadth first* or *beam searches*.

Suppose we have single-hypothesis search. A specific-to-general search may begin by randomly selecting a positive training example and creating a hypothesis that the target concept is exactly that example. Each time a new positive example is seen that is not covered by the hypothesis, the hypothesis must be *generalized*. That is, a new hypothesis is constructed that is general enough to cover all the examples covered by the previous hypothesis, as well as covering the new example. If the algorithm sees a negative example that is incorrectly covered by the current hypothesis, then the hypothesis must be *specialized*. That is, a new hypothesis is construct that is more specific than the current hypothesis such that all the positive examples that were previously covered are still covered by the new negative example is excluded.



Learning as Search. Figure 5. Generalization lattice

A similar method can be used for a general-to-specific search. In this case, the initial hypothesis is that the target concept covers every object in the universe. In both cases, the algorithm must choose how to construct either generalizations or specializations. That is, a method is needed to choose which nodes in the search to expand next. Here, the ►least general generalization (Plotkin, 1970) or the ►most general specialization are useful. These define the smallest steps that can be taken in expanding the search. For example, in Fig. 5, h_2 is the minimal specialization that can be made from h_1 or h_3 in a general-to-specific search that starts from the top of the lattice. Similarly, h_1 and h_3 are the least general generalizations of h_2. A search for the target concept can begin with an initial hypothesis and make minimal generalizations or specializations in expanding the next node in the search.

Rather than maintaining on a single current hypothesis, a search strategy may keep a set of candidate hypotheses. For example, a breadth first search generalizing from hypothesis h_2 will create a frontier for the search that is the set $\{h_1, h_3\}$. When there are many ways in which an hypothesis can be generalized or specialized, the size of the frontier set may be large. In algorithms such as Aq (Michalski, 1983) and CN2 (Clark and Niblett, 1989), a *beam search* is used. Rather than storing all possible hypotheses, the n best are kept are stored, where "best" can be defined in several ways. One metric for comparing hypotheses is given by

$$\frac{P_c + N_{\bar{c}}}{P + N}$$

where P and N are the number of positive and negative instances, respectively; P_c is the number of positive instances covered by the hypothesis; and $N_{\bar{c}}$ is the number of negative instances not covered.

Mitchell's (1997) candidate-elimination algorithm performs a bidirectional search in the hypothesis space. It maintains a set, S, of most specific hypotheses that are consistent with the training data and a set, G, of most general hypotheses consistent with the training data. These two sets form two boundaries on the version space. As new training examples are seen, the

Algorithm 1. The candidate-elimination algorithm, after Mitchell (1997)

Initialize *G* to the set of maximally general hypotheses in the hypothesis space
Initialize *S* to the maximally specific hypotheses in the hypothesis space
For each training example, *d*,
  if *d* is a positive example
    remove from *G* any hypothesis inconsistent with *d*
    For each hypothesis, *s*, in *S* that is not consistent with *d*
      remove *s* from *S*
        add all minimal generalizations, *h*, of *s* such that
          h is consistent with *d* and some member of *G* is more general than *h*
      remove from *S* any hypothesis that is more general than another hypothesis in *S*
  if *d* is a negative example
    remove from *S* any hypothesis inconsistent with *d*
    For each hypothesis, *g*, in *G* that is not consistent with *d*
      remove *g* from *G*
        add all minimal specializations, *h*, of *g* such that
          h is consistent with *d* and some member of *S* is more general than *h*
      remove from *G* any hypothesis that is less general than another hypothesis in *G*

boundaries are generalized or specialized to maintain consistency. If a new positive example is not covered by a hypothesis in *S*, then it must be generalized. If a new negative example is not rejected by an hypotheses in *G*, then it must be specialized. Any hypothesis in *G* not consistent with a positive example is removed and any hypothesis in *S* not consistent with a negative example is also removed. See Algorithm 1.

Noisy Data

Up to this point, we have assumed that the training data are free of noise. That is, all the examples are correctly classified and all the attribute values are correct. Once we relax this assumption, the algorithms described above must be modified to use approximate measures of consistency. The danger presented by noisy data is that the learning algorithm will *over fit* the training data by creating concept descriptions that try to cover the bad data as well as the good. For methods to handle noisy data see the entries in ►pruning.

Several standard texts give good introductions to search in learning, including Langley (1996), Mitchell (1997), Bratko (2000), Russell and Norvig (2009).

Cross References

►Decision Tree Learning
►Generalization
►Induction
►Instance-Based Learning
►Logic of Generality
►Rule Learning
►Subsumption

Recommended Reading

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66.

Bratko, I. (2000). *Prolog programming for artificial intelligence* (3rd ed.). Boston, MA: Addison-Wesley.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*(4), 261–283.

Langley, P. (1996). *Elements of machine learning*. San Mateo: Morgan Kaufmann.

Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule-learning (pp. 305–310). In *Proceedings of the fifth international joint conference on artificial intelligence*, Cambridge.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*(2), 203–226.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh: Edinburgh University Press.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Englewood cliffs, WJ: Prentice Hall.

Learning Bayesian Networks

►Learning Graphical Models

Learning Bias

►Inductive Bias

Learning By Demonstration

►Behavioral Cloning

Learning By Imitation

►Behavioral Cloning

Learning Classifier Systems

►Classifier Systems

Learning Control

Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. Learning control is usually distinguished from adaptive control in that the learning system is permitted to fail during the process of learning. In contrast, adaptive control emphasizes single trial convergence without failure. Thus, learning control resembles the way that humans and animals acquire new movement strategies, while adaptive control is a special case of learning control that fulfills stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots. In general, the control system can be any system that changes its state in response to a control signal, e.g., a web page with a hyperlink, a car, or a robot.

Learning Control Rules

►Behavioral Cloning

Learning Curves in Machine Learning

Claudia Perlich
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Synonyms

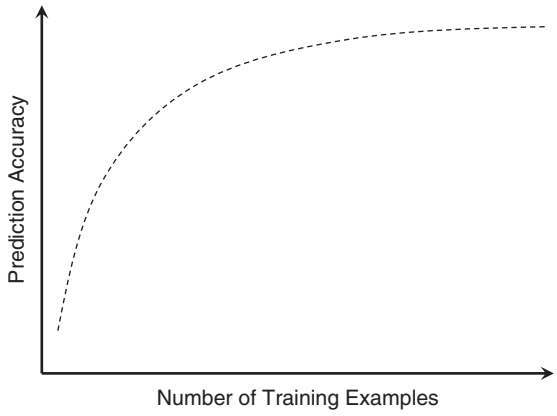
Error curve; Experience curve; Improvement curve; Training curve

Definition

A learning curve shows a measure of predictive performance on a given domain as a function of some measure of varying amounts of learning effort. The most common form of learning curves in the general field of machine learning shows predictive accuracy on the test examples as a function of the number of training examples as in Fig. 1.

Motivation and Background

Learning curves were initially introduced in educational and behavioral/cognitive psychology. The first person to describe the learning curve was Hermann Ebbinghaus in 1885 (Wozniak, 1999). He found that the time required to memorize a nonsense syllable increased sharply as the number of syllables increased. Wright (1936) described the effect of learning on labor



Learning Curves in Machine Learning. Figure 1. Stylized learning curve showing the model accuracy on test examples as a function of the number of training examples

productivity in the aircraft industry and proposed a mathematical model of the learning curve. Over time, the term has acquired related interpretation in many different fields including the above definition in machine learning and statistics.

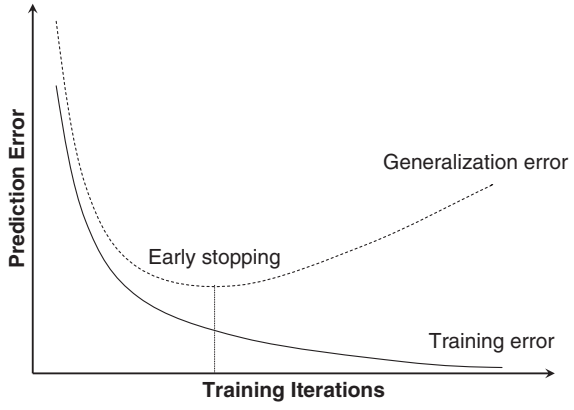
Use of Learning Curves in Machine Learning

In the area of machine learning, the term "learning curve" is used in two different contexts, the main difference being the variable on the *x*-axis of the curve.

- The ▶artificial neural network (ANN) literature has used the term to show the diverging behavior of in and out-of-sample performance as a function of the *number of training iterations* for a given number of training examples. Figure 2 shows this stylized effect.
- General machine learning uses learning curves to show the predictive ▶generalization performance as a function of the *number of training examples*. Both the graphs in Fig. 3 are examples of such learning curves.

Artificial Neural Networks

The origins of ANNs are heavily inspired by the social sciences and the goal of recreating the learning behavior of the brain. The original model of the "perceptron" mirrored closely the biological foundations of neural sciences. It is likely that the notion of learning curves was to some extent carried over from the social sciences of human learning into the field of ANNs. It shows the model error as a function of the training time measured in terms of the number of iterations. One iteration denotes in the context of neural network learning one single pass over the training data and the corresponding update of the network parameters (also called weights). The algorithm uses gradient descent minimizing the model error on the training data.



Learning Curves in Machine Learning. Figure 2. Learning curve for an artificial neural network

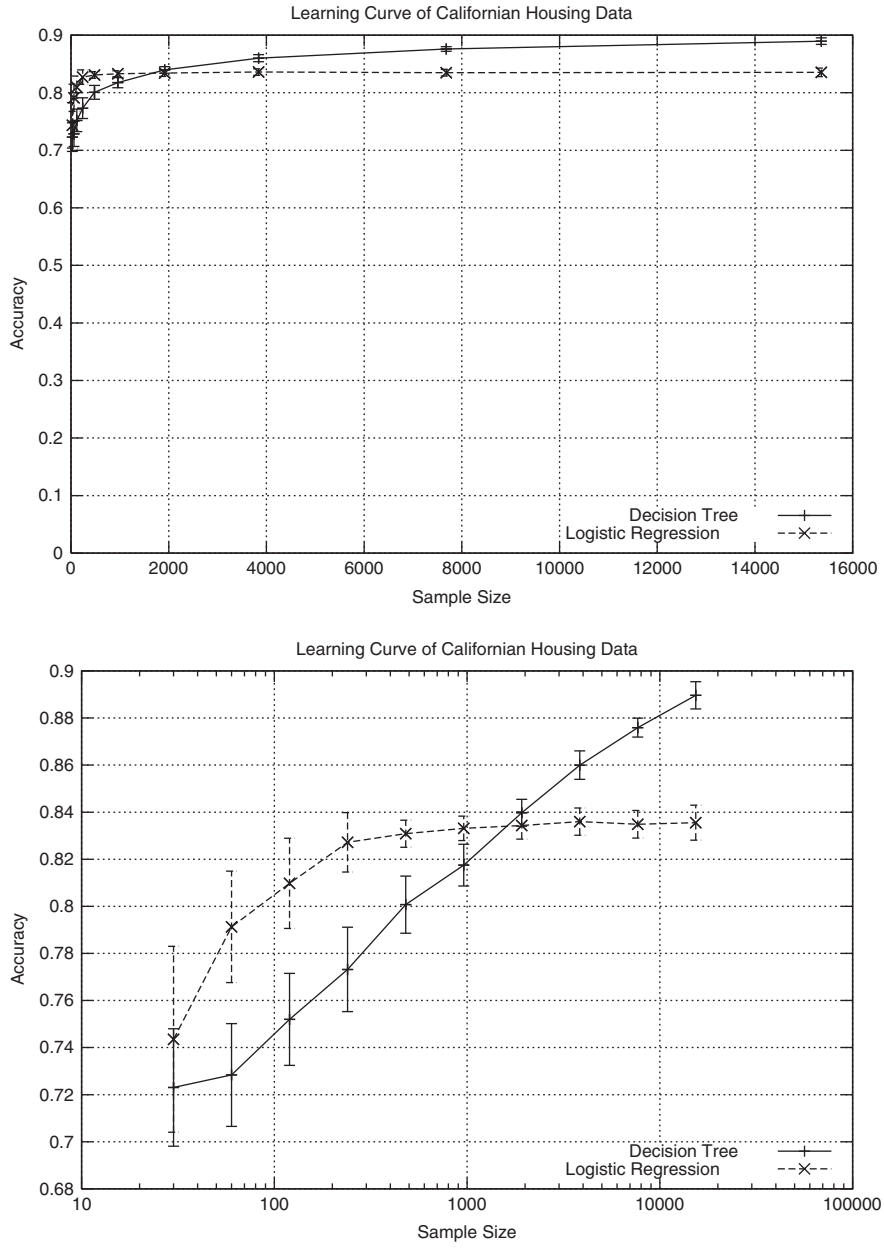
The learning curve in Fig. 2 shows the stylized effect of the relative training and generalization error on a test set as a function of the number of iterations. After initial decrease of both types of error, the generalization error reaches a minimum and starts to increase again while the training error continues to decrease.

This effect of increasing generalization error is closely related to the more general machine learning issue of ▶overfitting and variance error for models with high expressive power (or capacity). One of the initial solutions to this problem for neural networks was early stopping - some form of early regularization technique that picked the model at the minimum of the error curve on a validation subset of the data that was not used for training.

General Machine Learning

In the more general machine learning setting and statistics (Flury & Schmid, 1994), learning curves represent the generalization performance of the model as a function of the size of the training set.

Figure 3 was taken from Perlich, Provost, and Simonoff (2003) and shows two typical learning curves for two different modeling algorithms (▶decision tree and ▶logistic regression) on a fairly large domain. For smaller training-set sizes the curves are steep, but the increase in accuracy lessens for larger training-set sizes. Often for very large training-set sizes the standard representation in the upper graph obscures small, but non-trivial, gains. Therefore, to visualize the curves it is often useful to use a log scale on the horizontal axis and start the graph at the accuracy of the smallest training-set size (rather than at zero). In addition, one can include error bars that capture the estimated variance of the error over multiple experiments and provide some



Learning Curves in Machine Learning. Figure 3. Typical learning curves in original and log scale

impression of the relevance of the differences between two learning curves as shown in the graphs.

The figure also highlights a very important issue in comparative analysis of different modeling techniques: learning curves for the same domain and different models can cross. This implies an important pitfall as pointed out by Kibler and Langley (1998): "Typical empirical papers report results on training sets of fixed size, which tells one nothing about how the methods would fare given more or less data, rather than collecting learning curves …". A corollary on the above observation is the dangers of selecting an algorithm on a smaller subset of

the ultimately available training data either in the context of a proof of concept pre-study or some form of cross-validation.

Aside from its empirical relevance there has been significant theoretical work on learning curves - notably by Cortes, Jackel, Solla, Vapnik, and Denker (1994). They are addressing the question of predicting the expected generalization error from the training error of a model. Their analysis provides many additional insights about the generalization performance of different models as a function of not only training size but in addition the model capacity.

Cross References

►Artificial Neural Networks
►Computational Learning Theory
►Decision Tree
►Generalization Performance
►Logistic Regression
►Overfitting

Recommended Reading

Cortes, C., Jackel, L. D., Solla, S. A., Vapnik, V., & Denker, J. S. (1994). Learning curves: Asymptotic values and rate of convergence. *Advances in Neural Information Processing Systems, 6*, 327–334.

Flury, B. W., & Schmid, M. J. (1994). Error rates in quadratic discrimination with constraints on the covariance matrices. *Journal of Classification, 11*, 101–120.

Kibler, D., & Langley, P. (1988). Machine learning as an experimental science. In *Proceedings of the third European working session on learning*, Pittman, Glasgow (pp. 81–92). Hingham, MA: Kluwer Academic Publishers.

Perlich, C., Provost, F., & Simonoff, J. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *Journal of Machine Learning Research, 4*, 211–255.

Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning, 6*, 111–143.

Wozniak, R. H. (1999). Introduction to memory: Hermann Ebbinghaus (1885/1913). In *Classics in the history of psychology*. Bristol, UK: Thoemmes Press.

Wright, T. P. (1936). Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences, 3*(4), 122–128.

Learning from Complex Data

►Learning from Structured Data

Learning from Labeled and Unlabeled Data

►Semi-Supervised Learning

Learning from Nonpropositional Data

►Learning from Structured Data

Learning from Nonvectorial Data

►Learning from Structured Data

Learning from Preferences

►Preference Learning

Learning from Structured Data

Tamás Horváth, Stefan Wrobel
University of Bonn, Sankt Augustin, Germany

Synonyms

Learning from complex data; Learning from nonpropositional data; Learning from nonvectorial data

Definition

Learning from structured data refers to all those learning tasks where the objects to be considered as inputs and/or outputs can usefully be thought of as possessing internal structure and/or as being interrelated and dependent on each other, thus forming a structured space. Typical instances of data in structured learning tasks are sequences as they arise, e.g., in speech processing or bioinformatics, and trees or general graphs such as syntax trees in natural language processing and document analysis, molecule graphs in chemistry, relationship networks in social analysis, and link graphs in the World Wide Web. Learning from structured data presents special challenges,

since the commonly used feature vector representation and/or the i.i.d. (independently and identically distributed data) assumption are no longer applicable. Different flavors of learning from structured data are represented by (overlapping) areas such as ▶Inductive Logic Programming, ▶Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, ▶graph mining, and ▶collective classification.

Motivation and Background

For a long time, learning algorithms had almost exclusively considered data represented in rectangular tables defined by a fixed set of columns and a number of rows corresponding to the number of objects to be described. In this representation, each row independently and completely describes one object, each column containing the value of one particular property or feature of the object. Correspondingly, this representation is also known as feature vector representation, propositional representation, or vectorial data representation. Statistically, in such a representation, the values in each row (i.e., the objects) are assumed to be drawn i.i.d. from a fixed (but unknown) distribution.

However, when working with objects that are interrelated and/or have internal structure, this representation is no longer adequate. Consider representing chemical molecules with varying numbers of atoms and bonds in a table with a fixed number of columns. If we wanted each molecule to correspond to one row, we would have to fit the atoms and bonds into the columns, e.g., by reserving a certain number of columns for each one of them and their respective properties. To do that however, we would have to make the table wide enough to contain the largest possible molecule, resulting in many empty columns for smaller molecules, and by mapping the component atoms and bonds to columns, we would assign an order to them that would not be justified by the underlying problem and that would consequently mislead any feature vector learning algorithm.

The second issue with structured data arises from objects that are interrelated. Consider, e.g., the task of speech recognition, i.e., learning to map an acoustic unit into the corresponding lexical unit. Clearly, to solve this task, one must consider the sequence of such units, since both on the input and the output sides the probability of observing a particular unit will strongly depend on the preceding or subsequent units. The same is true, e.g., in classifying pages in the World Wide Web, where it is quite likely that the classification of the page will correlate with the classifications of neighboring pages. Therefore, any learning algorithm that would regard acoustic units or pages as independent and identically distributed objects is destined to fail, since for a successful solution the interdependencies must be modeled and exploited.

In machine learning, even though there has been interest in structured representation from the very beginning of the 1970s (cf. the systems Arch (Winston, 1975) or INDUCE (Michalski, 1983)), it was only in the 1990s, triggered by the popularity of logic programming and Horn clause representation, that learning from structured data was more intensively considered for logical representations in the subfield of Inductive Logic Programming. Outside of (what was then) machine learning, due to important applications such as speech processing, probabilistic models for sequence data such as ▶Hidden Markov Models have been considered much earlier. Toward the end of the 1990s, given an enormous surge of interest in applications in bioinformatics and the World Wide Web, and technical advances resulting from the integration of probabilistic and statistical approaches into machine learning (e.g., ▶Graphical Models and ▶kernel methods), work on learning from structured data has taken off and now represents a significant part of machine learning research in overlapping subareas such as Inductive Logic Programming, Statistical Relational Learning, probabilistic relational and logical learning, learning with structured outputs, sequence learning, learning with trees and graphs, graph mining, and collective inference.

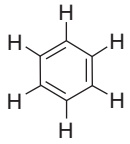
Main Tasks and Solution Approaches

A particular problem setting for learning from structured data is given by specifying, among others, (1) the language representing the input and output of the learning algorithms, (2) the type of the input and/or output data, and (3) the learning task.

1. Beyond attribute-value representation, the most intensively investigated representation languages

used in learning are ►First-Order Logic, in particular, the fragment of first-order Horn clauses, and labeled graphs. Although labeled graphs can be considered as special relational structures and thus form a special fragment of first-order logic, these two representation languages are handled separately in machine learning. As an example of first-order representation of labeled graphs, the molecular graph of a benzene ring can be represented as follows:

atom(a_1,carbon).,…,atom(a_6,carbon).,
atom(a_7,hydrogen).,…,atom(a_{12},hydrogen).,
edge(a_1,a_2,aromatic).,…,edge(a_6,a_1,aromatic).,
edge(a_1,a_7,single).,…,edge(a_6,a_{12},single).,
edge(X,Y) ← edge(Y,X).



The molecular graph of benzene rings (carbon atoms are unmarked)

Besides complexity reasons, the above two representation languages are motivated also by the difference in the matching operators typically used for these two representations. While in case of first-order logic, the matching operator is defined by logical implication or by relational homomorphism (often referred to as subsumption), which is a decidable, but thus, incomplete variant of logical implication, in case of labeled graphs it is defined by subgraph isomorphism (i.e., by injective homomorphism).

2. Another component defining a task for learning from structured data is the type of the input and/or output data (see ►Observation Language and ►Hypothesis Language). For the input, two main types can be distinguished: the instances are disjoint structures (structured instances) or substructures of some global structure (structured instance space). Molecular graphs formed by the atom-bond structure of chemical compounds are a common example of structured instances. For structured instance spaces, the web graph provides an example of a global structure; for this case, the set of instances corresponds to the set of vertices formed by the web sites. The primary goal of traditional discriminative learning is to approximate unknown target functions mapping the underlying instance space to some subset of the set of real numbers. In some of the applications, however, the elements of the range of the target function must also be structured. Such problems are referred to as learning in structured output spaces. As an example of structured output, we mention the protein secondary structure prediction problem, where the goal is to approximate the function mapping the primary structures of proteins to their secondary structures. Notice that primary and secondary structures can be represented by strings, which in turn can be considered as labeled directed paths.
3. Finally, the third component defining a problem setting is the learning task. Besides the classical learning tasks (e.g., supervised, semisupervised, unsupervised, transductive learning etc.), recent tasks include new problems such as, e.g., learning preferences (i.e., a directed graph, where an edge from vertex u to vertex v denotes that v is preferred to u), learning rankings (i.e., when the target preference relation must be a total order), etc.

Several classes of algorithms have been developed for the problem settings defined by the above components. ►Propositionalization techniques (e.g., as in LINUS (Lavrac et al., 1991)) first transform the structured data into a single table of fixed width by extracting a large number of propositional features and then use some propositional learner.

Non-propositionalization rule-based approaches follow mainly general-to-specific (top–down) or specific-to-general (bottom-up) search strategies. For top–down search (e.g., as in FOIL (Quinlan, 1990)), the crucial step of the algorithms is the definition of the refinement operators. While for graph structured data the specialization relation on the hypothesis space is usually defined by subgraph isomorphism and is therefore a partial order, for First-Order Logic it is typically defined by subsumption and is therefore only a preorder (i.e., antisymmetry does not hold), leading to undesirable algorithmic properties (e.g., incompleteness). For bottom–up search (e.g., as in GOLEM (Muggleton &

Feng, 1992)), which is less common for graph structured data, the generalization of hypotheses is usually defined by some variant of Plotkin's ►Least General Generalization operator for first-order clauses. While this generalization operator has nice algebraic properties, its application raises severe complexity issues, as the size of the hypotheses may exponentially grow in the number of examples.

Recent research in structural learning has been focusing very strongly on distance- and kernel-based approaches which in terms of accuracy have often turned out superior to rule-based approaches (e.g., in virtual screening of molecules). In such approaches, the basic algorithms carry over unchanged from the propositional case; instead, special distance (e.g., as in RIBL (Emde & Wettschereck, 1996)) or kernel functions for structural data are developed. Since even for graphs, computing any complete kernel (i.e., for which the underlying embedding function into the feature space is injective) is at least as hard as the graph isomorphism problem, most practical and efficient kernels are based on examining the structure for the occurrence of simpler parts (e.g., trees, walks, paths, and cycles) which are then counted and effectively used as feature vectors in an intersection kernel.

Finally, as a recent class of approaches, we also mention Statistical Relational Learning which extends probabilistic Graphical Models (e.g., Bayesian networks or Markov networks) with relational and logic elements (e.g., as in Alchemy (Domingos & Richardson, 2007), ICL (Poole, 2008), PRISM (Sato & Kameya, 2008)).

Applications

Virtual compound screening is a representative application example of learning from structured data. This computational problem in pharmaceutical research is concerned with the identification of chemical compounds that can be developed into drug candidates. Since current pharmaceutical compound repositories contain millions of molecules, the design of efficient algorithms for virtual compound screening has become an integral part of computer-aided drug design. One of the branches of the learning algorithms concerned with this prediction problem is based on using the compounds' 2D graph structures formed by their atoms and bonds. Depending on the representation of chemical graphs, this branch of algorithms can further be classified into logic and graph-based approaches. The first class of algorithms, developed mostly in Inductive Logic Programming, treats chemical graphs as relational structures addressing the problem to the context of learning in logic; the second class of algorithms regards them as labeled graphs addressing the problem to ►Graph Mining.

Cross References

►Hypothesis Language
►Inductive Logic Programming
►Observation Language
►Statistical Relational Learning
►Structured Induction

Recommended Reading

Cook, D., & Holder, L. (Eds.). (2007). *Mining graph data*. New York: Wiley.

De Raedt, L. (2008). *From inductive logic programming to multi-relational data mining*. Heidelberg: Springer.

Domingos, P., & Richardson, M. (2007). Markov logic: A unifying framework for statistical relational learning. In L. Getoor & B. Taskar (Eds.), *Introduction to statistical relational learning* (pp. 339–371). Cambridge, MA: MIT Press.

Emde, W., & Wettschereck, D. (1996). Relational instance based learning. In L. Saitta (Ed.), *Proceedings of the 13th international conference on machine learning* (pp. 122–130). San Francisco: Morgan Kaufmann.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations, 5*(1), 49–58.

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

Lavrac, N., Dzeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff (Ed.), *Proceedings of the 5th European working session on learning. Lecture notes in computer science* (Vol. 482, pp. 265–281). Berlin: Springer.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 83–134). San Francisco: Morgan Kaufmann.

Muggleton, S. H., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19,20*, 629–679.

Muggleton, S. H., & Feng, C. (1992). Efficient induction of logic programs. In S. Muggleton (Ed.), *Inductive logic programming* (pp. 291–298). London: Academic Press.

Poole, D. (2008). The independent choice logic and beyond. In L. De Raedt, P. Frasconi, K. Kersting, & S. Muggleton (Eds.), *Probabilistic inductive logic programming: Theory and application. Lecture notes in artificial intelligence* (Vol. 4911). Berlin: Springer.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*(3), 239–266.

Sato, T., & Kameya, Y. (2008). New advances in logic-based probabilistic modeling by PRISM. In L. De Raedt, P. Frasconi, K. Kersting, & S. Muggleton (Eds.), *Probabilistic inductive logic programming: Theory and application. Lecture notes in artificial intelligence* (Vol. 4911, pp. 118–155). Berlin: Springer.

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision* (pp. 157–209). New York: McGraw-Hill.

Learning from Labeled and Unlabeled Data

►Semi-Supervised Text Processing

Learning Graphical Models

Kevin B. Korb
Monash University, Clayton, Victoria, Australia

Synonyms

Bayesian model averaging; Causal discovery; Dynamic bayesian network; Learning bayesian networks

Definition

Learning graphical models (see Graphical Models) means to learn a graphical representation of either a causal or probabilistic model containing the variables $X_j \in \{X_i\}$. Although graphical models include more than directed acyclic graphs (DAGs), the focus here shall be on learning DAGs, as that is where the majority of research and application is taking place.

Definition 1 (Directed acyclic graph (DAG)) *A directed acyclic graph (DAG) is a set of variables (nodes, vertices) $\{X_i\}$ and a set of directed arcs (edges) between them such that following the arcs in their given direction can never lead from a variable back to itself.*

DAGs parameterized to represent probability distributions are otherwise known as *Bayesian networks*. Some necessary concepts and notation for discussing the learning of graphical models is given in Table 1.

A key characteristic of multivariate probability distributions is the conditional independence structure they give rise to, that is, the complete list of statements of the form

$$X_A \perp\!\!\!\perp X_B | X_C$$

true of the distribution. A goal of learning DAGs is to learn a minimal DAG representation of the conditional independence structure for a distribution given the Markov condition:

Definition 2 (Markov condition) *A DAG satisfies the Markov condition relative to a probability distribution if and only if for all X_i and X_j in the DAG $X_i \perp\!\!\!\perp X_j | \pi_{X_i}$ so long as X_j is not a descendant of X_i (i.e., X_j is not in the transitive closure of the parent relation starting from X_i).*

DAGs which violate the Markov condition are not capable of fully representing the relevant probabability distribution. Upon discovering such a violation, the normal response is to fix the model by adding missing arcs. In the causal discovery literature, this condition is often referred to as the *causal Markov condition*, which simply means the arcs are being interpreted as representing causal relationships and not merely as probabilistic dependencies.

Definition 3 (Markov Blanket) *The Markov blanket (MB) of a node X_i is the minimal set X_{MB} such that for all other nodes X_j in the model $X_i \perp\!\!\!\perp X_j | X_{MB}$.*

The Markov blanket consists of a node's parents, children, and its children's other parents.

Motivation and Background

Bayesian networks have enjoyed substantial success in thousands of diverse modeling, prediction, and control applications, including medical diagnosis, epidemiology, software engineering, ecology and environmental management, agriculture, intelligence and security, finance and marketing (see, e.g., http://www.norsys.com for customers implementing such applications and more). Many of these networks have been built by the traditional process of "knowledge engineering," that is, by eliciting both structure and conditional probabilities from human domain experts. That process is limited by the availability of expertise and also by the time and cost of performing such elicitation and subsequent model validation. In domains where significant quantities of

Learning Graphical Models. Table 1 Notation

Notation	Description
X_i	a random variable
$\mathbf{X}$	a set of random variables
$\{X_i\}$	a set of random variables indexed by $i \in I$
$X = x_j$ (or, x_j)	a random variable taking the value x_j
$p(x)$	the probability that $X = x$
$X_A \perp\!\!\!\perp X_B$	X_A and X_B are *independent* (i.e., $p(X_A) = p(X_A \mid X_B)$)
$X_A \perp\!\!\!\perp X_B \mid X_C$	X_A and X_B are *conditionally independent* given X_C (i.e., $p(X_A \mid X_C) = p(X_A \mid X_B, X_C)$)
$X_A \not\perp\!\!\!\perp X_B$	X_A and X_B are *dependent* (i.e., $p(X_A) \neq p(X_A \mid X_B)$)
$X_A \not\perp\!\!\!\perp X_B \mid X_C$	X_A and X_B are *conditionally dependent* given X_C (i.e., $p(X_A \mid X_C) \neq p(X_A \mid X_B, X_C)$)
π_{X_i}	the set of parents of X_i in a DAG (i.e., nodes Y such that $Y \rightarrow X_i$)

data are available it is pertinent to consider whether automated learning of Bayesian networks might either replace or compliment knowledge engineering. A variety of techniques for doing so have been developed, and the causal discovery of Bayesian networks is by now an important subindustry of data mining.

Theory

Probability and Causality

The key to learning Bayesian networks from sample data is the relation between causal dependence and probabilistic dependence. This is most easily understood in reference to undirected chains of variables, as in Fig. 1.

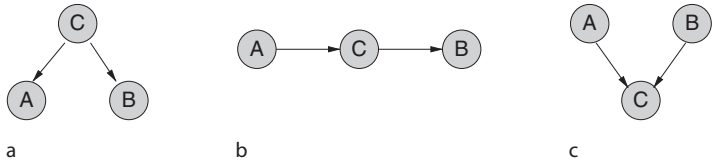
Where the arcs in Fig. 1 represent causal dependencies, then the probabilistic dependencies are as the caption describes. That is, in common causes and chains the end nodes A and B are rendered probabilistically independent of each other given knowledge of the state of C. Contrariwise, when A and B are parents of a common effect, and otherwise unrelated, they are probabilistically independent given no information (i.e., marginally independent), but *become* dependent given knowledge of C. This last relationship is often called "explaining away," because it corresponds to situations where, when already knowing the presence of, say, some disease C, the learning of the presence of a cause A reduces one's belief in some alternative explanation B of the disease.

These relationships between probabilistic dependence and causal dependence are the key for learning the causal structure of Bayesian networks because sample data allow one to estimate probabilistic dependencies directly, and the difference between conditional dependency structures in Fig. 1(a) and (b) versus its opposite in (c) allows automated learners to distinguish between these different underlying causal patterns. (This is related to *d-separation* in Graphical Models.) This distinction is explicitly made use of in constraint learners, but also implicitly used by metric learners.

In addition to structure learning, parameter learning is necessary, that is, learning the conditional probabilities of nodes given their parent values (conditional probability tables). Straightforward counting methods are frequently employed, although Expectation Maximization, Gibbs Sampling, and other techniques may come into play when the available data are noisy.

Statistical Equivalence

Two DAGs are said to be statistically equivalent (or, Markov equivalent) when they contain the same variables and each can be parameterized so as to represent any probability distribution that the other can



Learning Graphical Models. Figure 1. Causality and probabilistic dependence: (a) common cause with $A \perp\!\!\!\perp B|C$; (b) causal chain with $A \perp\!\!\!\perp B|C$; (c) common effect with $A \not\perp\!\!\!\perp B|C$

represent. Verma and Pearl (1990) proved that DAGs are statistically equivalent just in case they have the same undirected arc structures and the identical set of uncovered common effects, that is, common effects such as in Fig. 1(c) where the two parents are not themselves directly connected. They dubbed the set of statistically equivalent models *patterns*; these can be represented using partially directed acyclic graphs (PDAGs), that is, graphs with some arcs left undirected. Chickering (1995) showed that statistically equivalent models have identical maximum likelihoods relative to any given set of data. This latter result has suggested to many that causal learning programs can have no reasonable ambition to learn anything other than patterns, that is, any learner's discrimination between DAGs within a common pattern can only be based upon prior probability (e.g., prejudice). This is suggested, for example, by the fact that Bayesian learning combines (by multiplying) prior probabilities and likelihoods, so identical likelihoods will always lead to identical conclusions should the priors also be the same. One shall note some reason to doubt this supposed limit to causal discovery below.

Applications

Constraint Learners

The most direct application of the above ideas to learning Bayesian networks is exhibited in what may be called constraint learners. These programs assess conditional independencies between paired sets of variables given some other set of observed variables using statistical tests on the data, eliminating all DAGs that are incompatible with the independencies and dependencies asserted by the statistical test. (For this reason these programs are often called "conditional independence learners"; however, that tag is misleading, as is explained below.) The original such algorithm, the IC algorithm of Verma and Pearl (1990), can be described in simplified form as three rules for constructing a network from Yes or No answers to questions of the form "Is it the case that $X \perp\!\!\!\perp Y|\mathbf{W}$?"

Rule I: Put an undirected link between any two variables X and Y if and only if for every set of variables $\mathbf{W}$ s.t. $X, Y \notin \mathbf{W}$

$$X \not\perp\!\!\!\perp Y|\mathbf{W}$$

that is, X and Y are directly connected if and only if they are dependent under every conditioning set (including $\varnothing$).

Rule II: For every undirected structure $X - Y - Z$, orient the arcs $X \rightarrow Y \leftarrow Z$ if and only if

$$X \not\perp\!\!\!\perp Z|\mathbf{W}$$

for **every** $\mathbf{W}$ s.t. $X, Z \notin \mathbf{W}$ and $Y \in \mathbf{Z}$.
that is, Y is an uncovered common effect if and only if the end variables X and Z are dependent under every conditioning set that includes Y.

Rule I is justified by the need to express the probabilistic dependency between X and Y under all possible circumstances. Rule II is justified by the asymmetry in probabilistic dependencies illustrated in Fig. 1.

Application of these two rules is then followed by applying a Rule III, which just checks for any arc directions that are forced by further considerations, such as avoiding the introduction of cycles or any uncovered common effects not already identified in Rule II, and so not supported by the conditional independence tests.

This algorithm was first put into practice in the PC algorithm distributed as a part of the TETRAD program (Spirtes, Glymour, & Scheines, 1993). Aside from introducing some algorithmic efficiencies, PC adds orthodox statistical tests to answer the conditional independence questions. In the case of linear models, it uses a statistical signficance test for vanishing partial correlations,

accepting a dependence when and only when the test is statistically significant. For discrete networks a χ^2 test replaces the correlation test. Margaritis and Thrun further improve the algorithm's efficiency by limiting conditioning sets to the Markov blankets of the variable under test (Margaritis and Thrun, 2000). The PC algorithm has become the most widely used Bayesian network learner, available in `weka` and many Bayesian network modeling tools.

Metric Learners

Constraint learners attempt to build up a network using a sequence of independent statistical tests. One problem with them is that when one such test gives an incorrect result, subsequent tests will assume that result, with the potential for errors to cascade. Metric learners, by contrast, use some score applied to a network as a whole to assess it relative to the data. The earliest of this kind, by Cooper and Herskovits, turned the computation of a Bayesian measure into a counting problem. Under a number of fairly strong assumptions, such as that children variable states are always uniformly distributed given their parent states, they derived the measure

$$P(d,e) = P(d)\prod_{k=1}^{n}\prod_{j=1}^{s_{\pi(k)}}\frac{(s_k-1)!}{(S_{kj}+s_k-1)!}\prod_{l=1}^{s_k}\alpha_{kjl}!$$

where d is the DAG being scored, e the data, n the number of variables, s_k the number of values X_k may take, $s_{\pi(k)}$ the number of values the parents of X_k may take, S_{kj} the number of cases in the data where π_k takes its j-th value, and α_{kjl} is the number of cases where X_k takes its l-th value and π_k takes its j-th value. Cooper and Herskovits proved that this measure can be computed in polynomial time. Assuming the adequacy of this probability distribution, computation of the joint probability suffices for Bayesian learning, since by Bayes' Theorem maximizing $P(d,e)$ is equivalent to maximizing the posterior probability of d. Cooper and Herskovits applied this measure in the program K2, which required as inputs both the data and a total ordering of the variables. The latter input eliminates all problems about discovering arc orientations, which could be considered a cheat since, as the discussion of the IC algorithm showed, this is a part of the causal learning problem. Subsequently, Chow and Liu's (1968) maximum weighted spanning tree algorithm (MWST) has been used as a preprocessor to K2, doing a reasonable job of finding an ordering based upon the mutual information between pairs of variables.

A wide variety of alternative metrics for DAGs have been developed since K2. Heckerman, Geiger, and Chickering (1994) generalized the K2 metric to incorporate prior information, yielding BD (Bayesian metric with Dirichlet priors). Other alternatives include Minimum Description Length (MDL) scores (Bouckaert, 1993; Suzuki, 1996, 1999), Bayesian Information Criterion (BIC) (Cruz-Ramírez, Acosta-Mesa, Barrientos-Martínez, & Nava-Fernández , 2006) and Minimum Message Length (MML) (Korb & Nicholson, 2004; Wallace, Korb, & Dai, 1996). Although all of these measures score the DAG as a whole relative to some data set, they are just as (or more) sensitive to the individual dependencies and independencies between variables as are the constraint learners. The difference between the two types of learners is not whether they attend to the sets of conditional independencies expressed in the data, but whether they do so serially (which the constraint learners do) or collectively (as do the metric learners).

The question naturally arises whether constraint learners as a class are superior to metric learners or vice versa, or, indeed, which individual learner might be best. There is no settled answer to such questions, nor, in fact, is there any agreement about *how* such questions are best settled, even for fixed domains or data sets. Perhaps the issue is more general than that of learning Bayesian networks, since the fundamental theory of machine learning evaluation seems to be massively underdeveloped (see Algorithm Evaluation). In consequence, while nearly every new publication claims superiority in some sense for its preferred algorithm, the evidential basis for such claims remains suspect. It is clear, nonetheless, that many of the programs available are helpful with data analysis and are being so applied.

Search and Complexity

The space of DAGs is superexponential in the number of variables, making the learning process hard; it is NP-hard to be exact (Chickering, Heckerman, & Meek, 2004). In practice there are limits to the effectiveness of each algorithm, imposed by the number of variables (see Dimensionality), the number of

joint states the variables may take and the amount of data. The known limitations for different algorithms are scattered throughout the literature. The next section introduces some ideas for scaling up causal discovery.

Greedy search has frequently been used with both constraint-based and metric-based learning. The PC algorithm searching the space of patterns is an example, as it starts with a fully connected graph and searches greedily for arcs to remove. Chickering and Meek's Greedy Equivalence Search (GES) is another greedy algorithm operating in the pattern space (Chickering and Meek, 2002). Cooper and Herskovits' K2 is also a greedy searcher, adding arcs so long as single arc additions increases the probability score for the network. Bouckaert adopted this approach with his MDL score (Bouckaert, 1993). Greedy searches, of course, tend to get lost in local maxima, and Suzuki loosened the search method for his MDL scoring, using branch and bound (Suzuki, 1999).

Genetic algorithms (GAs) have been successfully applied to learning Bayesian networks. Larrañaga et al. used GAs over the space of total orderings to maximize the K2 score (Larrañaga, Kuijpers, Murga, & Yurramendi, 1996); Neil and Korb developed a GA searching the DAG space to maximize the MML score (Neil & Korb, 1999). A similar approach using MDL is found in Wong, Lam, and Leung (1999).

Markov Chain Monte Carlo (MCMC) searches perform stochastic sampling over the model space and have become a popular technique for Bayesian network learning. Gibbs sampling is used in Chickering and Heckerman (1997), where they compare a number of different metrics (and incorrectly conflate BIC and MDL scores; see Cruz-Ramírez, 2006) for learning a restricted class of Bayesian networks. Another MCMC approach, the Metropolis-Hastings algorithm, has been to estimate the posterior probability distribution over the space of total orderings, using the MML score (Korb & Nicholson, 2004, Chap. 8).

An alternative to model selection — searching for the single best model — is Bayesian model averaging, that is, searching for a set of models and weights for each of them (Chickering & Heckerman, 1997). And an alternative to *that* is to find a single Bayesian network that is equivalent to an averaged selection of networks (Dash & Cooper, 2004).

Markov Blanket Discovery

Recently, interest has grown in algorithms to learn, specifically, the Markov blankets around individual variables, which is a special kind of feature selection problem (see Feature Selection). One use for this is in prediction: since the MB renders all other variables conditionally independent of a target variable, finding the MB means having all the variables required for an optimal predictor. Tsamardinos et al. describe the max-min hill-climbing (MMHC) algorithm for MB discovery (Tsamardinos, Brown, & Aliferis, 2006). Nägele et al. apply this to learning in very high-dimensional spaces (Nägele, Dejori, & Stetter, 2007).

Given the MB for a target variable, one can then simply apply regression techniques (or any predictive classification technique) to the discovered variables. This works fine for standard prediction, but does not generalize to situations where some of the predictor variables are externally modified rather than observed. For an interesting collection of papers mostly applying some kind of Markov blanket discovery approach to prediction see the collection (Guyon et al., 2008).

A different use for local discovery is to avoid problems with computational complexity, whether due to the "curse of dimensionality" (too many variables) or the growing availability of very large data sets. Once the Markov blanket is found, one can employ causal discovery within the reduced set of variables, yielding local causal discovery. Iterating this will yield multiple causal subnetworks, when a global causal network might be stitched together from them (Aliferis, Statnikov, Tsamardinos, Mani, & Koutsoukos, 2010b), completing the whole causal discovery process while evading complexity problems. A current review of the issues and techniques can be found in two companion articles by Aliferis, Statnikov, Tsamardinos, Mani, and Koutsoukos (2010a, 2010b).

Knowledge Engineering with Bayesian Networks

Another approach to dealing with the complexity and tribulations of global causal discovery is to aid the discovery process with prior information. Bayesian inference is, after all, done by combining priors with likelihoods, and the priors need not always be perfectly flavorless, such as uniform priors over the DAG space. In almost all applications where data threaten to overwhelm automated discovery there is also at least some

expertise, if only the ability to say, for example, that the sex of a patient is determined before adult lifestyle practices are adopted. Such temporal information provided to a discovery algorithm can provide a huge boost to the discovery process.

This quite simple kind of prior information, the temporal tiers within which the variables may be allocated, has been available in many of the discovery programs for a long time. PC, for example, allows tiers to be specified. K2 more restrictively required a total ordering of the variables. The methods described by Heckerman, Geiger, and Chickering (1994) go beyond tiers. They provide for the specification of a network or subnetwork; the prior probability of any network in the search space can be computed according to its distance from the network provided. They also introduced the idea of equivalent sample size, that is, the weight to be given the prior information relative to the data, meaning that their priors are soft (probabilistic) rather than hard constraints. O'Donnell et al. (2006) adapted their MML score to allow soft priors for tiers, dependencies, direct and indirect causal relations, and networks or subnetworks, with variable degrees of confidence.

The flexible combination of prior information (expertise) with data in the causal discovery process allows for a full-fledged knowledge engineering process in the construction of Bayesian networks. Experts may be consulted for structural or parametric information, data may be gathered, and these different contributions may be weighted or reweighted according to the results of sensitivity analyses or other tests. The result can be a much faster and more useful approach to building and applying Bayesian networks.

Causal discovery with meaningful priors, by the way, shows that limiting discovery to patterns is insufficient: better priors, or better use of priors, can make a significant difference *within* patterns of DAGs.

Cross References

►Dimensionality
►Feature Selection
►Graphical Models
►Hidden Markov Models

Recommended Reading

The PC algorithm and variants were initially documented in Spirtes et al. (1993); their second edition (Spirtes, Glymour, & Scheines, 2000) covers more ground. Their TETRAD IV program is available from their web site http://www.phil.cmu.edu/projects/tetrad/. PC is contained within (and is available also with the weka machine learning platform at http://www.cs.waikato.ac.nz/ml/weka/).

A well-known tutorial by David Heckerman (1999) (reprinted without change in Heckerman, 2008) is well worth looking at for background in causal discovery and parameterizing Bayesian networks. A more current review of many of the topics introduced here is to be found in a forthcoming article (Daly, Shen, & Aitken, forthcoming). For other good treatments of parameterization see Cowell, Dawid, Lauritzen, and Spiegelhalter (1999) or Neapolitan (2003).

There are a number of useful anthologies in the area of learning graphical models. *Learning in Graphical Models* (Jordan, 1999) is one of the best, including Heckerman's tutorial and a variety of excellent reviews of causal discovery methods, such as Markov Chain Monte Carlo search techniques.

Textbooks treating the learning of Bayesian networks include Borgelt and Kruse (2002), Neapolitan (2003), Korb and Nicholson (2004), and Koller and Friedman (2009).

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., & Koutsoukos, X. D. (2010a). Local causal and Markov blanket induction for causal discovery and feature selection for classification. Part I: Algorithms and empirical evaluation. *Journal of Machine Learning Research, 11*, 171–234.

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., & Koutsoukos, X. D. (2010b). Local causal and Markov blanket induction for causal discovery and feature selection for classification. Part II: Analysis and extensions. *Journal of Machine Learning Research, 11*, 235–284.

Borgelt, C., & Kruse, R. (2002). *Graphical models: Methods for data analysis and mining*. New York: Wiley.

Bouckaert, R. (1993). Probabilistic network construction using the minimum description length principle. *Lecture Notes in Computer Science, 747*, 41–48.

Chickering, D. M. (1995). A tranformational characterization of equivalent Bayesian network structures. In P. Besnard & S. Hanks (Eds.), *Proceedings of the 11th conference on uncertainty in artificial intelligence*, San Francisco (pp. 87–98).

Chickering, D. M., & Heckerman, D. (1997). Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning, 29*, 181–212.

Chickering, D. M., Heckerman, D., & Meek, C. (2004). Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research, 5*, 1287–1330.

Chickering, D. M., & Meek, C. (2002). Finding optimal Bayesian networks. In *Proceedings of the 18th annual conference on uncertainty in AI*, San Francisco (pp. 94–102).

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Cowell, R. G., Dawid, A. P., Lauritzen, St. L., & Spiegelhalter, D. J. (1999). *Probabilistic networks and expert systems*. New York: Springer.

Cruz-Ramírez, N., Acosta-Mesa, H. G., Barrientos-Martínez, R. E., & Nava-Fernández, L. A. (2006). How good are the Bayesian information criterion and the Minimum Description Length principle for model selection? A Bayesian network analysis. *Lecture Notes in Computer Science, 4293*, 494–504.

Daly, R., Shen, Q., & Aitken, S. (forthcoming). Learning Bayesian networks: Approaches and issues. *The Knowledge Engineering Review*.

Dash, D., & Cooper, G. F. (2004). Model averaging for prediction with discrete Bayesian networks. *Journal of Machine Learning Research, 5*, 1177–1203.

Guyon, I., Aliferis, C., Cooper, G., Elisseeff, A., Pellet, J.-P., Spirtes, P., et al. (Eds.) (2008). JMLR workshop and conference proceedings: Causation and prediction challenge (WCCI 2008), volume 3. *Journal of Machine Learning Research*.

Heckerman, D. (1999). A tutorial on learning with Bayesian networks. In M. Jordan, (Ed.), *Learning in graphical models* (pp. 301–354). Cambridge: MIT.

Heckerman, D. (2008). A tutorial on learning with Bayesian networks. In *Innovations in Bayesian networks* (pp. 33–82). Berlin: Springer Verlag.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. In Lopes de Mantras & D. Poole (Eds.), *Proceedings of the tenth conference on uncertainty in artificial intelligence*, San Francisco (pp. 293–301).

Jordan, M. I. (1999). *Learning in graphical models*. Cambridge, MA: MIT.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques*. Cambridge, MA: MIT.

Korb, K. B., & Nicholson, A. E. (2004). *Bayesian artificial intelligence*. Boca Raton, FL: CRC.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., & Yurramendi, Y. (1996). Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 26*, 487–493.

Margaritis, D., & Thrun, S. (2000). Bayesian network induction via local neighborhoods. In S. A. Solla, T. K. Leen, & K. R. Müller (Eds.), *Advances in neural information processing systems* (Vol. 12, pp. 505–511). Cambridge: MIT.

Nägele, A., Dejori, M., & Stetter, M. (2007). Bayesian substructure learning – Approximate learning of very large network structures. In *Proceedings of the 18th European conference on machine learning; Lecture notes in AI* (Vol. 4701, pp. 238–249). Warsaw, Poland.

Neapolitan, R. E. (2003). *Learning Bayesian networks*. Upper Saddle River, NJ: Prentice Hall.

Neil, J. R., & Korb, K. B. (1999). The evolution of causal models. In N. Zhong & L. Zhous (Eds.), *Third Pacific-Asia Conference on Knowledge Discovery and Datamining (PAKDD-99)*, Beijing, China (pp. 432–437). New York: Springer.

O'Donnell, R., Nicholson, A., Han, B., Korb, K., Alam, M., & Hope, L. (2006). Causal discovery with prior information. In *Australasian joint conference on artificial intelligence*, (pp. 1162–1167). New York: Springer.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, prediction and search*. In *Lecture notes in statistics* (Vol. 81). New York: Springer.

Spirtes, P., Glymour, C., & Scheines, R. (2000). *Causation, prediction and search*. (2nd ed.). Cambridge: MIT.

Suzuki, J. (1996). Learning Bayesian belief networks based on the minimum description length principle. In L. Saitta (Ed.) *Proceedings of the 13th international conference on machine learning*, San Francisco (pp. 462–470). Morgan Kaufman.

Suzuki, J. (1999). Learning Bayesian belief networks based on the MDL principle: An efficient algorithm using the branch and bound technique. *IEEE Transactions on Information and Systems, 82*, 356–367.

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning, 65*(1), 31–78.

Verma, T. S., & Pearl, J. (1990). Equivalence and synthesis of causal models. In *Proceedings of the sixth conference on uncertainty in AI*, Boston (pp. 220–227). San Mateo, CA: Morgan Kaufmann.

Wallace, C. S., Korb, K. B., & Dai, H. (1996). Causal discovery via MML. In L. Saitta, (Ed.), *Proceedings of the 13th international conference on machine learning*, San Francisco (pp. 516–524). Morgan Kaufman.

Wong, M. L., Lam, W., & Leung, K. S. (1999). Using evolutionary programming and minimum description length principle for data mining of Bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 21*(2), 174–178.

Learning in Logic

►Inductive Logic Programming

Learning in Worlds with Objects

►Relational Reinforcement Learning

Learning Models of Biological Sequences

William Stafford Noble[1], Christina Leslie[2]
[1]University of Washington, Seattle, WA, USA
[2]Memorial Sloan-Kettering Cancer Center, New York, NY

Definition

Hereditary information is stored in the nucleus of every living cell as biopolymers of deoxyribonucleic acids (DNA). DNA thus encodes the blueprint for all known forms of life. A DNA sequence can be expressed as a finite string over an alphabet of {A, C, G, T}, corresponding to the four DNA bases. The human genome consists of approximately 3 billion bases, divided among 23 chromosomes.

During its life, each cell makes temporary copies of short segments of DNA. These shortlived copies are

comprised of ribonucleic acid (RNA). Each 3-mer of RNA can subsequently be translated, via the universal genetic code, into one of 20 amino acids. The resulting amino acid sequence is called a protein, and the DNA sequence that en codes the protein is called a gene.

Machine learning has been used to build models of many different types of biological sequences. These include models of short, functional elements within DNA or protein sequences, as well as models of genes, RNAs, and proteins.

Motivation and Background

Fundamentally, the motivation for building models of biological sequences is to understand the molecular mechanisms of the cell and the molecular basis for human disease. Each subheading below describes a different type of model, each of which attempts to capture a different facet of the underlying biology. All these models, ultimately, aim to uncover either evolutionary or functional relationships among sequences.

Although DNA and protein sequences were available in small numbers as early as the 1950s, a significant number of sequences were not available until the 1980s. Most of the advances in model development occurred in the 1990s, with the exception of phylogenetic models, which were already being developed in the 1970s.

Structure of Learning System

Motifs

In the context of biological sequences, a "motif" is a short (typically 6–20 letters) subsequence that is functionally significant. A motif may correspond to, e.g., the location along the DNA strand where a particular protein binds, or conversely, the location along the protein that binds to the DNA. The motif can arise either via convergent evolution (when two sequences evolve to look similar to one another) or via evolutionary conservation (if sequences that lack the motif are likely to be eliminated via natural selection).

Motif discovery is the problem of identifying a previously unknown motif within a given collection of sequences, by finding patterns that occur more often than one would expect by chance. The problem is challenging in part because two occurences of a given motif may not resemble each other exactly.

Work on motif discovery falls into two camps, based upon how the motifs themselves are represented. One camp uses position-specific scoring matrices (PSSMs), in which a motif of width w over an alphabet of size A is represented as a w-by-A probability matrix. In this matrix, each entry represents the probability that a given letter occurs at the given position. Early work in this area used expectation-maximization to identify protein motifs (Lawrence & Reilly, 1990). This effort was significantly extended in the MEME algorithm (Bailey & Elkan, 1994), which continues to be widely used today. A complementary approach uses Gibbs sampling (Lawrence, Altschul, Boguski, Liu, Neuwald, & Wootton, 1993), which offers several benefits, including avoiding local minima and the ability to sample multiple motifs simultaneously.

The other motif discovery camp uses a discrete motif representation, in which each motif is represented as a consensus sequence plus a specified maximum number of mismatches. In this formalism, enumerative methods can guarantee solving a given problem to optimality. For realistic problem sizes, this approach is most applicable to DNA, because of its much smaller alphabet size. Currently, perhaps the most popular such method is Weeder (Pavesi, Mereghetti, Mauri, & Pesole, 2004), which performed well in a recent comparison of motif discovery algorithms (Tompa, Li, Bailey, Church, Moor, Eskin, et al., 2005).

Proteins

A central problem in computational biology is the classification of proteins into functional and structural classes given their amino acid sequences. The 3D structure that a protein assumes after folding largely determines its function in the cell. However, directly obtaining a protein's 3D structure involves difficult experimental techniques such as X-ray crystallography or nuclear magnetic resonance, whereas it is relatively easy to determine a protein's sequence. Through evolution, structure is more conserved than sequence, so that detecting even very subtle sequence similarities, or remote homology, is important for predicting function.

Since the early 1980s, researchers have developed a battery of successively more powerful methods for detecting protein sequence similarities. This development can be broken into three main stages. Early methods focused on the *pairwise comparison* problem

and assessed the statistical significance of similarities between two proteins based on pairwise alignment. These methods are only capable of recognizing relatively close homologies. The BLAST algorithm (Altschul, Gish, Miller, Myers, & Lipman, 1990), based on heuristic alignment, and related tools are the most widely used methods for pairwise sequence comparison and database search today.

In the second stage, further accuracy was achieved by collecting aggregate statistics from a set of similar sequences and comparing the resulting statistics to a single, unlabeled protein of interest. One important example of *family-based models* are profile hidden Markov models (HMMs) (Krogh, Brown, Mian, Sjolander, & Haussler, 1994), probabilistic generative models estimated from a multiple alignment of sequences from a protein family. Profile HMMs generate variable length sequences by allowing insertions and deletions relative to the core residues of the alignment.

The third stage introduced *discriminative algorithms* based on classifiers like support vector machines for protein classification and remote homology detection. Such methods train both on positive sequences belonging to a protein family as well as negative examples consisting of sequences unrelated to the family. They require protein sequences to be represented using an explicit feature mapping or a kernel function in order to train the classifier. The first discriminative protein classification algorithm was the SVM-Fisher method (Jaakkola, Diekhans, & Haussler, 2000), which uses a profile HMM to extract a feature vector of Fisher scores for each input sequence x, defined by the gradient vector

$$\nabla_\theta \log P(x|\theta)|_{\theta=\theta_0},$$

where $\log P(x|\theta)$ is the log likelihood function of the sequence relative to the HMM and θ_0 is the maximum likelihood estimate for the model parameters. Another feature represention that has been used is the empirical kernel map

$$\Phi(x) = \langle s(x_1, x), \ldots, s(x_m, x)\rangle,$$

where $s(x, y)$ is a function depending on a pairwise similarity score between x and y and x_i, $i = 1 \ldots m$, are the training sequences Liano et al. (2002). In addition, it is possible to construct useful kernels directly without explicitly depending on generative models by using subsequence-based string kernels. For example, the mismatch kernel (Leslie, Eskin, Weston, & Noble, 2003) is defined by a histogram-like feature map. The feature space is indexed by all possible k-length subsequences $\alpha = a_1 a_2 \ldots a_k$, where each a_i is a character in the alphabet $\mathcal{A}$ of amino acids. The feature map is defined on k-gram α by $\Phi(\alpha) = (\phi_\beta(\alpha))_{\mathcal{A}^k}$ where $\phi_\beta(\alpha) = 1$ if α is within m mismatches of β, 0 otherwise, and is extended additively to longer sequences: $\Phi(x) = \sum_{k-\text{grams}\in x} \Phi(\alpha)$.

Genes

After a genome (or a portion of a genome) has been sequenced, a biologist's first question is usually, "Where are the genes?" In simple organisms, most of the genome is translated into proteins, and so the gene-finding problem reduces, essentially, to identifying the boundaries between genes. In more complex organisms, a large proportion of the genome consists of non protein coding DNA. The human genome, for example, is comprised of approximately 98% non-coding DNA. This non-coding DNA is interspersed between coding regions and even in the midst of a single coding region. The gene-finding problem, canonically, is to identify the regions of a given DNA sequence that encode proteins.

Initial methods for gene finding combined scores produced by different types of detectors. A *signal* detector attempts to recognize local, fixed-length features, such as characterize the boundaries between coding and non-coding regions within a single gene. A *content* detector attempts to recognize larger patterns on the basis of compositional statistics. Early gene finding algorithms combined these various scores in an *ad hoc* fashion to identify gene-like regions.

In the mid-1990s, several research groups began using HMMs for gene finding. HMMs provide a coherent, fully probabilistic method that is capable of capturing many of the complexities of real genes. Perhaps the most widely used such method is Genscan (Burge & Karlin, 1997), which uses fifth-order Markov statistics along with variable duration HMMs.

Gene finding is now a very mature field, but advances continue to be made using, e.g., conditional random field models (Bernal, Crammer, Hatzigeorgiou, & Pereira, 2007) and large-margin structured output techniques (Rätsch et al., 2007).

RNAs

Most RNA molecules are so-called messenger RNAs, which are used in the production of a corresponding protein molecule. Some RNAs, however, do not code for proteins but instead function on their own. These RNAs fall into functional categories, but they are not easily recognized by HMMs because (1) the RNAs themselves are often very short, and (2) functional RNA typically folds up in a deterministic fashion, and therefore exhibits nonlocal dependencies along the RNA sequence.

Useful RNA modeling is therefore accomplished using covariance models, which are a subclass of stochastic context-free grammars. The foundational work in this area was due to Eddy and Durbin (1994), who addressed both the structure inference problem and the inference of transition and emission probabilities given the structure. They applied these algorithms to transfer RNAs (tRNAs), and the approach was the basis for widely used tools such as Rfam.

Much effort in RNA covariance models has been devoted to improving the time and space efficiency of the algorithms associated with covariance models. For example, Eddy (2002) introduced a memory-efficient variant of the core dynamic programming algorithm used to align a covariance model to an RNA sequence. This improvement was practically important, since it reduced the $O(N^3)$ space requirement for a length N RNA sequence. Other work has focused on accelerating database search using the modeled families.

Recent efforts have focused on algorithms for genome-wide screens to discover functional noncoding RNAs as well as small regulatory RNAs like microRNAs. Various approaches to this problem have incorporated conservation as well as RNA structure prediction, both using covariance models and other methodologies. One such algorithm is RNAz (Washietl, Hofacker, & Stadler, 2005), which combines a measure for thermodynamic stability with a measure for structure conservation in an SVM approach to detect functional RNAs in multiple sequence alignments.

Phylogenetic Models

Phylogenetic models attempt to infer the series of evolutionary events (mutations, insertions, deletions, etc.) that gave rise to an observed collection of DNA or protein sequences. In most cases, these models ignore the possibility of copying DNA between individuals or species, and therefore represent the history as a phylogenetic tree, in which leaf nodes represent the observed sequences, and the internal nodes represent unobserved ancestral sequences. Of primary interest is inferring the topology and branch lengths of this tree.

Methods for phylogenetic tree inference can be divided into three classes: parsimony, distance, and likelihood methods, all described in detail in Felsenstein (2003).

Parsimony methods search for a tree that requires the smallest number of mutations, insertions or deletions along its branches. Because the search space of possible tree topologies is so large, this approach is feasible only for relatively small sets of sequences – tens rather than hundreds. Also, because parsimony models do not allow for so-called back-mutations – where a letter mutates to a different letter and then back again – and other similar events, parsimony models are provably suboptimal for distantly related sequences.

Distance methods replace parsimony with a generalized notion of distance, which may include back-mutation. A series of increasingly sophisticated distance metrics have been developed in this domain, starting with the one-parameter Jukes-Cantor model and the two-parameter Kimura model. Given an all-versus-all distance matrix, various tree inference algorithms can be used, including neighbor joining and agglomerative hierarchical clustering (called UPGMA in phylogenetics).

The third class of models use a fully probabilistic approach and attempt to infer the tree with maximum likelihood, given the observed sequences. This approach was first outlined by Felsenstein (1973), but was not computationally feasible for large sets of sequences until recently. Current methods employ Markov chain Monte Carlo methods to carry out the search.

Programs and Data

Following are some of the more popular web sites for performing biological sequence analysis:

- BLAST and PSI-BLAST (http://www.ncbi.nlm.nih.gov/BLAST) search a protein or DNA sequence database with a given, query sequence, and return a ranked list of homologs.

- MEME (http://meme.sdsc.edu) searches a given set of DNA or protein sequences for one or more recurrent motif patterns.
- HMMER (http://hmmer.janelia.org) is an HMM toolkit for training and searching with profile HMMs of proteins.
- Pfam (http://pfam.janelia.org) is a searchable library of profile HMMs corresponding to a curated collection of homologous protein domains.
- Rfam (http://rfam.janelia.org) is an analagous database of multiple sequence alignments and covariance models covering many common noncoding RNA families.
- PHYLIP (http://evolution.genetics.washington.edu/phylip.html) is a free software toolkit that includes many common phylogenetic inference algorithms.

Recommended Reading

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). A basic local alignment search tool. *Journal of Molecular Biology, 215*, 403–410.

Bailey, T. L., & Elkan, C. P. (1994). Fitting a mixture model by expectation-maximization to discover motifs in biopolymers. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, & D. Searls (Eds.), *Proceedings of the second international conference on intelligent systems for molecular biology* (pp. 28–36). AAAI Press.

Bernal, A., Crammer, K., Hatzigeorgiou, A., & Pereira, F. (2007). Global discriminative learning for higher-accuracy computational gene prediction. *PLoS Computational Biology*, 3, c54.

Burge, C., & Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, *268*(1), 78–94.

Eddy, S. R. (2002). A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an rna secondary structure. *BMC Bioinformatics*, *3*, 18.

Eddy, S. R., & Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucleic Acids Research*, *22*, 2079–2088.

Felsenstein, J. (1973). Maximum-likelihood estimation of evolutionary trees from continuous characters. *American Journal of Human Genetics*, *25*, 471–492.

Felsenstein, J. (2003). *Inferring phylogenies*. Sunderland MA: Sinauer Associates, 2003.

Jaakkola, T., Diekhans, M., & Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2), 95–114.

Krogh, A., Brown, M., Mian, I., Sjolander, K., & Haussler, D. (1994). Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, *235*, 1501–1531.

Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., & Wootton, J. C. (1993). Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, *262*(5131), 208–214.

Lawrence, C. E., & Reilly, A. A. (1990). An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, *7*(1), 41–51.

Leslie, C., Eskin, E., Weston, J., & Noble, W. S. (2003). Mismatch string kernels for SVM protein classification. In S. Becker, Thrun, & Obermayer (Eds.) *Advances in neural information processing systems*, (pp. 1441–1448). Cambridge, MA: 2003. MIT Press.

Liao, Li and William Stafford Noble. "Combining pairwise sequence similarity and support vector machines for remote protein homology detection". In *Proceedings of the sixth annual international conferrence on research in computational molecular biology*, April 18-21, 2002. pp. 225–232.

Pavesi, G., Mereghetti, P., Mauri, G., & Pesole, G. (2004). Weeder web: Discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, *32* (Web server issue), W199–203.

Rätsch, G., Sonnenburg, S., Srinivasan, J., Witte, H., Müller, K. R., Sommer, R., et al. (2007). Improving the *C. elegans* genome annotation using machine learning. *PLoS Computational Biology*, *3*(2), e20.

Tompa, M., Li, N., Bailey, T. L., Church, G. M., de Moor, B., Eskin, E., et al. (2005). Assessing Computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, *23*(1), 137–144.

Washietl, S., Hofacker, I. L., & Stadler, P. F. (2005). Fast and reliable prediction of noncoding rnas. *Proceedings of the National Academy of Sciences USA*, *102*(7), 2454–2459.

Learning Vector Quantization

Synonyms

LVQ

Definition

Learning vector quantization (LVQ) algorithms produce prototype-based classifiers. Given a set of labeled prototype vectors, each input vector is mapped to the closest prototype, and classified according to its label. The basic LVQ learning algorithm works by iteratively moving the closest prototype toward the current input if their labels are the same, and away from the input if not. Some variants of the algorithm have been shown to approximate Bayes optimal decision borders. The algorithm was introduced by Kohonen, and being prototype-based it bears close resemblance to ▶competitive learning and ▶Self-Organizing Maps. The differences are that LVQ is supervised and the prototypes are not ordered (i.e., there is no neighborhood function).

Learning with Different Classification Costs

►Cost-Sensitive Learning

Learning with Hidden Context

►Concept Drift

Learning Word Senses

►Word Sense Disambiguation

Least-Squares Reinforcement Learning Methods

Michail G. Lagoudakis
Technical University of Crete
Crete, Greece

Definition

Most algorithms for sequential decision making rely on computing or learning a value function that captures the expected long-term return of a decision at any given state. Value functions are in general complex, nonlinear functions that cannot be represented compactly as they are defined over the entire state or state-action space. Therefore, most practical algorithms rely on value function approximation methods and the most common choice for approximation architecture is a linear architecture. Exploiting the properties of linear architectures, a number of efficient learning algorithms based on least-squares techniques have been developed. These algorithms focus on different aspects of the approximation problem and deliver diverse solutions, nevertheless they share the tendency to process data collectively (batch mode) and, in general, achieve better results compared to their counterpart algorithms based on stochastic approximation.

Motivation and Background

Consider a ►Markov Decision ►Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is a Markovian transition model, $\mathcal{R}(s,a)$ is a reward function, $\gamma \in (0,1]$ is the discount factor, and $\mathcal{D}$ is the initial state distribution. A linear approximation architecture approximates the value function $V^\pi(s)$ or $Q^\pi(s,a)$ of a stationary (stochastic) policy $\pi(a|s)$ as a linear weighted combination of linearly-independent basis functions or features ϕ:

$$\widehat{V}^\pi(s;w) = \sum_{j=1}^{k} \phi_j(s) w_j = \phi(s)^\top w$$

$$\widehat{Q}^\pi(s,a;w) = \sum_{j=1}^{m} \phi_j(s,a) w_j = \phi(s,a)^\top w.$$

The parameters or weights of the approximation are the coefficients w.

Let V^π and $\widehat{V}^\pi$ be the exact and the approximate, respectively, state value function of a policy π, both given as column vectors of size $|\mathcal{S}|$. Define $\mathbf{\Phi}_V$ as the $(|\mathcal{S}| \times k)$ matrix with elements $\phi_j(s)$, where $s \in \mathcal{S}$ span the rows and $j = 1, 2, ..., k$ span the columns. Then, $\widehat{V}^\pi$ can be expressed compactly as $\widehat{V}^\pi = \mathbf{\Phi}_V w^\pi$. Similarly, let Q^π and $\widehat{Q}^\pi$ be the exact and the approximate, respectively, state-action value function of a policy π, both given as column vectors of size $|\mathcal{S}||\mathcal{A}|$. Define $\mathbf{\Phi}_Q$ as the $(|\mathcal{S}||\mathcal{A}| \times m)$ matrix with elements $\phi_j(s,a)$, where $(s,a) \in (\mathcal{S} \times \mathcal{A})$ span the rows and $j = 1, 2, ..., m$ span the columns. Then, $\widehat{Q}^\pi$ can be expressed compactly as $\widehat{Q}^\pi = \mathbf{\Phi}_Q w^\pi$. In addition, let $\mathcal{R}$ be a vector of size $|\mathcal{S}||\mathcal{A}|$ with entries $\mathcal{R}(s,a)$ that contains the reward function, $\mathbf{P}$ be a stochastic matrix of size $(|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|)$ that contains the transition model $(\mathbf{P}((s,a),s') = \mathcal{P}(s'|s,a))$, and $\mathbf{\Pi}_\pi$ be a stochastic matrix of size $(|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|)$ that describes policy π $(\mathbf{\Pi}_\pi(s,(s,a)) = \pi(a|s))$. The state value function V^π and the state-action value function Q^π are the solutions of the linear Bellman equations

$$V^\pi = \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} V^\pi)$$

$$Q^\pi = \mathcal{R} + \gamma \mathbf{P} \mathbf{\Pi}_\pi Q^\pi$$

and also the fixed points of the corresponding linear Bellman operators

$$V^\pi = T_V^\pi(V^\pi), \quad \text{where } T_V^\pi(x) = \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P} x)$$
$$Q^\pi = T_Q^\pi(Q^\pi), \quad \text{where } T_Q^\pi(x) = \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi x.$$

If V^π and Q^π were known, they could be projected orthogonally onto the space spanned by the basis functions to obtain the optimal least-squares approximation. (For simplicity of presentation, we consider only uniform least-squares criteria in this text, but generalization to weighted least-squares criteria is possible in all cases). For the state value function we have:

$$\widehat{V}^\pi = \mathbf{\Phi}_V w^\pi = \mathbf{\Phi}_V \left(\mathbf{\Phi}_V^\top \mathbf{\Phi}_V\right)^{-1} \mathbf{\Phi}_V^\top V^\pi$$
$$w^\pi = \mathbf{\Phi}_V^{-1} \mathbf{\Phi}_V \left(\mathbf{\Phi}_V^\top \mathbf{\Phi}_V\right)^{-1} \mathbf{\Phi}_V^\top V^\pi,$$

whereas for the state-action value function we have:

$$\widehat{Q}^\pi = \mathbf{\Phi}_Q w^\pi = \mathbf{\Phi}_Q \left(\mathbf{\Phi}_Q^\top \mathbf{\Phi}_Q\right)^{-1} \mathbf{\Phi}_Q^\top Q^\pi$$
$$w^\pi = \mathbf{\Phi}_Q^{-1} \mathbf{\Phi}_Q \left(\mathbf{\Phi}_Q^\top \mathbf{\Phi}_Q\right)^{-1} \mathbf{\Phi}_Q^\top Q^\pi.$$

The learning algorithms described here strive to find a set of parameters w, such that the approximate value function is a good approximation to the true one. However, since the exact value functions are unknown, these algorithms have to rely on information contained in the Bellman equations and the Bellman operators to derive expressions that characterize a good choice for w. It has been shown that, in many cases, this kind of learning is equivalent to approximating the MDP using a linear (compressed) model and solving exactly the approximate model (Parr et al., 2008).

Bellman Residual Minimizing Approximation

An obvious approach to deriving a good approximation is to require that the approximate function satisfies the linear Bellman equation as closely as possible. Substituting the approximation $\widehat{V}^\pi$ into the Bellman equation for V^π yields an overconstrained linear system over the k parameters w^π:

$$\widehat{V}^\pi \approx \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P}\widehat{V}^\pi)$$
$$\mathbf{\Phi}_V w^\pi \approx \mathbf{\Pi}_\pi(\mathcal{R} + \gamma \mathbf{P}\mathbf{\Phi}_V w^\pi)$$
$$(\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V) w^\pi \approx \mathbf{\Pi}_\pi \mathcal{R}.$$

Solving this overconstrained system in the least-squares sense is a $(k \times k)$ system

$$(\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V)^\top (\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V) w^\pi = (\mathbf{\Phi}_V - \gamma \mathbf{\Pi}_\pi \mathbf{P}\mathbf{\Phi}_V)^\top \mathbf{\Pi}_\pi \mathcal{R} \quad (1)$$

whose solution is unique and minimizes $\left\| T_V^\pi(\widehat{V}^\pi) - \widehat{V}^\pi \right\|_2$. Similarly, substituting the approximation $\widehat{Q}^\pi$ into the Bellman equation for Q^π yields an overconstrained linear system over the m parameters w^π:

$$\widehat{Q}^\pi \approx \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi \widehat{Q}^\pi$$
$$\mathbf{\Phi}_Q w^\pi \approx \mathcal{R} + \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q w^\pi$$
$$(\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q) w^\pi \approx \mathcal{R}.$$

Solving this overconstrained system in the least-squares sense is a $(m \times m)$ system

$$(\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q)^\top (\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q) w^\pi = (\mathbf{\Phi}_Q - \gamma \mathbf{P}\mathbf{\Pi}_\pi \mathbf{\Phi}_Q)^\top \mathcal{R} \quad (2)$$

whose solution is unique and minimizes $\left\| T_Q^\pi(\widehat{Q}^\pi) - \widehat{Q}^\pi \right\|_2$. In both cases, the solution minimizes the L_2 norm of the Bellman residual (the difference between the left-hand side and the right-hand side of the linear Bellman equation).

Least-Squares Fixed-Point Approximation

Recall that a value function is also the fixed point of the corresponding linear Bellman operator. Another way to go about finding a good approximation is to force the approximate value function to be a fixed point under the linear Bellman operator. For that to be possible, the fixed point has to lie in the space of approximate value functions, which is the space spanned by the basis functions. Even though the approximate function itself lies in that space by definition, the result of applying the linear Bellman operator to the approximation will in general be out of that space and must be projected back. Considering the orthogonal projection $\left(\mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top\right)$ (which minimizes the L_2 norm) onto the column space of $\mathbf{\Phi}$, we seek an approximate value function that is invariant under one application of the linear Bellman operator followed by orthogonal projection onto the space spanned by the basis functions.

More specifically, for the state value function, we require that

$$\widehat{V}^{\pi} = \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^{\top}\boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^{\top} \left(T_V^{\pi}(\widehat{V}^{\pi})\right)$$
$$\boldsymbol{\Phi}_V w^{\pi} = \boldsymbol{\Phi}_V \left(\boldsymbol{\Phi}_V^{\top}\boldsymbol{\Phi}_V\right)^{-1} \boldsymbol{\Phi}_V^{\top} \left(\boldsymbol{\Pi}_{\pi}(\mathcal{R} + \gamma \mathbf{P}\boldsymbol{\Phi}_V w^{\pi})\right).$$

Note that the orthogonal projection to the column space of $\boldsymbol{\Phi}_V$ is well-defined, because the columns of $\boldsymbol{\Phi}_V$ (the basis functions) are linearly independent by definition. The expression above is equivalent to solving a $(k \times k)$ linear system

$$\boldsymbol{\Phi}_V^{\top}(\boldsymbol{\Phi}_V - \gamma \boldsymbol{\Pi}_{\pi}\mathbf{P}\boldsymbol{\Phi}_V)w^{\pi} = \boldsymbol{\Phi}_V^{\top}\boldsymbol{\Pi}_{\pi}\mathcal{R} \tag{3}$$

whose solution is guaranteed to exist for all, but finitely many, values of γ (Koller and Parr, 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^{π} yields a value function $\widehat{V}^{\pi}$, which is the least-squares fixed-point approximation to the true state value function. Similarly, for the state-action value function, we require that

$$\widehat{Q}^{\pi} = \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^{\top}\boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^{\top} \left(T_Q^{\pi}(\widehat{Q}^{\pi})\right)$$
$$\boldsymbol{\Phi}_Q w^{\pi} = \boldsymbol{\Phi}_Q \left(\boldsymbol{\Phi}_Q^{\top}\boldsymbol{\Phi}_Q\right)^{-1} \boldsymbol{\Phi}_Q^{\top} \left(\mathcal{R} + \gamma \mathbf{P}\boldsymbol{\Pi}_{\pi}\boldsymbol{\Phi}_Q w^{\pi}\right).$$

This is equivalent to solving a $(m \times m)$ linear system

$$\boldsymbol{\Phi}_Q^{\top}(\boldsymbol{\Phi}_Q - \gamma \mathbf{P}\boldsymbol{\Pi}_{\pi}\boldsymbol{\Phi}_Q)w^{\pi} = \boldsymbol{\Phi}_Q^{\top}\mathcal{R} \tag{4}$$

whose solution is again guaranteed to exist for all, but finitely many, values of γ (Koller and Parr, 2000) and minimizes (in fact, zeros out) the projected Bellman residual. Since the orthogonal projection minimizes the L_2 norm, the solution w^{π} yields a value function $\widehat{Q}^{\pi}$, which is the least-squares fixed-point approximation to the true state-action value function.

Structure of Learning System

Least-Squares Temporal Difference Learning

The least-squares temporal difference (LSTD) learning algorithm (Bradtke and Barto, 1996) learns the least-squares fixed-point approximation to the state value function V^{π} of a fixed policy π. In essence, LSTD attempts to form and solve the linear system of Equation 3 using sampling. Each sample (s, r, s') indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. LSTD processes a set of samples collectively to derive the weights of the approximate value function. LSTD is an on-policy method; it requires that all training samples are collected using the policy under evaluation. The LSTD algorithm is summarized in Algorithm 1.

LSTD improves upon the temporal difference (TD) learning algorithm for linear architectures by making efficient use of data and converging faster. The main advantage of LSTD over TD is the elimination of the slow stochastic approximation and the learning rate that needs careful adjustment. TD uses samples to make small modifications and then discards them. In contrast, with LSTD, the information gathered from a single sample remains present in the matrices of the linear system and is used in full every time the parameters are computed. In addition, as a consequence of the elimination of stochastic approximation, LSTD does not diverge.

LSTD(λ) (Boyan, 1999) is an extension to LSTD that spans the spectrum between LSTD ($\lambda = 0$) and ▶linear regression over Monte-Carlo returns ($\lambda = 1$) for $\lambda \in [0,1]$. LSTD(λ) for $\lambda > 0$ requires that samples come from complete episodes. RLSTD(λ) is a variant of LSTD(λ) that uses recursive least-squares techniques for efficient implementation (Xu et al., 2002).

Algorithm 1 Least-Squares Temporal Difference (LSTD)

$w = \textbf{LSTD}(D, k, \phi, \gamma)$

Input: samples D, integer k, basis functions ϕ, discount factor γ

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each sample $(s, r, s') \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\left(\phi(s) - \gamma\phi(s')\right)^{\top}$
  $b \leftarrow b + \phi(s)r$
end for
$w \leftarrow \mathbf{A}^{-1}b$
return w

Bellman Residual Minimization Learning

The main idea behind LSTD can also be used to learn the Bellman residual minimization approximation to the state value function V^π of a fixed policy π. In this case, the goal is to form and solve the linear system of Equation 1 using sampling. However, the structure of the system, in this case, requires that samples are "paired," which means that two independent samples (s, r, s') and (s, r, s'') for the same state s must be drawn to perform one update. This is necessary to obtain unbiased estimates of the system matrices. Each sample (s, r, s') again indicates a minimal interaction with the unknown process, whereby in some state s, a decision was made using policy π, and reward r was observed, as well as a transition to state s'. Obtaining paired samples is trivial with a generative model (a simulator) of the process, but virtually impossible when samples are drawn directly from a physical process. This fact makes the Bellman residual minimization approximation somewhat impractical for learning, but otherwise a reasonable approach for computing approximate state value functions from the model of the process (Schweitzer and Seidmann, 1985). The learning algorithm for Bellman residual minimization is summarized in Algorithm 2.

Hybrid Least-Squares Learning

Value function learning algorithms, either in the Bellman residual minimization or in the fixed point sense, have been used within approximate policy iteration schemes for policy learning, but in practice they exhibit quite diverse performance. Fixed-point approximations tend to deliver better policies, whereas Bellman residual minimization approximations fluctuate less between different rounds of policy iteration. Motivated by a desire to combine the advantages of both approximations, some researchers have focused on learning hybrid approximations that lie somewhere between these two extremes. Johns et al. (2009) have proposed two different approaches to combine these two approximations. The first relies on a derivation that begins with the goal of minimizing a convex combination of the two objectives (Bellman residual and projected Bellman residual); the resulting learning algorithm is quite expensive as it requires the maintenance of three matrices and two vectors (as opposed to one matrix and one vector when learning a non-hybrid approximation), as well as the inversion of one of the three matrices at each update. The second approach focuses directly on a convex combination of the linear systems produced by the two extreme approximations (Equations 1 and 3); the resulting learning algorithm has the same complexity as non-hybrid algorithms and in many cases exhibits better performance than the original approximations. On the other hand, both hybrid learning algorithms still have to deal with the paired samples problem and additionally require tuning of the convex combination parameter.

Algorithm 2 Bellman Residual Minimization Learning

$w = \mathbf{BRML}(D, k, \phi, \gamma)$

Input: paired samples D, integer k, basis functions ϕ, discount factor γ

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each pair of samples $[(s, r, s'), (s, r, s'')] \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + (\phi(s) - \gamma\phi(s'))(\phi(s) - \gamma\phi(s''))^\top$
  $b \leftarrow b + (\phi(s) - \gamma\phi(s'))\, r$
end for
$w \leftarrow \mathbf{A}^{-1} b$
return w

Least-Squares Policy Evaluation

The least-squares policy evaluation (LSPE) learning algorithm (Nedić and Bertsekas, 2003), like LSTD, learns the least-squares fixed-point approximation to the state value function V^π of a fixed policy π. Both LSPE and LSTD strive to obtain the solution to the same linear system (Equation 3), but using different methods; LSPE uses an iterative method, whereas LSTD uses direct matrix inversion. Unlike LSTD, LSPE begins with some arbitrary approximation to the value function (given by a parameter vector w') and focuses on the one-step application of the Bellman operator within the lower dimensional space spanned by the basis functions aiming at producing an incremental improvement on the parameters. In that sense, LSPE can take advantage of a good initialization of the parameter vector. Given the current parameters w' and a set

Algorithm 3 Least-Squares Policy Evaluation (LSPE)

$w = \textbf{LSPE}(D, k, \phi, \gamma, w', \alpha)$

Input: samples D, integer k, basis functions ϕ, discount factor γ, weights w', stepsize α

Output: weights w of the learned state value function

$\mathbf{A} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$b \leftarrow \mathbf{0}$ // $(k \times 1)$ vector
for each sample $(s, r, s') \in D$ **do**
  $\mathbf{A} \leftarrow \mathbf{A} + \phi(s)\phi(s)^\top$
  $b \leftarrow b + \phi(s)\left(r + \gamma\phi(s')^\top w'\right)$
end for
$\bar{w} \leftarrow \mathbf{A}^{-1} b$
$w \leftarrow \alpha w' + (1 - \alpha)\bar{w}$
return w

$\{(s_k, r_k, s'_k) : k = 0, \ldots, t\}$ of samples, LSPE first computes the solution $\bar{w}$ to the least-squares problem

$$\min_w \sum_{k=0}^{t} \left(\phi(s_k)^\top w - \left(r_k + \gamma\phi\left(s'_k\right)^\top w'\right)\right)^2$$

and then updates w' toward $\bar{w}$ using a stepsize $\alpha \in (0, 1]$. The LSPE algorithm is summarized in Algorithm 3.

The LSPE incremental update at the extreme can be performed whenever a new sample arrives or whenever a batch of samples becomes available to remedy computational costs. An efficiency improvement to LSPE is to use recursive least-squares computations, so that the least-squares problem can be solved without matrix inversion. LSPE(λ) for $\lambda \in [0, 1]$ is an extension of LSPE to multistep updates in the same spirit as LSTD(λ). LSPE(λ) for $\lambda > 0$ requires that samples come from complete episodes.

Least-Squares Policy Iteration

Least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) is a model-free, reinforcement learning algorithm for policy learning based on the approximate policy iteration framework. LSPI learns in a batch manner by processing multiple times the same set of samples. LSPI is an off-policy method; samples can be collected arbitrarily from the process using any policy. Each sample (s, a, r, s') indicates that the learner observed the current state s, chose an action a, and observed the resulting next state s' and the reward received r. LSPI iteratively learns a (weighted) least-squares fixed-point approximation of the state-action value functions (Equation 4) of a sequence of improving (deterministic) policies π. At each iteration, the value function of the policy is approximated by solving a $(m \times m)$ linear system, formed using the single sample set and the policy from the previous iteration. LSPI offers a non-divergence guarantee and in most cases it converges in just a few iterations. LSPI exhibits excellent sample efficiency and has been used widely in many domains. Algorithm 4 summarizes the LSPI algorithm.

The default internal policy evaluation procedure in LSPI is the variation of LSTD for the state-action value function (LSTDQ). However, any other value function learning algorithm, such as BRML or LSPE, could be used instead; nevertheless, the λ extensions are not applicable in this case, because the samples in LSPI have been collected arbitrarily and not by the policy being evaluated each time. The variation of LSPI that internally learns the Bellman residual minimizing approximation (Equation 2) using BRML has produced inferior policies, in general, and suffers from the paired samples problem.

Algorithm 4 Least-Squares Policy Iteration (LSPI)

$w = \textbf{LSPI}(D, m, \phi, \gamma, \epsilon)$

Input: samples D, integer m, basis functions ϕ, discount factor γ, tolerance ϵ

Output: weights w of the learned value function of the best learned policy

$w \leftarrow \mathbf{0}$
repeat
  $\mathbf{A} \leftarrow \mathbf{0}$ // $(m \times m)$ matrix
  $b \leftarrow \mathbf{0}$ // $(m \times 1)$ vector
  $w' \leftarrow w$
  for each sample (s, a, r, s') in D **do**
    $a' = \arg\max_{a'' \in \mathcal{A}} \phi(s', a'')^\top w'$
    $\mathbf{A} \leftarrow \mathbf{A} + \phi(s, a)\left(\phi(s, a) - \gamma\phi(s', a')\right)^\top$
    $b \leftarrow b + \phi(s, a)r$
  end for
  $w \leftarrow \mathbf{A}^{-1} b$
until $(\|w - w'\| < \epsilon)$
return w

Least-Squares Fitted *Q*-Iteration

Fitted *Q*-iteration (FQI) (Ernst et al., 2005) is a batch reinforcement learning algorithm for policy learning based on the popular *Q*-Learning algorithm. FQI uses an iterative scheme to approximate the optimal value function, whereby an improved value function Q is learned at each iteration by fitting a function approximator to a set of training examples generated using a set of samples from the process and the *Q*-Learning update rule. Any function approximation architecture and the corresponding supervised learning algorithm could be used in the iteration. The simplest choice is to use least-squares regression along with a linear architecture to learn the least-squares fixed-point approximation of the state-action value function (Equation 4). This version of least-squares fitted *Q*-iteration is summarized in Algorithm 5. In a sense, this version of FQI combines ideas from LSPE and LSPI. Like LSPI, FQI is an off-policy method; samples can be collected arbitrarily from the process using any policy. In practice, FQI produces very good policies within a moderate number of iterations.

Algorithm 5 Least-Squares Fitted Q-Iteration

$w = \textbf{LS-FQI}(D, m, \phi, \gamma, N)$

Input: samples D, integer m, basis functions ϕ, discount factor γ, iterations N

Output: weights w of the learned value function of the best learned policy

$i \leftarrow 0$
$w \leftarrow \mathbf{0}$
while $(i < N)$ **do**
  $\mathbf{A} \leftarrow \mathbf{0}$ // $(m \times m)$ matrix
  $b \leftarrow \mathbf{0}$ // $(m \times 1)$ vector
  for each sample (s, a, r, s') in D **do**
    $\mathbf{A} \leftarrow \mathbf{A} + \phi(s,a)\phi(s,a)^\top$
    $b \leftarrow b + \phi(s,a)\left(r + \gamma \max_{a' \in \mathcal{A}} \{\phi(s',a')^\top w\}\right)$
  end for
  $w \leftarrow \mathbf{A}^{-1} b$
  $i \leftarrow i + 1$
end while
return w

Cross References

- ▶Curse of Dimensionality
- ▶Feature Selection
- ▶Radial Basis Functions
- ▶Reinforcement Learning
- ▶Temporal Difference Learning
- ▶Value Function Approximation

Recommended Reading

Boyan, J. A. (1999). Least-squares temporal difference learning. *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, pp. 49–56.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning, 22*, 33–57.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research, 6*, 503–556.

Johns, J., Petrik, M., & Mahadevan, S. (2009). Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning, 76*(2–3), 243–256.

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, Stanford, CA, USA, pp. 326–334.

Lagoudakis, M. G., Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research, 4*, 1107–1149.

Nedić, A., & Bertsekas, D. P. (2003). Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications, 13*(1–2), 79–110.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., & Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning, Proceedings of the twenty-fifth international conference on machine learning, Helsinki, Finland, pp. 752–759.

Schweitzer, P. J., & Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications, 110*(6), 568–582.

Xu, X., He, H. G., & Hu, D. (2002). Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research, 16*, 259–292.

Leave-One-Out Cross-Validation

Definition

Leave-one-out cross-validation is a special case of ▶cross-validation where the number of folds equals the number of ▶instances in the ▶data set. Thus, the learning algorithm is applied once for each instance, using all other instances as a ▶training set and using the selected instance as a single-item ▶test set. This process is closely

related to the statistical method of jack-knife estimation (Efron, 1982).

Cross References

►Algorithm Evaluation

Recommended Reading

Efron, B. (1982). The Jackknife, the Bootstrap and other resampling plans. In *CBMS-NSF regional conference series in applied mathematics 1982*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM).

Leave-One-Out Error

Synonyms

Hold-one-out error; LOO error

Definition

Leave-one-out error is an estimate of ►error obtained by ►leave-one-out cross-validation.

Lessons-Learned Systems

►Case-Based Reasoning

Lifelong Learning

►Cumulative Learning

Life-Long Learning

►Continual Learning

Lift

Lift is a measure of the relative utility of a ►classification rule. It is calculated by dividing the probability of the consequent of the rule, given its antecedent by the prior probability of the consequent:

$$\text{lift}(x \rightarrow y) = \text{P}(Y = y \mid X = x)/\text{P}(Y = y).$$

In practice, the probabilities are usually estimated from either ►training data or ►test data. In this case,

$$\text{lift}(x \rightarrow y) = \text{F}(Y = y \mid X = x)/\text{F}(Y = y)$$

where $\text{F}(Y = y \mid X = x)$ is the frequency with which the consequent occurs in the data in the context of the antecedent and $\text{F}(Y = y)$ is the frequency of the consequent in the data.

Linear Discriminant

Novi Quadrianto, Wray L. Buntine
RSISE, ANU and SML, NICTA, Canberra, Australia

Definition

A discriminant is a function that takes an input variable and outputs a class label for it. A linear discriminant is a discriminant that is linear in the input variables. This article focuses on one such linear discriminant function called Fisher's linear discriminant. Fisher's discriminant works by finding a projection of input variables to a lower dimensional space while maintaining a class separability property.

Motivation and Background

The curse of dimensionality (►Curse of Dimensionality) is an ongoing problem in applying statistical techniques to pattern recognition problems. Techniques that are computationally tractable in low-dimensional spaces can become completely impractical in high-dimensional spaces. Consequently, various methods have been proposed to reduce the dimensionality of the input or feature space in the hope of obtaining a more manageable problem. This relies on the fact that real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the output variables occur may be so confined. For example, we can reduce a d-dimensional problem to one dimension

if we project the d-dimensional data onto a line. However, arbitrary projections will usually produce cluttered projected samples from all of the classes. Thus, the aim is to find a good projection so that the projected samples are well separated. This is exactly the goal of Fisher's linear discriminant analysis.

Fisher's Discriminant for Two-Category Problem

Given N observed training data points $\{(x_i, y_i)\}_{i=1}^{N}$ where $y_i \in \{1, \ldots, \Omega\}$ is the label for an input variable $x_i \in \mathbb{R}^d$, our task is to find the underlying discriminant function, $f : \mathbb{R}^d \to \{1, \ldots, \Omega\}$. The linear discriminant seeks a projection of d-dimensional input onto a line in the direction of $w \in \mathbb{R}^d$, such that

$$y = w^T x. \tag{1}$$

Subsequently, a class label assignment can be performed by thresholding the projected values, for example $y \geq C$ as class 1 and otherwise as class 2 for an appropriate choice of constant C. While the magnitude of w has no real significance (acts only as a scaling factor to y), the direction of w plays a crucial role. Inappropriate choice of w can result in an non-informative heavily cluttered line. However, by adjusting the components of weight w, we can find a projection that maximizes the class separability (Fig. 1). It is crucial to note that whenever the underlying data distributions are multimodal and highly overlapping, it might not be possible to find such a projection.

Consider a two-category problem, Ω_1 and Ω_2 with N_1 and N_2 number of data points, respectively. The d-dimensional sample mean is given by

$$\mu_1 = \frac{1}{N_1} \sum_{i \in \Omega_1} x_i \qquad \mu_2 = \frac{1}{N_2} \sum_{i \in \Omega_2} x_i. \tag{2}$$

The simplest class separability criterion is the separation of the projected class mean, that is we can find the weight w that maximizes

$$m_2 - m_1 = \frac{1}{N_2} \sum_{i \in \Omega_2} w^T x_i - \frac{1}{N_1} \sum_{i \in \Omega_1} w^T x_i = w^T (\mu_2 - \mu_1), \tag{3}$$

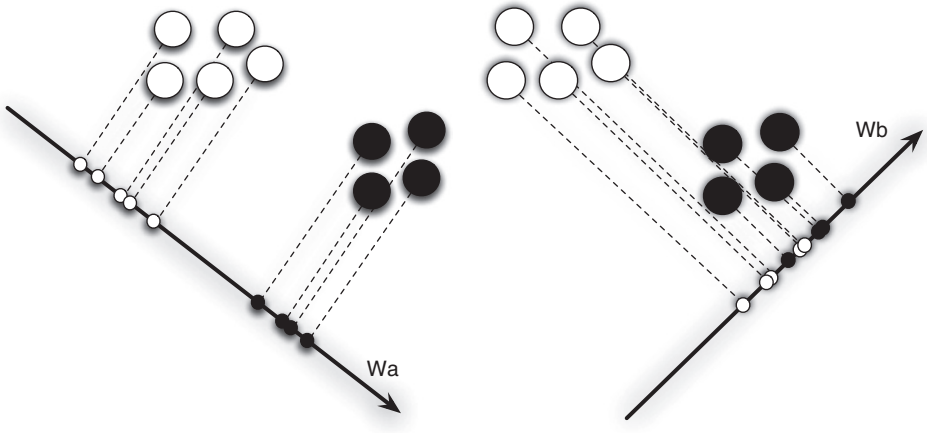
where m_1 and m_2 are the projected class means. An additional unit length constraint on w, i.e., $\sum_i w_i^2 = 1$ should be imposed to have a well-defined maximization problem. The above separability criterion produces a line that is parallel to the line joining the two means. However, this projection is sub-optimal whenever the data has distinct covariances depending on class (i.e., it is un-isotropic).

Fisher's criterion maximizes a large separation between the projected class means *while also* minimizing a variance within each class. This criterion can be expressed as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}. \tag{4}$$

where the total within-class covariance matrix is

$$S_W = \sum_{i \in \Omega_1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i \in \Omega_2} (x_i - \mu_2)(x_i - \mu_2)^T, \tag{5}$$



Linear Discriminant. Figure 1. Colors encode class labels. Projection of samples onto two different lines. The plot on the left shows greater separation between the white and black projected points

and a between-class covariance matrix is

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \quad (6)$$

The maximizer of (4) can be found by setting its first derivative with respect to the weights vector to zero, that is

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w. \quad (7)$$

It is clear from (6), that S_B is always in the direction of $(m_2 - m_1)$. As only the direction of w is important, we can drop the scaling factors in (7), those are $(w^T S_B w)$ and $(w^T S_W w)$. Multiplying both sides of (7) by S_W^{-1}, we can then obtain the solution of w that maximizes (4) as

$$w = S_W^{-1}(\mu_1 - \mu_2). \quad (8)$$

Fisher's Discriminant for Multi-category Problem

For the general Ω-class problem, we seek a projection from d-dimensional space to a $(\Omega - 1)$-dimensional space which is accomplished by $\Omega-1$ linear discriminant functions, that is

$$y_c = w_c^T x \qquad c = 1, \ldots, \Omega - 1. \quad (9)$$

In the matrix notation, $y = W^T x$ for $W \in \mathbb{R}^{d\times(\Omega-1)}$ and $y \in \mathbb{R}^{(\Omega-1)}$. The generalization of the within-class covariance matrix in (5) to the case of Ω classes is $S_W = \sum_{c=1}^{\Omega} S_c$ with $S_c = \sum_{i\in c}(x_i - \mu_c)(x_i - \mu_c)^T$. Following Duda and Hart (1973), the between-class covariance matrix is defined as the difference between the total covariance matrix, $\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^T$, where μ denotes the total sample mean of the dataset, and the within-class covariance matrix. One of the criterion to be optimized is (Fukunaga, 1990)

$$J(w) = \text{Trace}((W^T S_W W)^{-1}(W^T S_B W)). \quad (10)$$

The maximizer of (10) is eigenvectors of $S_W^{-1}S_B$ associated with $\Omega - 1$ largest eigenvalues. It is important to note that the between-class covariance matrix S_B is the sum of Ω matrices of rank one or less, and because only $\Omega - 1$ of these matrices are independent, S_B has rank at most equal to $\Omega-1$ and so there are at most $\Omega-1$ nonzero eigenvalues. Therefore, we are unable to find more than $\Omega - 1$ discriminant functions (Fukunaga, 1990).

Cross References

►Regression
►Support Vector Machines

Recommended Reading

Most good statistical text books cover this.
Bellman, R. E. (1961). *Adaptive control processes*. Princeton: Princeton University Press.
Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley.
Fukunaga, K. (1990). *Introduction to statistical pattern recognition* (2nd ed.). San Diego: Academic.

Linear Regression

Novi Quadrianto, Wray L. Buntine
RSISE, ANU and SML, NICTA, Canberra, Australia

Definition

Linear Regression is an instance of the ►Regression problem which is an approach to modelling a functional relationship between input variables x and an output/response variable y. In linear regression, a linear function of the input variables is used, and more generally a linear function of some vector function of the input variables $\phi(x)$ can also be used. The linear function estimates the mean of y (or more generally the median or a quantile).

Motivation and Background

Assume we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points, so y is to be approximated in some way by $f(x)$. This is the task covered in more detail in Regression. A general approach to the problem is to make the function $f()$ be linear. Depending on the optimization criteria used to fit between the linear function $f(x)$ and the output y, this can include many different regression techniques, but optimization is generally easier because of the linearity.

Theory/Solution

Formally, in a regression problem, we are interested in recovering a functional dependency $y_i = f(x_i) + \epsilon_i$ from N observed training data points $\{(x_i, y_i)\}_{i=1}^N$, where $y_i \in \mathbb{R}$ is the noisy observed output at input location $x_i \in \mathbb{R}^d$. For the linear parametric technique, we tackle this regression problem by parameterizing the latent regression function $f()$ by a parameter $w \in \mathbb{R}^H$, that is $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. Note that the function is a linear function of the weight vector w. The simplest form of the linear parametric model is when $\phi(x_i) = x_i \in \mathbb{R}^d$, that is the model is also linear with respect to the input variables, $f(x_i) := w_0 + w_1 x_i^1 + \cdots + w_d x_i^d$. Here the weight w_0 allows for any constant offset in the data. With general basis functions such as polynomials, exponentials, sigmoids, or even more sophisticated Fourier or wavelets bases, we can obtain a regression function which is nonlinear with respect to the input variables although still linear with respect to the parameters.

In the subsequent section, the simplest and thus common linear parametric method for solving a regression problem is covered, the least squares method.

Least Squares Method Let $X \in \mathbb{R}^{N \times d}$ be a matrix of input variables and $y \in \mathbb{R}^N$ be a vector of output variables. The least squares method minimizes the following sum of squared error,

$$E(w) = (Xw - y)^T (Xw - y) \tag{1}$$

to infer the weight vector w. Note that the above error function is quadratic in the w, thus the minimization has a unique solution and leads to a closed-form expression for the estimated value of the unknown weight vector w. The minimizer of the error function in (1) can be found by setting its first derivative with respect to the weight vector to zero, that is

$$\partial_w E(w) = 2X^T(Xw - y) = 0 \tag{2}$$

$$w^* = (X^T X)^{-1} X^T y. \tag{3}$$

The term

$$(X^T X)^{-1} X^T := X^\dagger \tag{4}$$

is known as the Moore-Penrose pseudo-inverse (Golub & Van Loan, 1996) of the matrix X. This quantity can be regarded as a generalization of a matrix inverse to nonsquare matrices. Whenever X is square and invertible, $X^\dagger \equiv X^{-1}$. Having computed the optimal weight vector, we can then predict the output value at a novel input location x_{new} simply by taking an inner product: $y_{\text{new}} = \langle \phi(x_{\text{new}}), w^* \rangle$.

Under the assumption of an independent and normally distributed noise term, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, the above least squares approach can be shown to be equivalent to the maximum likelihood solution. With the Gaussian noise term, the log-likelihood model on an output vector y and an input matrix X is

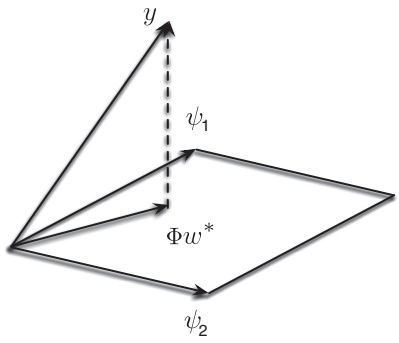
$$\ln p(y|X, w) = \ln \mathcal{N}(Xw, \sigma^2 I) \tag{5}$$

$$= -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - Xw)^T(y - Xw). \tag{6}$$

Maximizing the above likelihood function with respect to w will give the optimal weight to be in the form of (3). We can also find the maximum likelihood estimate of the noise variance by setting the first derivative of (6) with respect to σ^2 to zero, that is

$$\sigma^2_{\text{ML}} = \frac{1}{N}(y - Xw)^T(y - Xw). \tag{7}$$

Geometrical Interpretation of Least Squares Method Let y be a vector in an N-dimensional space whose axes are given by $\{y_i\}_{i=1}^N$. Each of the H basis functions evaluated at N input locations can also be represented as a vector in the same N-dimensional space. For notational convenience, we denote this vector as ψ_h. The H vectors ψ_h will span a linear subspace of dimensionality H whenever the number of basis functions H is smaller than the number of input locations N (see Fig. 1). Denote $\Phi \in \mathbb{R}^{N \times H}$ as a matrix whose rows are the vectors $\{\phi_h(x_i)\}_{h=1}^H$. Our linear prediction model, Φw (in the simplest form Xw) will be an arbitrary linear combination of the vectors ψ_h. Thus, it can live anywhere in the H-dimensional space. The sum of squared error criterion in (1) then corresponds to the squared Euclidean distance between Φw and y. Therefore, the least squares solution of w corresponds to the orthogonal projection of y onto the linear subspace. This orthogonal projection is associated with the minimum of the squared Euclidean distance. As a side note, from Fig. 1, it is clear that the vector $y - \Phi w$ is normal (perpendicular) to the range of Φ thus $\Phi^T \Phi w = \Phi^T y$ is called the normal equation associated with the least squares problem.



Linear Regression. Figure 1. Geometrical interpretation of least squares. The optimal solution w^* with respect to the least squares criterion corresponds to the orthogonal projection of y onto the linear subspace which is formed by the vectors of the basis functions

Practical note: The computation of (3) requires an inversion of an H by H matrix $\Phi^T\Phi$ (or a d by d matrix X^TX). A direct inversion of this matrix might lead to numerical difficulties when two or more basis vectors ψ_h or input dimensions are (nearly) collinear. This problem can be addressed conveniently by using Singular Value Decomposition (SVD) (Press, Teukolsky, Vetterling, & Flannery, 1992). It is important to note that adding a regularization term (see also the later section on ridge regression) ensures the non-singularity of $\Phi^T\Phi$ matrix, even in the presence of degeneracies.

Sequential Learning of Least Squares Method Computation of the optimal weight vector in (3) involves the whole training set comprising N data points. This learning technique is known as a batch algorithm. Real datasets can however involve large numbers of data points which might make batch techniques computationally prohibitive. In contrast, sequential algorithms or online algorithms process one data point at a time, and can be more suited to handle large datasets.

We can use a sequential algorithm called stochastic gradient descent for learning the optimal weight vector. The objective function of (1) can be decomposed into $\sum_{i=1}^{N}(\langle x_i, w\rangle - y_i)^2$. This transformation suggests a simple stochastic gradient descent procedure: we traverse the data point i and update the weight vector using

$$w^{t+1} \leftarrow w^t - 2\eta(\langle x_i, w^t\rangle - y_i)x_i, \tag{8}$$

This algorithm is known as LMS (Least Mean Squares) algorithm. In the above equation, t denotes the iteration number and η denotes the learning rate. The value of η needs to be chosen carefully to ensure the convergence of the algorithm.

Regularized/Penalized Least Squares Method The issue of over-fitting as mentioned in Regression is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of

$$E_{\text{reg}} = E(w) + \lambda R(w). \tag{9}$$

Here $E(w)$ measures the quality (such as least squares quality) of the solution on the observed data points, $R(w)$ penalizes complex solutions, and λ is called the regularization parameter which controls the relative importance between the two. This regularized formulation is sometimes called *coefficient shrinkage* as it shrinks coefficients/weights toward zero (c.f. *coefficient subset selection* formulation where the best k out of H basis functions are greedily selected). Two simple penalty terms $R(w)$ are given next, but more generally measures of curvature can also be used to penalize non-smooth functions.

Ridge regression The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} w_d^2. \tag{10}$$

Considering $E(w)$ to be in the form of (1), the regularized least squares quality function is now

$$(Xw - y)^T(Xw - y) + \lambda w^T w. \tag{11}$$

Since the additional term is a quadratic of w, the regularized objective function is still quadratic in w, thus the optimal solution is unique and can be found in closed form. As before, setting the first derivative of (11) with respect to w to zero, the optimal weight vector is in the form of

$$\partial_w E_{\text{reg}}(w) = 2X^T(Xw - y) + 2\lambda w = 0 \tag{12}$$

$$w^* = (X^TX + \lambda I)^{-1}X^Ty. \tag{13}$$

The effect of the regularization term is to put a small weight for those basis functions which are useful only in a minor way as the penalty for small weights is very small.

Lasso regression The regularization term is in the form of

$$R(w) = \sum_{d=1}^{D} |w_d|. \quad (14)$$

In contrast to ridge regression, lasso regression (Tibshirani, 1996) has no closed-form solution. In fact, the non-differentiability of the regularization term has produced many approaches. Most of the methods involve quadratic programming and recently coordinate-wise descent algorithms for large lasso problems (Friedman et al., 2007). Lasso regression leads to sparsity in w, that is, only a subset of w is nonzero, so irrelevant basis functions will be ignored.

Cross References

►Correlation Matrix
►Gaussian Processes
►Regression

Recommended Reading

Statistical textbooks and machine learning textbooks, such as Bishop (2006) among others, introduce different linear regression models. For a large variety of built-in linear regression techniques, refer to R (http://www.r-project.org/).

Bishop, C. (2006). *Pattern recognition and machine learning*. New York: Springer.

Friedman, J., Hastie, T., Hölfling, H., & Tibshirani, R. (2007). Pathwise coordinate optimization. *Annals of statistics, 1*(2): 302–332.

Golub, G.H., & Van Loan, C.F. (1996). *Matrix computations* (3rd ed.). Baltimore: John Hopkins University Press.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P. (1992). *Numerical recipes in C: The art of scientific computing* (2nd ed.). Cambridge: Cambridge University Press. ISBN 0-521-43108-5.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, **58**, 267–288.

Linear Regression Trees

►Model Trees

Linear Separability

Two classes are linearly separable if there exists a hyperplane that separates the data for each of the classes.

Cross References

►Perceptrons
►Support Vector Machines

Link Analysis

►Link Mining and Link Discovery

Link Mining and Link Discovery

Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Link analysis; Network analysis

Definition

Many domains of interest today are best described as a linked collection of interrelated objects. Datasets describing these domains may describe homogeneous networks, in which there is a single-object type and link type, or richer, heterogeneous networks, in which there may be multiple object and link types (and possibly other semantic information). Examples of homogeneous networks include social networks, such as people connected by friendship links, or the WWW, a collection of linked web pages. Examples of heterogeneous networks include those in medical domains describing patients, diseases, treatments and contacts, or bibliographic domains describing publications, authors, and venues. *Link mining* refers to data mining techniques that explicitly consider these links when building predictive or descriptive models of the linked data. Commonly addressed link mining tasks include collective classification, object ranking, group detection, link prediction, and subgraph discovery. Additional important

components include entity resolution, and other data cleaning and data mapping operations.

Motivation and Background

"Links," or more generically "relationships," among data instances are ubiquitous. These links often exhibit patterns that can indicate properties of the data instances such as the importance, rank, or category of the instances. In some cases, not all links will be observed; therefore, we may be interested in predicting the existence of links between instances. Or, we may be interested in identifying unusual or anomalous links. In other domains, where the links are evolving over time, our goal may be to predict whether a link will exist in the future, given the previously observed links. By taking links into account, more complex patterns may be discernable as well. This observation leads to other challenges focused on discovering substructures, such as communities, groups, or common subgraphs. In addition, links can also help in the process of ►entity resolution, or figuring out when two instance references refer to the same underlying entity.

Link mining is a newly emerging research area that is at the intersection of the work in link analysis (Feldman, 2002; Jensen & Goldberg, 1998) hypertext and web mining (Chakrabarti, 2002), ►relational learning and ►inductive logic programming (Raedt, 2008), and ►graph mining (Cook & Holder, 2000). We use the term link mining to put a special emphasis on the links – moving them up to first-class citizens in the data analysis endeavor.

Theory/Solution

Traditional data mining algorithms such as ►association rule mining, market basket analysis, and cluster analysis commonly attempt to find patterns in a dataset characterized by a collection of independent instances of a single relation. This is consistent with the classical statistical inference problem of trying to identify a model given an independent, identically distributed (IID) sample. One can think of this process as learning a model for the node attributes of a homogeneous graph while ignoring the links between the nodes.

A key emerging challenge for data mining is tackling the problem of mining richly structured, heterogeneous datasets. These kinds of datasets are commonly described as networks or graphs. The domains often consist of a variety of object types; the objects can be linked in a variety of ways. Thus, the graph may have different node and edge (or hyperedge) types. Naively applying traditional statistical inference procedures, which assume that instances are independent, can lead to inappropriate conclusions about the data (Jensen, 1999). Care must be taken that potential correlations due to links are handled appropriately. In fact, object linkage is knowledge that should be exploited. This information can be used to improve the predictive accuracy of the learned models: attributes of linked objects are often correlated, and links are more likely to exist between objects that have some commonality. In addition, the graph structure itself may be an important element to include in the model. Structural properties such as degree and connectivity can be important indicators.

Data Representation

While data representation and feature selection are significant issues for traditional machine learning algorithms, data representation for linked data is even more complex. Consider a simple example from Singh et al. (2005) of a social network describing actors and their participation in events. Such social networks are commonly called *affiliation networks* (Wasserman & Faust, 1994), and are easily represented by three tables representing the actors, the events, and the participation relationships. Even this simple structure can be represented as several distinct graphs. The most natural representation is a bipartite graph, with a set of actor nodes, a set of event nodes, and edges that represent an actor's participation in an event. Other representations may enable different insights and analysis. For example, we may construct a network in which the actors are nodes and edges correspond to actors who have participated in an event together. This representation allows us to perform a more actor-centric analysis. Alternatively, we may represent these relations as a graph in which the events are nodes, and events are linked if they have an actor in common. This representation may allow us to more easily see connections between events.

This flexibility in the representation of a graph arises from a basic graph representation duality. This duality is illustrated by the following simple example: Consider

a data set represented as a simple $G = (\mathbf{O}, \mathbf{L})$, where $\mathbf{O}$ is the set of objects (i.e., the nodes or vertices) and $\mathbf{L}$ is the set of links (i.e., the edges or hyperedges). The graph $G(\mathbf{O}, \mathbf{L})$ can be transformed into a new graph $G'(\mathbf{O}', \mathbf{L}')$, in which the links l_i, l_j in G are objects in G' and there exists an link between $o_i, o_j \in \mathbf{O}'$ if and only if l_i and l_j share an object in G. This basic graph duality illustrates one kind of simple data representation transformation. For graphs with multiple node and edge types, the number of possible transformations becomes immense. Typically, these reformulations are not considered as part of the link mining process. However, the representation chosen can have a significant impact on the quality of the statistical inferences that can be made. Therefore, the choice of an appropriate representation is actually an important issue in effective link mining, and is often more complex than in the case where we have IID data instances.

Link Mining Tasks

Link mining puts a new twist on some classic data mining tasks, and also poses new problems. One way to understand the different types of learning and inference problems is to categorize them in terms of the components of the data that are being targeted. Table 1 gives a simple characterization. Note that for the object-related tasks, even though we are concerned with classifying, clustering, consolidating, or ranking the objects, we will be exploiting the links. Similarly for link-related tasks, we can use information about the objects that participate in the links, and their links to other objects and so on.

Link Mining and Link Discovery. Table 1 A simple categorization of different link mining tasks

1. Object-related tasks
a. Object classification (collective classification)
b. Object clustering (group detection)
c. Object consolidation (entity resolution)
d. Object ranking
2. Link-related tasks
a. Link labeling/classification
b. Link prediction
c. Link ranking
3. Graph-related tasks
a. Subgraph discovery
b. Graph classification

In addition, because of the underlying link structure, link mining affords the opportunity for inferences and predictions to be *collective* or dependent on one another. The simplest example of this is in collective classification, where the inferred label of one node can depend on the inferred label of its neighbors. There are a variety of ways of modeling and exploiting this dependence. Methods include performing joint inference in the appropriate probabilistic model, use of information diffusion models, constructing and optimizing the appropriate structured prediction using a max margin approach, and others.

Additional information on different link mining subtasks is provided in separate entries on *collective classification*, *entity resolution*, *group detection*, and *link prediction*. Related problems and techniques can be found in the entries on *relational learning*, *graph mining*, and *inductive logic programming*.

Cross References

- ▶Collective Classification
- ▶Entity Resolution
- ▶Graph Clustering
- ▶Graph Mining
- ▶Group Detection
- ▶Inductive Logic Programming
- ▶Link Prediction
- ▶Relational Learning

Recommended Reading

Chakrabarti, S. (2002). *Mining the web*. San Francisco, CA: Morgan Kaufman.

Cook, D. J., & Holder, L. B. (2000). Graph-based data mining. *IEEE Intelligent Systems, 15*(2), 32–41. ISSN 1094-7167. doi: http://dx.doi.org/10.1109/5254.850825.

Feldman, R. (2002). Link analysis: Current state of the art. In *Proceedings of the KDD '02*, Edmonton, Alberta, Canada.

Jensen, D. (1999). Statistical challenges to inductive inference in linked data. In *Seventh international workshop on artificial intelligence and statistics*, Fort Lauderdale, FL. San Francisco, CA: Morgan Kaufmann.

Jensen, D., & Goldberg, H. (1998). *AAAI fall symposium on AI and link analysis*, Orlando, FL. Menlo Park, CA: AAAI Press.

Raedt, L. D., (Ed.). (2008). *Logical and relational learning*. Berlin: Springer.
Singh, L., Getoor, L., & Licamele, L. (2005). Pruning social networks using structural properties and descriptive attributes. In *International conference on data mining, 2005*. Houston, TX: IEEE Computer Society.
Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications*. Cambridge: Cambridge University Press.

Link Prediction

Galileo Namata, Lise Getoor
University of Maryland, College Park, Maryland, USA

Synonyms

Edge prediction; Relationship extraction

Definition

Many datasets can naturally be represented as graph where nodes represent instances and links represent relationships between those instances. A fundamental problem with these types of data is that the link information in the graph maybe of dubious quality; links may incorrectly exist between unrelated nodes and links may be missing between two related nodes. The goal of link prediction is to predict the existence of incorrect or missing links between the nodes of the graph.

Theory/Solution

Inferring the existences of edges between nodes in a graph has traditionally been referred to as *link prediction* (Liben-Nowell & Kleinberg, 2003; Taskar, Wong, Abbeel, & Koller, 2003). Link prediction is a challenging problem that has been studied in various guises in different domains. For example, in social network analysis, there is work on predicting friendship links (Zheleva, Getoor, Golbeck, & Kuter, 2008), event participation links (i.e., coauthorship (O'Madadhain, Hutchins, & Smyth, 2005)), communication links (i.e., email (O'Madadhain et al., 2005)), and links representing semantic relationships (i.e., advisor-of (Taskar et al., 2003), subordinate-manager (Diehl, Namata, & Getoor, 2007)). In bioinformatics, there is interest in predicting the existence of edges representing physical protein–protein interactions (Yu, Paccanaro, Trifonov, & Gerstein, 2006; Szilagyi et al., 2005), domain–domain interactions (Deng, Mehta, Sun, & Chen, 2002), and regulatory interactions (Albert et al., 2007). Similarly, in computer network systems there is work in inferring unobserved connections between routers, as well as inferring relationships between autonomous systems and service providers (Spring, Wetherall, & Anderson, 2004). There is also work on using link prediction to improve recommender systems (Farrell, Campbell, & Myagmar, 2005), Web site navigation (Zhu, 2003), surveillance (Huang & Lin, 2008), and automatic document cross referencing (Milne & Witten, 2008).

We begin with some basic definitions and notation. We refer to the set of possible edges in a graph as *potential edges*. The set of potential edges depends on the graph type, and how the edges for the graph are defined. For example, in a directed graph, the set of potential edges consists of all edges $e = (v_1, v_2)$ where v_1 and v_2 are any two nodes V in the graph (i.e., the number of potential edges is $|V| \times |V|$). In an undirected bipartite graph with two subsets of nodes $(V_1, V_2 \in V)$, while the edges still consist of a pair of nodes, $e = (v_1, v_2)$, there is an added condition such that one node must be from V_1 and the other node must be from V_2; this results in $|V_1| \times |V_2|$ potential edges. Next, we refer to set of "true" edges in a graph as *positive edges*, and we refer to the "true" non-edges in a graph (i.e., pairs of nodes without edges between them) as *negative edges*. For a given graph, typically we only have information about a subset of the edges; we refer to this set as the *observed* edges. The observed edges can include both positive and negative edges, though in many formulations there is an assumption of positive-only information. We can view link prediction as a probabilistic inference problem, where the evidence includes the observed edges, the attribute values of the nodes involved in the potential edge, and possibly other information about the network, and for any unobserved, potential edge, we want to compute the probability of it existing. This can be reframed as a binary classification problem by choosing some probability threshold, and concluding that potential edges with existence probability above the threshold are true edges, and those below the threshold are considered false edges (more complex schemes are possible as well). For noisy and incomplete networks, we use terminology from the machine learning literature and refer to an edge that is inferred to exists

and is a true edge in the graph as a *true positive edge*, an edge that should exist but is not inferred as a *false negative edge*, an edge that should not exist and is not inferred as a *true negative edge*, and an edge that should not exist but is incorrectly inferred to exist as a *false positive edge*.

One of the early and simple formulations of the link prediction problem was proposed by Liben-Nowell and Kleinberg (2003). They proposed a temporal prediction problem defined over a dynamic network where given a graph $G_t(V_t, E_t)$ at time t, the problem is to infer the set of edges at the next time step $t + 1$. More formally, the objective is to infer a set of edges E_{new} where $E_{t+1} = E_t \bigcup E_{new}$. We use a more general definition of link prediction proposed by Taskar et al. (2003) where given a graph G and the set of potential edges in G, denoted $P(G)$, the problem of link prediction is to predict for all $p \in P(G)$ whether p exists or does not exists, remaining agnostic on whether G is a noisy graph with missing edges or a snapshot of a dynamic graph at a particular time point.

Approaches

In this section, we discuss the two general categories of the current link prediction models: topology-based approaches and node attribute-based approaches. Topology-based approaches are methods that rely solely on the topology of the network to infer edges. Node attribute-based approaches make predictions based on the attribute values of the nodes incident to the edges. In addition, there are models that make use of both structure and attribute values.

Topology-Based Approaches

A number of link prediction models have been proposed, which rely solely on the topology of the network. These models typically rely on some notion of structural proximity, where nodes that are close are likely to share an edge (e.g., sharing common neighbors, nodes with a small shortest path distance between). The earliest topological approach for link prediction was proposed by Liben-Nowell and Kleinberg (2003). In this work, Liben-Nowell and Kleinberg proposed various structure based similarity scores and applied them over the unobserved edges of an undirected graph. They then use a threshold k, and only predict edges with the top k scores as existing. A variety of similarity scores were proposed, given two nodes v_1 and v_2, including graph distance (the length of the shortest path between v_1 and v_2), common neighbors (the size of the intersection of the sets of neighbors of v_1 and v_2), and more complex measures such as the Katz measure, (the sum of the lengths of the paths between v_1 and v_2 exponentially damped by length to count short paths more heavily). Evaluating over a coauthorship network, the best performing proximity score measure was the Katz measure, however the simple measures, which rely only on the intersection of the set of nodes adjacent to both nodes, performed surprisingly well. A related approach was proposed by Yu et al. (2006), which applies the link prediction problem to predicting missing protein–protein interactions (PPI) from PPI networks generated by high throughput methods. This work assumes that interacting proteins tend to form a clique. Thus, missing edges can be predicted by predicting the existence of edges that will create cliques in the network. More recent work by Clauset, Moore, and Newman (2008) has tried to go beyond predicting edges between neighboring nodes. In their problem domain of food webs, for example, pairs of predators often prey on a shared prey species but rarely prey on each other. Thus, in these networks, predicting "predator–prey" edges need to go beyond proximity. For this, they propose a "hierarchical random graph" approach, which fits a hierarchical model to all possible dendrograms of a given network. The model is then used to calculate the likelihood of an edge existing in the network.

Node Attribute-Based Approaches

Although topology is useful in link prediction, topology-based approaches ignore an important source of information in networks, the attributes of nodes. Often there are correlations in the attributes of nodes that share an edge with each other. One approach that exploits this correlation was proposed by Taskar et al. (2003). In their approach, Taskar et al. (2003) applied the relational Markov network (RMN) framework to link prediction to predicting the existence and class of edges between Web sites. They exploit the fact that certain links can only exist between nodes of the appropriate type. For example, an "advisor" edge can only exist between student and faculty.

Another approach that uses node attributes was proposed by Popescul and Ungar (2003). In that approach, they used a structured ▶logistic regression model over learned relational features to predict citation edges in a citation network. Their relational features are built over attributes such as the words used in the paper nodes. O'Madadhain et al. (2005) also approached an attribute based approach, constructing local conditional probability models based on the attributes such as node attribute similarity, topic distribution, and geographical location in predicting "co-participation" edges in an email communication network. More recently, there is work on exploiting other node attributes like the group membership of the nodes. Zheleva et al. (2008) showed that membership in family groups are very useful in predicting friendship links in social networks. Similarly, Sprinzak, Altuvia, & Margalit (2006) showed that using protein complex information can be useful in predicting protein–protein interactions. Finally, we note that in link prediction, as in classification, the quality of predictions can be improved by making the predictions collectively. Aside from the relational Markov network approach by Taskar et al. (2003) mentioned earlier, Markov logic networks (Richardson & Domingos, 2006) and probabilistic relational models (Getoor, Friedman, Koller, & Taskar, 2003) have also been proposed for link prediction and are capable of performing joint inference.

Issues

There are a number of challenges that make link prediction very difficult. The most difficult challenge is the large class skew between the number of edges that exist and the number of edges that do not. To illustrate, consider directed graph denoted by $G(V, E)$. While the number of edges $|E|$ is often $O(|V|)$, the number of edges that do not exist is often $O(|V|^2)$. Consequently, the prior probability edge existence is very small. This causes many supervised models, which naively optimize for accuracy, to learn a trivial model, which always predicts that a link does not exist. A related problem in link prediction is the large number of edges whose existence must be considered. The number of potential edges is $O(|V|^2)$ and this limits the size of the data sets that can be considered.

In practice, there are general approaches to addressing these issues either prior to or during the link prediction. With both large class skew and number of edges to contend with, the general approach is to make assumptions that reduce the number of edges to consider. One common way to do this is to partition the set of nodes where we only consider potential edges between nodes of the same partition; edges between partitions are not explicitly modeled, but are assumed not to exist. This is useful in many domains where there is some sort of natural partition among the nodes available (e.g., geography in social networks, location of proteins in a cell), which make edges across partitions unlikely. Another way is to define some simple, computationally inexpensive distance measure such that only edges whose nodes are within some distance are considered.

Another practical issue in link prediction is that while real-world data often indicates which edges exist (positive examples), the edges which do not exist (negative examples) are rarely annotated for use by link prediction models. In bioinformatics, for example, the protein–protein interaction network of yeast, the most and annotated studied organism, is annotated with thousands of observed edges (physical interactions) between the nodes (proteins) gathered from numerous experiments. There are currently, however, no major datasets available that indicate which proteins definitely do not physically interact. This is an issue not only in creating and learning models for link prediction, but is also an issue evaluating them. Often, it is unclear whether a predicted edge which is not in our ground truth data is an incorrectly predicted edge or an edge resulting from incomplete data.

Related Problems

In addition to the definition of link prediction discussed above, it is also important to mention three closely related problems: *link completion*, *leak detection*, and *anomalous link discovery*, whose objectives are different but very similar to link prediction. Link completion (Chaiwanarom & Lursinsap, 2008; Goldenberg, Kubica, Komarek, Moore, & Schneider, 2003) and leak detection (Balasubramanyan, Carvalho, & Cohen, 2009; Carvalho & Cohen, 2007), are a variation of link prediction over hypergraphs. A hypergraph is a graph where the edges (known as hyperedges) can connect any number

of nodes. For example, in a hypergraph representing an email communication network, a hyperedge may connect nodes representing email addresses that are recipients of a particular email communication. In link completion, given the set of nodes that participate in a particular hyperedge, the objective is to infer nodes that are missing. For the email communication network example, link completion may involve inferring which email addresses need to be added to the recipients list of an email communication. Conversely, in leak detection, given the set of nodes participating in a particular hyperedge, the objective is to infer which nodes should not be part of that hyperedge. For example, in email communications, leak detection will attempt to infer which email address nodes are incorrectly part of the hyperedge representing the recipient list of the email communication.

The last problem, anomalous link discovery (Huang & Zeng, 2006; Rattigan & Jensen, 2005), has been proposed as an alternate task to link prediction. As with link completion, the existence of the edges are assumed to be observed, and the objective is to infer which of the observed links are anomalous or unusual. Specifically, anomalous link discovery identifies which links are statistically improbable with the idea that these may be of interest for those analyzing the network. Rattigan and Jensen (2005) show that some methods that perform poorly for link prediction can still perform well for anomalous link discovery.

Cross References

►Graph Mining
►Statistical Relational Learning

Recommended Reading

Albert, R., DasGupta, B., Dondi, R., Kachalo, S., Sontag, E., Zelikovsky, A., et al. (2007). A novel method for signal transduction network inference from indirect experimental evidence. *Journal of Computational Biology, 14*, 407–419.

Balasubramanyan, R., Carvalho, V. R., & Cohen, W. (2009). Cutonce recipient recommendation and leak detection in action. In *Workshop on enhanced messaging*.

Carvalho, V. R., & Cohen, W. W. (2007). Preventing information leaks in email. In *SIAM conference on data mining*.

Chaiwanarom, P., & Lursinsap, C. (2008). Link completion using prediction by partial matching. In *International symposium on communications and information technologies*.

Clauset, A., Moore, C., & Newman, M. E. J. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature, 453*, 98.

Deng, M., Mehta, S., Sun, F., & Chen, T. (2002). Inferring domain-domain interactions from protein-protein interactions. *Genome Research, 12*(10), 1540–1548.

Diehl, C., Namata, G. M., & Getoor, L. (2007). Relationship identification for social network discovery. In *Proceedings of the 22nd national conference on artificial intelligence*.

Farrell, S., Campbell, C., & Myagmar, S. (2005). Relescope: An experiment in accelerating relationships. In *Extended abstracts on human factors in computing systems*.

Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2003). Learning probabilistic models of link structure. *Machine Learning, 3*, 679–707.

Goldenberg, A., Kubica, J., Komarek, P., Moore, A., & Schneider, J. (2003). A comparison of statistical and machine learning algorithms on the task of link completion. In *Conference on knowledge discovery and data mining, Workshop on link analysis for detecting complex behavior*.

Huang, Z., & Lin, D. K. J. (2008). The time-series link prediction problem with applications in communication surveillance. *Informs Journal on Computing, 21*, 286–303.

Huang, Z., & Zeng, D. D. (2006). A link prediction approach to anomalous email detection. In *IEEE International conference on systems, man, and cybernetics*, Taipei, Taiwan.

Liben-Nowell, D., & Kleinberg, J. (2003). The link prediction problem for social networks. In *International conference on information and knowledge management*.

Milne, D., & Witten, I. H. (2008). Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on information and knowledge management*.

O'Madadhain, J., Hutchins, J., & Smyth, P. (2005). Prediction and ranking algorithms for event-based network data. *SIGKDD Explorations Newsletter, 7*(2), 23–30.

Popescul, A., & Ungar, L. H. (2003). Statistical relational learning for link prediction. In *International joint conferences on artificial intelligence workshop on learning statistical models from relational data*.

Rattigan, M. J., & Jensen, D. (2005). The case for anomalous link discovery. *SIGKDD Explorations Newsletter, 7*, 41–47.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*, 107–136.

Spring, N., Wetherall, D., & Anderson, T. (2004). Reverse engineering the internet. *SIGCOMM Computer Communication Review, 34*(1), 3–8.

Sprinzak, E., Altuvia, Y., & Margalit, H. (2006). Characterization and prediction of protein-protein interactions within and between complexes. *Proceedings of the National Academy of Sciences, 103*(40), 14718–14723.

Szilagyi, A., Grimm, V., Arakaki, A. K., & Skolnick, J. (2005). Prediction of physical protein-protein interactions. *Physical Biology, 2*(2), S1–S16.

Taskar, B., Wong, M.-F., Abbeel, P., & Koller, D. (2003). Link prediction in relational data. In *Advances in neural information processing systems*.

Yu, H., Paccanaro, A., Trifonov, V., & Gerstein, M. (2006). Predicting interactions in protein networks by completing defective cliques. *Bioinformatics, 22*(7), 823–829.

Zheleva, E., Getoor, L., Golbeck, J., & Kuter, U. (2008). Using friendship ties and family circles for link prediction. In *2nd ACM SIGKDD workshop on social network mining and analysis*.

Zhu, J. (2003). *Mining web site link structure for adaptive web site navigation and search*. Ph.D. thesis, University of Ulster at Jordanstown, UK.

Link-Based Classification

►Collective Classification

Liquid State Machine

►Reservoir Computing

Local Distance Metric Adaptation

Synonyms

Supersmoothing; Nonstationary kernels; Kernel shaping

Definition

In learning systems with kernels, the shape and size of a kernel plays a critical role for accuracy and generalization. Most kernels have a distance metric parameter, which determines the size and shape of the kernel in the sense of a Mahalanobis distance. Advanced kernel learning tune every kernel's distance metric individually, instead of turning one global distance metric for all kernels.

Cross References

►Locally Weighted Regression for Control

Local Feature Selection

►Projective Clustering

Locality Sensitive Hashing Based Clustering

Xin Jin, Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, IL, USA

The basic idea of the LSH (Gionis, Indyk, & Motwani, 1999) technique is using multiple hash functions to hash the data points and guarantee that there is a high probability of collision for points which are close to each other and low collision probability for dissimilar points. LSH schemes exist for many distance measures, such as Hamming norm, L_p norms, cosine distance, earth movers distance (EMD), and Jaccard coefficient.

In LSH, define a family $H = \{h : S \to U\}$ as locality-sensitive, if for any a, the function $p(t) = Pr_H[h(a) = h(b) : ||a - b|| = x]$ is decreasing in x. Based on this definition, the probability of collision of points a and b is decreasing with their distance.

Although LSH was originally proposed for approximate nearest neighbor search in high dimensions, it can be used for clustering as well (Das, Datar, Garg, & Rajaram, 2007; Haveliwala, Gionis, & Indyk, 2000). The buckets could be used as the bases for clustering. Seeding the hash functions several times can help getting better quality clustering.

Recommended Reading

Das, A. S., Datar, M., Garg, A., & Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (pp. 271–280). New York: ACM.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th international conference on very large data bases* (pp. 518–529). San Francisco: Morgan Kaufmann Publishers.

Haveliwala, T. H., Gionis, A., & Indyk, P. (2000). Scalable techniques for clustering the web (extended abstract). In *Proceedings of the third international workshop on the web and databases* (pp. 129–134). Stanford, CA: Stanford University.

Locally Weighted Learning

►Locally Weighted Regression for Control

Locally Weighted Regression for Control

Jo-Anne Ting[1], Sethu Vijayakumar[1,2], Stefan Schaal[2,3]
[1]University of Edinburgh
[2]University of Southern California
[3]ATR Computational Neuroscience Labs

Synonyms

Kernel shaping; Lazy learning; Local distance metric adaptation; Locally weighted learning; LWPR; LWR; Nonstationary kernels supersmoothing

Definition

This article addresses two topics: ►learning control and locally weighted regression.

►Learning control refers to the process of acquiring a control strategy for a particular control system and a particular task by trial and error. It is usually distinguished from adaptive control (Aström & Wittenmark, 1989) in that the learning system is permitted to fail during the process of learning, resembling how humans and animals acquire new movement strategies. In contrast, adaptive control emphasizes single trial convergence without failure, fulfilling stringent performance constraints, e.g., as needed in life-critical systems like airplanes and industrial robots.

Locally weighted regression refers to ►supervised learning of continuous functions (otherwise known as function approximation or ►regression) by means of spatially localized algorithms, which are often discussed in the context of ►kernel regression, ►nearest neighbor methods, or ►lazy learning (Atkeson, Moore, & Schaal, 1997). Most regression algorithms are global learning systems. For instance, many algorithms can be understood in terms of minimizing a global ►loss function such as the expected sum squared error:

$$J_{\text{global}} = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \mathbf{y}_i\right)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{t}_i - \phi\left(\mathbf{x}_i\right)^T\boldsymbol{\beta}\right)^2\right] \tag{1}$$

where $E[\cdot]$ denotes the expectation operator, $\mathbf{t}_i$ the noise-corrupted target value for an input $\mathbf{x}_i$, which is expanded by basis functions into a basis function vector $\phi(\mathbf{x}_i)$, and $\boldsymbol{\beta}$ the vector of (usually linear) regression coefficients. Classical feedforward ►neural networks, ►radial basis function networks, ►mixture models, or ►Gaussian Process regression are all global function approximators in the spirit of Eq. (1).

In contrast, local learning systems split up conceptually the cost function into multiple independent local function approximation problems, using a cost function such as the one below:

$$\begin{aligned} J_{\text{global}} &= E\left[\frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \\ &= \frac{1}{2}\sum_{k=1}^{K} E\left[\sum_{i=1}^{N} w_{k,i}\left(\mathbf{t}_i - \mathbf{x}_i^T\boldsymbol{\beta}_k\right)^2\right] \end{aligned} \tag{2}$$

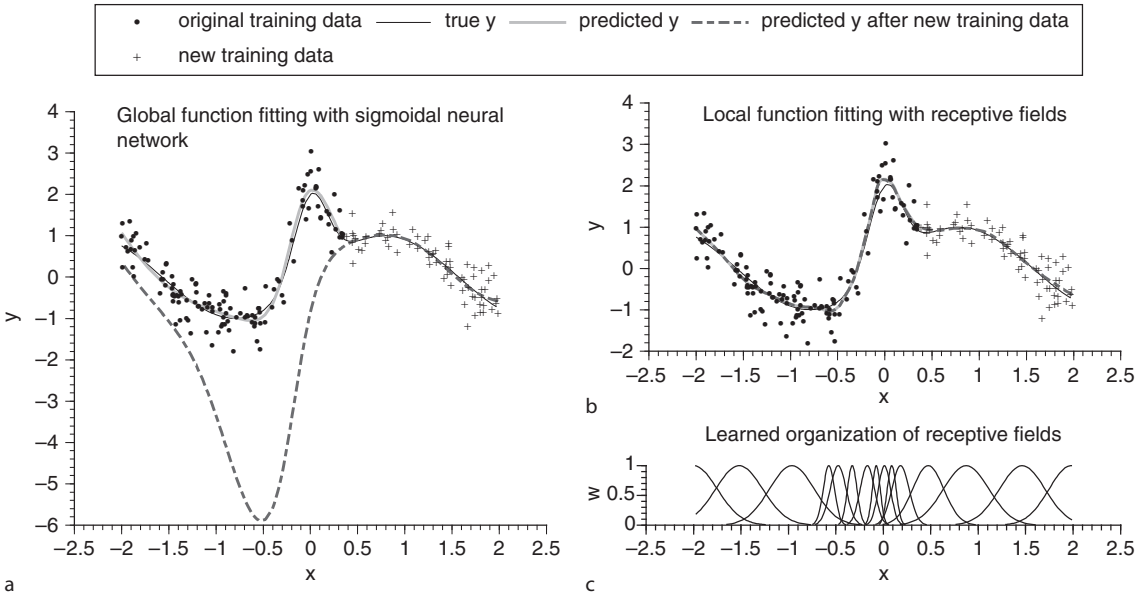
Motivation and Background

Figure 1 illustrates why locally weighted regression methods are often favored over global methods when it comes to learning from incrementally arriving data, especially when dealing with nonstationary input distributions. The figure shows the division of the training data into two sets: the "original training data" and the "new training data" (in dots and crosses, respectively).

Initially, a sigmoidal ►neural network and a locally weighted regression algorithm are trained on the "original training data," using 20% of the data as a cross-validation set to assess convergence of the learning. In a second phase, both learning systems are trained solely on the "new training data" (again with a similar cross-validation procedure), but without using any data from the "original training data." While both algorithms generalize well on the "new training data," the global learner incurred catastrophic interference, unlearning what was learned initially, as seen in Fig. 1a, b shows that the locally weighted regression algorithm does not have this problem since learning (along with ►generalization) is restricted to a local area.

Appealing properties of locally weighted regression include the following:

- Function approximation can be performed incrementally with nonstationary input and output distributions and without significant danger of interference. Locally weighted regression can provide ►posterior probability distributions, offer confidence assessments, and deal with heteroscedastic data.
- Locally weighted learning algorithms are computationally inexpensive to compute. It is well suited for online computations (e.g., for ►online and ►incremental learning) in the fast control loop of a robot – typically on the order of 100–1000 Hz.
- Locally weighted regression methods can implement continual learning and learning from large amounts of data without running into severe computational problems on modern computing hardware.
- Locally weighted regression is a nonparametric method (i.e., it does not require that the user determine *a priori* the number of local models in the learning system), and the learning systems grows with the complexity of the data it tries to model.
- Locally weighted regression can include ►feature selection, ►dimensionality reduction, and ►Bayesian inference – all which are required for robust statistical inference.



Locally Weighted Regression for Control. Figure 1. Function approximation results for the function $y = \sin(2x) + 2\exp(-16x^2) + N(0, 0.16)$ with (a) a sigmoidal neural network; (b) a locally weighted regression algorithm (note that the data traces "true y," "predicted y," and "predicted y after new training data" largely coincide); and (c) the organization of the (Gaussian) kernels of (b) after training. See Schaal and Atkeson (1998) for more details

- Locally weighted regression works favorably with locally linear models (Hastie & Loader, 1993), and local linearizations are of ubiquitous use in control applications.

Background

Returning to Eqs. (1) and (2), the main differences between both equations are listed below:

(i) A weight $w_{i,k}$ is introduced that focuses the function approximation on only a small neighborhood around a point of interest $\mathbf{c}_k$ in input space (see Eq. 3 below).
(ii) The cost function is split into K independent optimization problems.
(iii) Due to the restricted scope of the function approximation problem, we do not need a nonlinear basis function expansion and can, instead, work with simple local functions or local polynomials (Hastie & Loader, 1993).

The weights $w_{k,i}$ in Eq. (2) are typically computed from some ▶kernel function (Atkeson, Moore, & Schaal, 1997) such as a squared exponential kernel

$$w_{k,i} = \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{c}_k)^T \mathbf{D}_k (\mathbf{x}_i - \mathbf{c}_k)\right) \quad (3)$$

with $\mathbf{D}_k$ denoting a positive semidefinite distance metric and $\mathbf{c}_k$ the center of the kernel. The number of kernels K is not finite. In many local learning algorithms, the kernels are never maintained in memory. Instead, for every query point $\mathbf{x}_q$, a new kernel is centered at $\mathbf{c}_k = \mathbf{x}_q$, and the localized function approximation is solved with weighted ▶regression techniques (Atkeson et al., 1997).

Locally weighted regression should not be confused with mixture of experts models (Jordan & Jacobs, 1994). ▶Mixture models are *global* learning systems since the experts compete globally to cover training data. Mixture models address the ▶bias-variance dilemma (Intuitively, the ▶bias-variance dilemma addresses how many parameters to use for a function approximation problem to find an optimal balance between ▶overfitting and oversmoothing of the training data) by finding the right number of local experts. Locally weighted regression addresses the ▶bias-variance dilemma in a local way by finding the

optimal distance metric for computing the weights in the locally weighted regression (Schaal & Atkeson, 1998). We describe some algorithms to find $\mathbf{D}_k$ next.

Structure of Learning System

For a locally linear model centered at the query point $\mathbf{x}_q$, the regression coefficients would be

$$\boldsymbol{\beta}_q = \left(\mathbf{X}^T \mathbf{W}_q \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W}_q \mathbf{t} \tag{4}$$

where $\mathbf{X}$ is a matrix that has all training input data points in its rows (with a column of 1s added in the last column for the offset parameter in ►linear regression). $\mathbf{W}_q$ is a diagonal matrix with the corresponding weights for all data points, computed from Eq. (3) with $\mathbf{c}_k = \mathbf{x}_q$, and $\mathbf{t}$ is the vector of regression targets for all training points. Such a "compute-the-prediction-on-the-fly" approach is often called lazy learning (The approach is "lazy" because the computational of a prediction is deferred until the last moment, i.e., when a prediction is needed) and is a memory-based learning system where all training data is kept in memory for making predictions.

Alternatively, kernels can be created as needed to cover the input space, and the sufficient statistics of the weighted regression are updated incrementally with recursive least squares (Schaal & Atkeson, 1998). This approach does not require storage of data points in memory. Predictions of neighboring local models can be blended, improving function fitting results in the spirit of committee machines.

Memory-Based Locally Weighted Regression (LWR)

The original locally weighted regression algorithm was introduced by Cleveland (1979) and popularized in the machine learning and learning control community by Atkeson (1989). The algorithm is largely summarized by Eq. (4) (for algorithmic pseudo-code, see (Schaal, Atkeson, & Vijayakumar, 2002)):

- All training data is collected in the matrix $\mathbf{X}$ and the vector $\mathbf{t}$ (For simplicity, only functions with a scalar output are addressed. Vector-valued outputs can be learned either by fitting a separate learning system for each output or by modifying the algorithms to fit multiple outputs (similar to multi-output linear regression)).
- For every query point $\mathbf{x}_q$, the weighting kernel is centered at the query point.
- The weights are computed with Eq. (3).
- The local regression coefficients are computed according to Eq. (4).
- A prediction is formed with $y_q = \left[\mathbf{x}_q^T \; 1\right] \boldsymbol{\beta}_q$.

As in all kernel methods, it is important to optimize the kernel parameters in order to get optimal function fitting quality. For LWR, the critical parameter determining the ►bias-variance tradeoff is the distance metric $\mathbf{D}_q$. If the kernel is too narrow, it starts fitting noise. If it is too broad, oversmoothing will occur. $\mathbf{D}_q$ can be optimized with leave-one-out cross-validation to obtain a *globally* optimal value, i.e., the same $\mathbf{D}_q = \mathbf{D}$ is used throughout the entire input space of the data. Alternatively, $\mathbf{D}_q$ can be *locally* optimized as a function of the query point, i.e., obtain a $\mathbf{D}_q$ as a function of the query point (as already indicated by the subscript "q"). In the recent machine learning literature (in particular, work related to kernel methods), such input dependent kernels are referred to as nonstationary kernels.

Locally Weighted Projection Regression (LWPR)

Schaal and Atkeson (1998) suggested a memoryless version of LWR in order to avoid the expensive ►nearest neighbor computations – particularly for large training data sets – of LWR and to have fast real-time (In most robotic systems, "real-time" means on the order of maximally 1–10 ms computation time, corresponding to a 1000–100 Hz control loop) prediction performance. The main ideas of the RFWR algorithm (Schaal & Atkeson, 1998) are listed below:

- Create new kernels only if no existing kernel in memory covers a training point with some minimal activation weight.
- Keep all created kernels in memory and update the weighted regression with weighted recursive least squares for new training points $\{\mathbf{x}, t\}$:

$$\begin{aligned} \boldsymbol{\beta}_k^{n+1} &= \boldsymbol{\beta}_k^n + w \mathbf{P}^{n+1} \tilde{\mathbf{x}} \left(t - \tilde{\mathbf{x}}^T \boldsymbol{\beta}_k^n\right) \\ \text{where } \mathbf{P}_k^{n+1} &= \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n \tilde{\mathbf{x}} \tilde{\mathbf{x}}^T \mathbf{P}_k^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T \mathbf{P}_k^n \tilde{\mathbf{x}}}\right) \\ \text{and } \tilde{\mathbf{x}} &= \left[\mathbf{x}^T \; 1\right]^T. \end{aligned} \tag{5}$$

- Adjust the distance metric $\mathbf{D}_q$ for each kernel with a gradient descent technique using leave-one-out cross-validation.
- Make a prediction for a query point taking a weighted average of predictions from all local models:

$$\mathbf{y}_q = \frac{\sum_{k=1}^{K} w_{q,k} \hat{\mathbf{y}}_{q,k}}{\sum_{k=1}^{K} w_{q,k}} \tag{6}$$

Adjusting the distance metric $\mathbf{D}_q$ with leave-one-out cross-validation *without* keeping all training data in memory is possible due to the PRESS residual. The PRESS residual allows the leave-one-out cross-validation error to be computed in closed form without needing to actually exclude a data point from the training data.

Another deficiency of LWR is its inability to scale well to high-dimensional input spaces since the ►covariance matrix inversion in Eq. (4) becomes severely ill-conditioned. Additionally, LWR becomes expensive to evaluate as the number of local models to be maintained increases. Vijayakumar, D'Souza and Schaal (2005) suggested local ►dimensionality reduction techniques to handle this problem. Partial least squares (PLS) regression is a useful ►dimensionality reduction method that is used in the LWPR algorithm (Vijayakumar et al., 2005). In contrast to PCA methods, PLS performs ►dimensionality reduction for ►regression, i.e., it eliminates subspaces of the input space that minimally correlate with the outputs, not just parts of the input space that have low variance.

LWPR is currently one of the best developed locally weighted regression algorithms for control (Klanke, Vijayakumar, & Schaal, 2008) and has been applied to learning control problems with over 100 input dimensions.

A Full Bayesian Treatment of Locally Weighted Regression

Ting, Kalakrishnan, Vijayakumar, and Schaal (2008) proposed a fully probabilistic treatment of LWR in an attempt to avoid cross-validation procedures and minimize any manual parameter tuning (e.g., gradient descent rates, kernel initialization, and forgetting rates). The resulting Bayesian algorithm learns the distance metric of local linear model (For simplicity, a local linear model is assumed, although local polynomials can be used as well) probabilistically, can cope with high input dimensions, and rejects data outliers automatically. The main ideas of Bayesian LWR are listed below (please see Ting (2009) for details):

- Introduce hidden variables $\mathbf{z}$ to the local linear model (as in Variational Bayesian least squares (Ting et al., 2005)) to decompose the statistical estimation problem into d individual estimation problems (where d is the number of input dimensions). The result is an iterative Expectation-Maximization (EM) algorithm that is of linear ►computational complexity in d and the number of training data samples N, i.e., $O(Nd)$.
- Associate a scalar weight w_i with each training data sample $\{\mathbf{x}_i, t_i\}$, placing a Bernoulli ►prior probability distribution over a weight *for each input dimension* so that the weights are positive and between 0 and 1:

$$w_i = \prod_{m=1}^{d} w_{im} \text{ where}$$
$$w_{im} \sim \text{Bernoulli}\,(q_{im}) \text{ for } i = 1, .., N; m = 1, .., d \tag{7}$$

where the weight w_i is decomposed into independent components in each input dimension w_{im} and q_{im} is the parameter of the Bernoulli ►probability distribution. The weight w_i indicates a training sample's contribution to the local model. An outlier will have a weight of 0 and will, thus, be automatically rejected. The formulation of q_{im} determines the shape of the weighting function applied to the local model. The weighting function q_{im} used in Bayesian LWR is listed below:

$$q_{im} = \frac{1}{1 + \left(x_{im} - x_{qm}\right)^2 h_m} \text{ for } i = 1, .., N; m = 1, .., d \tag{8}$$

where $\mathbf{x}_q \in \Re^{d \times 1}$ is the query input point and h_m is the bandwidth parameter/distance metric of the local model in the m-th input dimension (The distance metric/bandwidth is assumed to be a diagonal matrix, i.e., bandwidths in each input dimension are independent. That is to say, $\mathbf{D} = \mathbf{H}$, where $\mathbf{h}$ is the diagonal vector and h_m are the coefficients of $\mathbf{h}$).

- Place a Gamma ▶prior probability distribution over the distance metric h_m:

$$h_m \sim \text{Gamma}\,(a_{hm0}, b_{hm0}) \tag{9}$$

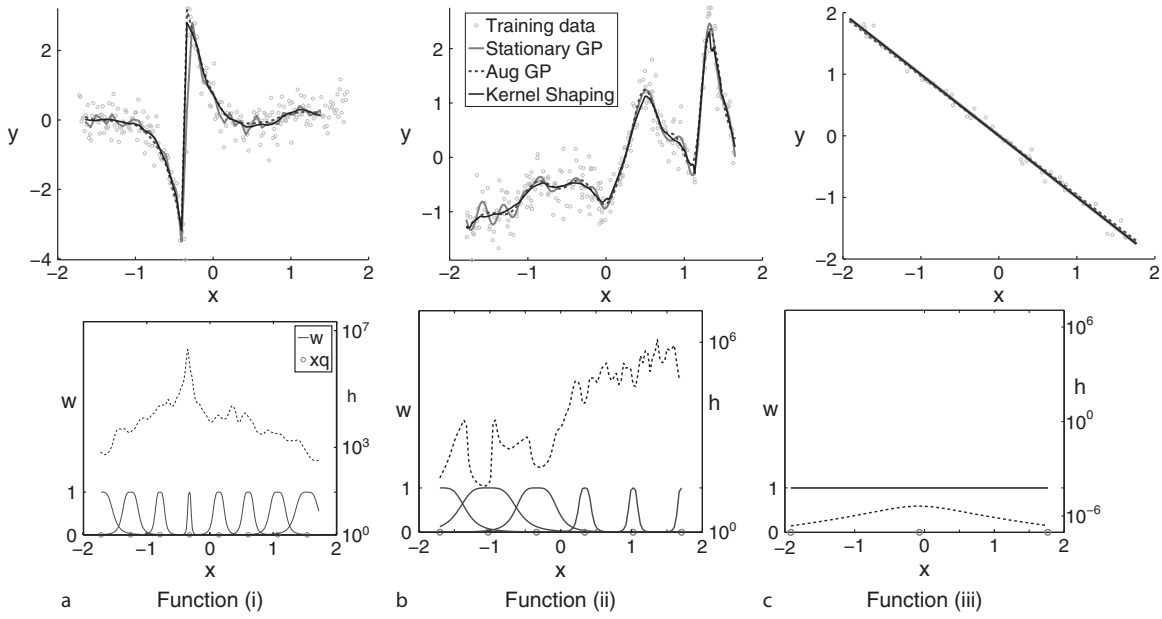
where $\{a_{hm0}, b_{hm0}\}$ are the prior parameter values of the Gamma distribution.
- Treat the model as an EM-like ▶regression problem, using ▶variational approximations to achieve analytically tractable inference of the ▶posterior probability distributions.

The initial parameters $\{a_{hm0}, b_{hm0}\}$ should be set so that the ▶prior probability distribution over h_m is uninformative and wide (e.g., $a_{hm0} = b_{hm0} = 10^{-6}$). The other ▶prior probability distribution that needs to be specified is the one over the noise variance random variable – and this is best set to reflect how noisy the data set is believed to be. More details can be found in Ting (2009).

This Bayesian method can can also be applied as general kernel shaping algorithm for global ▶kernel learning methods that are linear in the parameters (e.g., to realize nonstationary ▶Gaussian processes (Ting et al., 2008), resulting in an augmented nonstationary ▶Gaussian Process).

Figure 2 illustrates Bayesian kernel shaping's bandwidth adaptation abilities on several synthetic data sets, comparing it to a stationary ▶Gaussian Process and the augmented nonstationary ▶Gaussian Process. For the ease of visualization, the following one-dimensional functions are considered: (i) a function with a discontinuity, (ii) a spatially inhomogeneous function, and (iii) a straight line function. The data set for function (i) consists of 250 training samples, 201 test inputs (evenly spaced across the input space), and output noise with variance of 0.3025; the data set for function (ii) consists of 250 training samples, 101 test inputs, and an output signal-to-noise ratio (SNR) of 10; and the data set for function (iii) has 50 training samples, 21 test inputs, and an output SNR of 100. Figure 2 shows the predicted outputs of all three algorithms for data sets (i)–(iii). The local kernel shaping algorithm smoothes over regions where a stationary ▶Gaussian Process overfits and yet, it still manages to capture regions of highly varying curvature, as seen in Figs. 2a and 2b.

It correctly adjusts the bandwidths h with the curvature of the function. When the data looks linear, the algorithm opens up the weighting kernel so that all data samples are considered, as Fig. 2c shows.



Locally Weighted Regression for Control. Figure 2. Predicted outputs using a stationary Gaussian Process (GP), the augmented nonstationary GP and local kernel shaping on three different data sets. Figures on the bottom row show the bandwidths learned by local kernel shaping and the corresponding weighting kernels (*in dotted black lines*) for various input query points (*shown in red circles*)

From the viewpoint of ▶learning control, ▶overfitting – as seen in the ▶Gaussian Process in Fig. 2 – can be detrimental since ▶learning control often relies on extracting local linearizations to derive ▶controllers (see Applications section). Obtaining the wrong sign on a slope in a local linearization may destabilize a ▶controller.

In contrast to LWPR, the Bayesian LWR method is memory-based, although memoryless versions could be derived. Future work will also have to address how to incorporate ▶dimensionality reduction methods for robustness in high dimensions. Nevertheless, it is a first step toward a probabilistic locally weighted regression method with minimal parameter tuning required by the user.

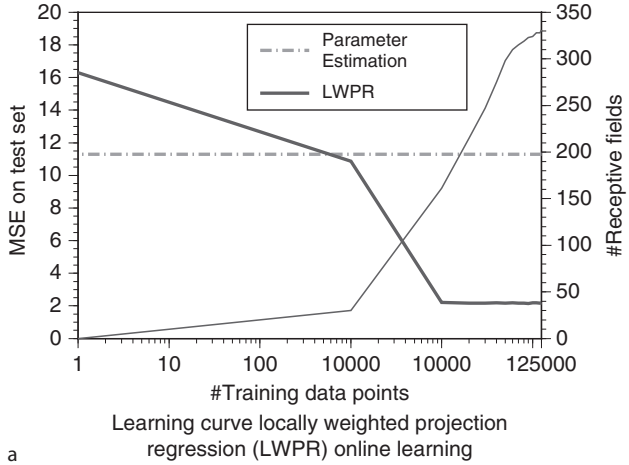
Applications

Learning Internal Models with LWPR

Learning an internal model is one of most typical applications of LWR methods for control. The model could be a forward model (e.g, the nonlinear differential equations of robot dynamics), an inverse model (e.g., the equations that predict the amount of torque to achieve a change of state in a robot), or any other function that models associations between input and output data about the environment. The models are used, subsequently, to compute a ▶controller e.g., an inverse dynamics controller similar to Eq. (12). Models for complex robots such as humanoids exceed easily a hundred input dimensions. In such high-dimensional spaces, it is hopeless to assume that a representative data set can be collected for offline training that can generalize sufficiently to related tasks. Thus, the LWR philosophy involves having a learning algorithm that can learn rapidly when entering a new part of the state space such that it can achieve acceptable ▶generalization performance almost instantaneously.

Figure 3 demonstrates ▶online learning of an inverse dynamics model for the elbow joint (cf. Eq. 12) for a Sarcos Dexterous Robot Arm. The robot starts with no knowledge about this model, and it tracks some randomly varying desired trajectories with a proportional-derivative (PD) controller. During its movements, training data consisting of tuples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \tau)$ – which model a mapping from joint position, joint velocities and joint accelerations $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ to motor torques τ – are collected (at about every 2 ms). Every data point is used to train a LWPR function approximator, which generates a feedforward command for the controller. The ▶learning curve is shown in Fig. 3a.

Using a test set created beforehand, the model predictions of LWPR are compared every 1,000 training points with that of a parameter estimation method. The parameter estimation approach fits the minimal number of parameters to an analytical model of the robot dynamics under an idealized rigid body dynamics (RBD) assumptions, using all training data (i.e., not incrementally). Given that the Sarcos robot is a



a Learning curve locally weighted projection regression (LWPR) online learning

b Seven Degree-of-Freedom Sarcos Robot Arm

Locally Weighted Regression for Control. Figure 3. Learning an inverse dynamics model in real-time with a high-performance anthropomprohic robot arm

hydraulic robot, the RBD assumption is not very suitable, and, as Fig. 3a shows, LWPR (in thick red line) outperforms the analytical model (in dotted blue line) after a rather short amount of training. After about 5 min of training (about 125,000 data points), very good performance is achieved, using about 350 local models. This example demonstrates (i) the quality of function approximation that can be achieved with LWPR and (ii) the online allocation of more local models as needed.

Learning Paired Inverse-Forward Models

Learning inverse models (such as inverse kinematics and inverse dynamics models) can be challenging since the inverse model problem is often a relation, not a function, with a one-to-many mapping. Applying any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance, as the training data can form non-convex solution spaces, in which averaging is inappropriate. Architectures such as ►mixture models (in particular, mixture density networks) have been proposed to address problems with non-convex solution spaces. A particularly interesting approach in control involves learning linearizations of a forward model (which is proper function) and learning an inverse mapping within the local region of the forward model.

Ting et al. (2008) demonstrated such a forward-inverse model learning approach with Bayesian LWR to learn an inverse kinematics model for a haptic robot arm (shown in Fig. 4) in order to control the end-effector along a desired trajectory in task space. Training data was collected while the arm performed random sinusoidal movements within a constrained box volume of Cartesian space. Each sample consists of the arm's joint angles $\mathbf{q}$, joint velocities $\dot{\mathbf{q}}$, end-effector position in Cartesian space $\mathbf{x}$, and end-effector velocities $\dot{\mathbf{x}}$. From this data, a forward kinematics model is learned:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{10}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix. The transformation from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ can be assumed to be locally linear at a particular configuration $\mathbf{q}$ of the robot arm. Bayesian LWR is used to learn the forward model, and, as in LWPR, local models are only added if a training point is not already sufficiently covered by an existing local model. Importantly, the kernel functions in LWR are localized only with respect to $\mathbf{q}$, while the regression of each model is trained only on a mapping from $\dot{\mathbf{q}}$ to $\dot{\mathbf{x}}$ – these geometric insights are easily incorporated as priors in Bayesian LWR, as they are natural to locally linear models. Incorporating these priors in other function approximators, e.g., ►Gaussian Process regression, is not straightforward.

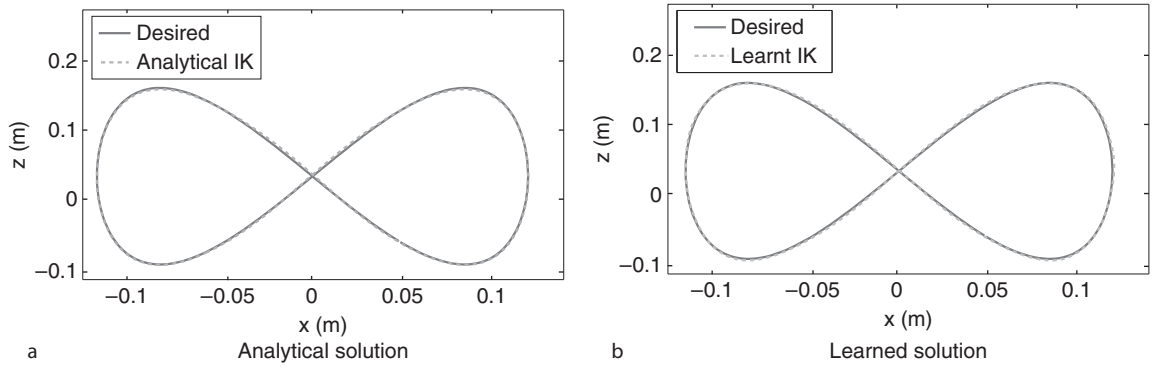
The goal of the robot task is to track a desired trajectory $(\mathbf{x}, \dot{\mathbf{x}})$ specified only in terms of x, z positions and velocities, i.e., the movement is supposed to be in a vertical plane in front of the robot, but the exact position of the vertical plane is not given. Thus, the task has one degree of redundancy, and the learning system needs to generate a mapping from $\{\mathbf{x}, \dot{\mathbf{x}}\}$ to $\dot{\mathbf{q}}$. Analytically, the inverse kinematics equation is

$$\dot{\mathbf{q}} = \mathbf{J}^{\#}(\mathbf{q})\dot{\mathbf{x}} - \alpha(\mathbf{I} - \mathbf{J}^{\#}\mathbf{J})\frac{\partial g}{\partial \mathbf{q}} \tag{11}$$

where $J^{\#}(\mathbf{q})$ is the pseudo-inverse of the Jacobian. The second term is an gradient descent optimization term for redundancy resolution, specified here by a cost function g in terms of joint angles $\mathbf{q}$.

To learn an inverse kinematics model, the local regions of $\mathbf{q}$ from the forward model can be re-used since any inverse of $\mathbf{J}$ is locally linear within these regions. Moreover, for locally linear models, all solution spaces for the inverse model are locally convex, such that an inverse can be learned without problems. The redundancy issue can be solved by applying an additional weight to each data point according to a reward function. Since the experimental task is specified in

Locally Weighted Regression for Control. Figure 4. SensAble Phantom haptic robotic arm



Locally Weighted Regression for Control. Figure 5. Desired versus actual trajectories for SensAble Phantom robot arm

terms of $\{\dot{x}, \dot{z}\}$, a reward is defined, based on a desired y coordinate, y_{des}, and enforced as a soft constraint. The resulting reward function, is $g = e^{-\frac{1}{2}h(k(y_{des}-y)-\dot{y})^2}$, where k is a gain and h specifies the steepness of the reward. This ensures that the learned inverse model chooses a solution that pushes $\dot{y}$ toward y_{des}. Each forward local model is inverted using a weighted ▶linear regression, where each data point is weighted by the kernel weight from the forward model and additionally weighted by the reward. Thus, a piecewise locally linear solution to the inverse problem can be learned efficiently.

Figure 5 shows the performance of the learned inverse model (Learnt IK) in a figure-eight tracking task. The learned model performs as well as the analytical inverse kinematics solution (Analytical IK), with root mean squared tracking errors in positions and velocities very close to that of the analytical solution.

Learning Trajectory Optimizations

Mitrovic, Klanke, and Vijayakumar (2008) have explored a theory for sensorimotor adaptation in humans, i.e., how humans replan their movement trajectories in the presence of perturbations. They rely on the iterative Linear Quadratic Gaussian (iLQG) algorithm (Todorov & Li, 2004) to deal with the nonlinear and changing plant dynamics that may result from altered morphology, wear and tear, or external perturbations. They take advantage of the "on-the-fly" adaptation of locally weighted regression methods like LWPR to learn the forward dynamics of a simulated arm for the purpose of optimizing a movement trajectory between a start point and an end point.

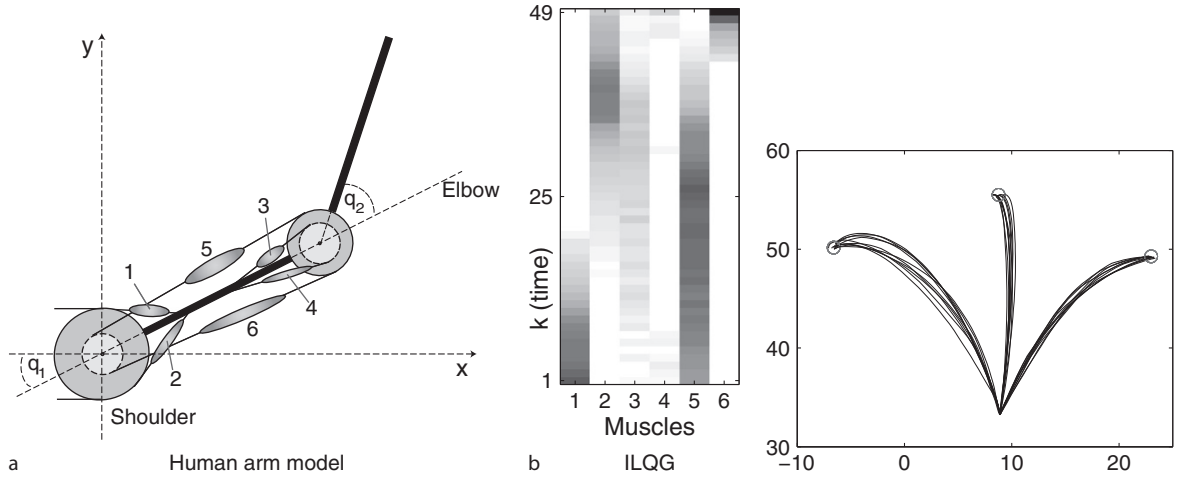
Figure 6a shows the diagram of a two degrees-of-freedom planar human arm model, which is actuated by four single-joint and two double-joint antagonistic muscles. Although kinematically simple, the system is over-actuated and, therefore, it is an interesting testbed because large redundancies in the dynamics have to be resolved. The dimensionality of the control signals makes adaptation processes (e.g., to external force fields) quite demanding.

The dynamics of the arm is, in part, based on standard RBD equations of motion:

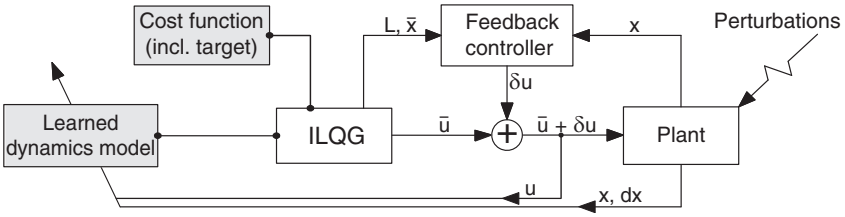
$$\tau = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} \tag{12}$$

where τ are the joint torques; $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint angles and velocities, respectively; $\mathbf{M}(\mathbf{q})$ is the two-dimensional symmetric joint space inertia matrix; and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ accounts for Coriolis and centripetal forces. Given the antagonistic muscle-based actuation, it is not possible to command joint torques directly. Instead, the effective torques from the muscle activations $\mathbf{u}$ – which happens to be quadratic in $\mathbf{u}$ – should be used. As a result, in contrast to standard torque-controlled robots, the dynamics equation in Eq. (12) is *nonlinear in the control signals* $\mathbf{u}$.

The iLQG algorithm (Todorov & Li, 2004) is used to calculate solutions to "localized" linear and quadratic approximations, which are iterated to improve the global control solution. However, it relies on an analytical forward dynamics model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ and finite difference methods to compute gradients. To alleviate this requirement and to make iLQG adaptive, LWPR can be used to learn an approximation of the plant's forward dynamics model. Figure 7 shows the control



Locally Weighted Regression for Control. Figure 6. (a) Human arm model with 6 muscles; (b) Optimized control sequence (*left*) and resulting trajectories (*right*) using the known analytic dynamics model. The control sequences (left target only) for each muscle (1–6) are drawn from bottom to top, with darker grey levels indicating stronger muscle activation

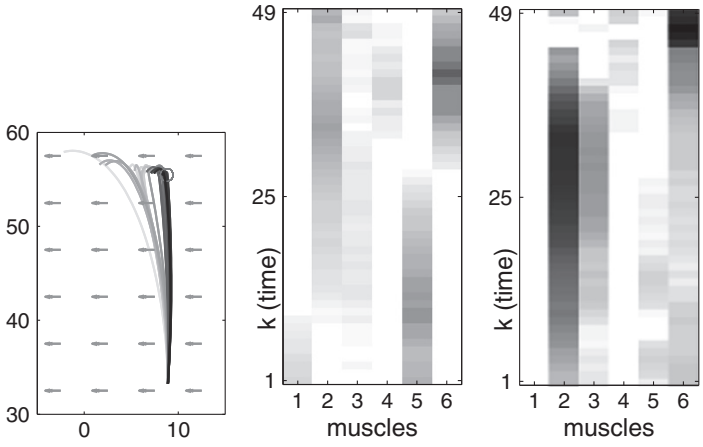


Locally Weighted Regression for Control. Figure 7. Illustration of learning and control scheme of the iterative Linear Quadratic Gaussian (iLQG) algorithm with learned dynamics

diagram, where the "learned dynamics model" (the forward model learned by LWPR) is then updated in an online fashion with every iteration to cope with changes in dynamics. The resulting framework is called iLQG-LD (iLQG with learned dynamics).

Movements of the arm model in Fig. 6a are studied for fixed time horizon reaching movement. The manipulator starts at an initial position $\mathbf{q}_0$ and reaches towards a target $\mathbf{q}_{tar}$. The cost function to be optimized during the movement is a combination of target accuracy and amount of muscle activation (i.e., energy consumption). Figure 6b shows trajectories of generated movements for three reference targets (shown in red circles) using the feedback controller from iLQG with the analytical plant dynamics. The trajectories generated with iLQG-LD (where the forward plant dynamics are learned with LWPR) are omitted as they are hardly distinguishable from the analytical solution.

A major advantage of iLQG-LD is that it does not rely on an accurate analytic dynamics model; this enables the framework to predict adaptation behavior under an ideal observer planning model. Reaching movements were studied where a constant unidirectional force field acting perpendicular to the reaching movement was generated as a perturbation (see Fig. 8 (*left*)). Using the iLQG-LD model, the manipulator gets strongly deflected when reaching for the target because the learned dynamics model cannot yet account for the "spurious" forces. However, when the deflected trajectory is used as training data and the dynamics model is updated online, the tracking improves with each new successive trial (Fig. 8 (*left*)). Please refer to Mitrovic et al. (2008) for more details. Aftereffects upon removing the force field, very similar to those observed in human experiments, are also observed.



Locally Weighted Regression for Control. Figure 8. Adaptation to a unidirectional constant force field (indicated by the arrows). Darker lines indicate better trained models. In particular, the left-most trajectory corresponds to the "initial" control sequence, which was calculated using the LWPR model *before* the adaptation process. The fully "adapted" control sequence results in a nearly straight line reaching movement

Cross References

►Bias and Variance
►Dimensionality Reduction
►Incremental Learning
►Kernel Function
►Kernel Methods
►Lazy Learning
►Linear Regression
►Mixture Models
►Online Learning
►Overfitting
►Radial Basis Functions
►Regression
►Supervised Learning

Programs and Data

http://www-clmc.usc.edu/software
http://www.ipab.inf.ed.ac.uk/slmc/software/

Recommended Reading

Aström, K. J., & Wittenmark, B. (1989). *Adaptive control*. Reading, MA: Addison-Wesley.

Atkeson, C., Moore, A., & Schaal, S. (1997). Locally weighted learning. *AI Review, 11*, 11–73.

Atkeson, C. (1989). Using local models to control movement. In *Proceedings of the advances in neural information processing systems 1* (pp. 157–183). San Francisco, CA: Morgan Kaufmann.

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association, 74*, 829–836.

Hastie, T., & Loader, C. (1993). Local regression: Automatic kernel carpentry. *Statistical Science, 8*, 120–143.

Jordan, M. I., & Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation, 6*, 181–214.

Klanke, S., Vijayakumar, S., & Schaal, S. (2008). A library for locally weighted projection regression. *Journal of Machine Learning Research, 9*, 623–626.

Mitrovic, D., Klanke, S., & Vijayakumar, S. (2008). Adaptive optimal control for redundantly actuated arms. In *Proceedings of the 10th international conference on the simulation of adaptive behavior*, Osaka, Japan (pp. 93–102). Berlin: Springer-Verlag.

Schaal, S., & Atkeson, C. G. (1998). Constructive incremental learning from only local information. *Neural Computation, 10*(8), 2047–2084.

Schaal, S., Atkeson, C. G., & Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics. *Applied Intelligence, 17*, 49–60.

Ting, J., D'Souza, A., Yamamoto, K., Yoshioka, T., Hoffman, D., Kakei, S., et al. (2005). Predicting EMG data from M1 neurons with variational Bayesian least squares. In *Proceedings of advances in neural information processing systems 18*, Cambridge: MIT Press.

Ting, J., Kalakrishnan, M., Vijayakumar, S., & Schaal, S. (2008). Bayesian kernel shaping for learning control. In *Proceedings of advances in neural information processing systems 21* (pp. 1673–1680). Cambridge: MIT Press.

Ting, J. (2009). Bayesian methods for autonomous learning systems. Ph.D. Thesis, Department of Computer Science, University of Southern California, 2009.

Todorov, E., & Li, W. (2004). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of 1st international conference of informatics in control, automation and robotics*, Setúbal, Portugal.

Vijayakumar, S., D'Souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation, 17*, 2602–2634.

Logic of Generality

Luc De Raedt
Katholieke Universiteit Leuven
Heverlee, Belgium

Synonyms

Generality and logic; Induction as inverted deduction; Inductive inference rules; Is more general than; Is more specific than; Specialization

Definition

One hypothesis is *more general* than another one if it covers all instances that are also covered by the latter one. The former hypothesis is called a ▶*generalization* of the latter one, and the latter a ▶*specialization* of the former. When using logical formulae as hypotheses, the generality relation coincides with the notion of logical entailment, which implies that the generality relation can be analyzed from a logical perspective. The logical analysis of generality, which is pursued in this chapter, leads to the perspective of *induction as the inverse of deduction*. This forms the basis for an analysis of various logical frameworks for reasoning about generality and for traversing the space of possible hypotheses. Many of these frameworks (such as for instance, *θ-subsumption*) are employed in the field of ▶inductive logic programming and are introduced below.

Motivation and Background

Symbolic machine learning methods typically learn by searching a hypothesis space. The hypothesis space can be (partially) ordered by the ▶generality relation, which serves as the basis for defining operators to traverse the space as well as for pruning away unpromising parts of the search space. This is often realized through the use of ▶refinement operators, that is, generalization and specialization operators. Because many learning methods employ a ▶hypothesis language that is logical or that can be reformulated in logic, it is interesting to analyze the generality relation from a logical perspective. When using logical formulae as hypotheses, the generality relation closely corresponds to logical entailment. This allows us to directly transfer results from logic to a machine learning context. In particular, machine learning operators can be derived from logical inference rules. The logical theory of generality provides a framework for transferring these results. Within the standard setting of inductive logic programming, learning from entailment, specialization is realized through deduction, and generalization through induction, which is considered to be the inverse of deduction. Different deductive inference rules lead to different frameworks for generalization and specialization. The most popular one is that of θ-subsumption, which is employed by the vast majority of contemporary inductive logic programming systems.

Theory

A hypothesis g is *more general than* a hypothesis s if and only if g covers all instances that are also covered by s, more formally, if $\text{covers}(s) \subseteq \text{covers}(g)$, in which case, $\text{covers}(h)$ denotes the set of all instances covered by the hypothesis h.

There are several possible ways to represent hypotheses and instances in logic (De Raedt, 1997, 2008), each of which results in a different setting with a corresponding covers relation. Some of the best known settings are *learning from entailment*, learning from interpretations, and learning from proofs.

Learning from Entailment

In learning from entailment, both hypotheses and instances are logical formulae, typically *definite clauses*, which underlie the programming language Prolog (Flach, 1994). Furthermore, when learning from entailment, a hypothesis h covers an instance e if and only if $h \models e$, that is, when h logically entails e, or equivalently, when e is a logical consequence of h. For instance, consider the hypothesis h:

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

The first clause or rule can be read as `flies` *if* `normal` *and* `bird`, that is, normal birds fly. The second and third states that blackbirds are birds. Consider now the examples e_1:

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

and e_2:

```
flies :- ostrich, small.
```

Example e_1 is covered by h, because it is a logical consequence of h, that is, $h \models e_1$. On the other hand, example e_2 is not covered, which we denote as $h \not\models e_2$.

When learning from entailment, the following property holds:

Property 1 *A hypothesis g is more general than a hypothesis s if and only if g logically entails s, that is, $g \models s$.*

This is easy to see. Indeed, g is more general than s if and only if $\text{covers}(s) \subseteq \text{covers}(g)$ if and only if for all examples $e : (s \models e) \rightarrow (g \models e)$, if and only if $g \models s$. For instance, consider the hypothesis h_1:

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

Because $h \models h_1$, it follows that h covers all examples covered by h_1, and hence, h generalizes h_1.

Property 1 states that the generality relation coincides with logical entailment when learning from entailment. In other learning settings, such as when *learning from interpretations*, this relationship also holds though the direction of the relationship might change.

Learning from Interpretations

In learning from interpretations, hypotheses are logical formulae, typically sets of definite clauses, and instances are interpretations. For propositional theories, interpretations are assignments of truth-values to propositional variables. For instance, continuing the `flies` illustration, two interpretations could be

```
{blackbird, bird, normal, flies} and
{ostrich, small}
```

where we specify interpretations through the set of propositional variables that are true. An interpretation specifies a kind of possible world. A hypothesis h then covers an interpretation if and only if the interpretation is a model for the hypothesis. An interpretation is a model for a hypothesis if it satisfies all clauses in the hypothesis. In our illustration, the first interpretation is a model for the theory h, but the second is not. Because the condition part of the rule `bird :- ostrich.` is satisfied in the second interpretation (as it contains `ostrich`), the conclusion part, that is, `bird`, should also belong to the interpretation in order to have a model. Thus, the first example is covered by the theory h, but the second is not.

When learning from interpretations, a hypothesis g is more general than a hypothesis s if and only if for all examples e: (e is a model of s) $\rightarrow$ (e is a model of g), if and only if $s \models g$.

Because the learning from entailment setting is more popular than the learning from interpretations setting, we shall employ in this section the usual convention that states that one hypothesis g is more general than a hypothesis s if and only if $g \models s$.

An Operational Perspective

Property 1 lies at the heart of the theory of *inductive logic programming* and generalization because it directly relates the central notions of logic with those of machine learning (Muggleton & De Raedt, 1994). It is also extremely useful because it allows us to directly transfer results from logic to machine learning.

This can be illustrated using traditional deductive inference rules, which start from a set of formulae and derive a formula that is entailed by the original set. For instance, consider the resolution inference rule for propositional definite clauses:

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n}. \quad (1)$$

This inference rule starts from the two rules above the line and derives the so-called *resolvent* below the line. This rule can be used to infer h_1 from h. An alternative deductive inference rule adds a condition to a rule:

$$\frac{h \leftarrow a_1, \ldots, a_n}{h \leftarrow a, a_1, \ldots, a_n}. \quad (2)$$

This rule can be used to infer that h_1 is more general than the clause used in example e_1. In general, a deductive inference rule can be written as

$$\frac{g}{s}. \tag{3}$$

If s can be inferred from g and the operator is *sound*, then $g \models s$. Thus, applying a deductive inference rule realizes specialization, and hence, deductive inference rules can be used as specialization operators. A *specialization operator* maps a hypothesis onto a set of its specializations. Because specialization is the inverse of generalization, *generalization operators* – which map a hypothesis onto a set of its generalizations – can be obtained by inverting deductive inference rules. The inverse of a deductive inference rule written in format (3) works from bottom to top, that is, from s to g. Such an inverted deductive inference rule is called an *inductive* inference rule. This leads to the view of induction as the inverse of deduction. This view is operational as it implies that each deductive inference rule can be inverted into an inductive one, and, also, that each inference rule provides an alternative framework for generalization.

An example of a generalization operator is obtained by inverting the adding condition rule (2). It corresponds to the well-known "dropping condition" rule (Michalski, 1983). As will be seen soon, it is also possible to invert the resolution principle (1).

Before deploying inference rules, it is necessary to determine their properties. Two desirable properties are *soundness* and *completeness*. These properties are based on the repeated application of inference rules. Therefore, we write $g \vdash_r s$ when there exists a sequence of hypotheses $h_1, \ldots, h_n$ such that

$$\frac{g}{h_1}, \frac{h_1}{h_2}, \ldots, \frac{h_n}{s} \text{ using } r. \tag{4}$$

A set of inference rules r is *sound* whenever $g \vdash_r s$ implies $g \models s$; and *complete* whenever $g \models s$ implies $g \vdash_r s$. In practice, soundness is always enforced though completeness is not always required in a machine learning setting. When working with incomplete rules, one should realize that the generality relation "$\vdash_r$" is weaker than the logical one "$\models$."

The most important logical frameworks for reasoning about generality, such as θ-subsumption and resolution, are introduced below using the above introduced logical theory of generality.

Frameworks for Generality

Propositional Subsumption

Many propositional learning systems employ hypotheses that consist of rules, often definite clauses as in the `flies` illustration above. The propositional subsumption relation defines a generality relation among clauses and is defined through the adding condition rule (2). The properties follow from this inference rule by applying the logical theory of generalization presented above. More specifically, the generality relation $\vdash_a$ induced by the adding condition rule states that a clause g is more general than a clause s, if s can be derived from g by adding a sequence of conditions to g. Observing that a clause $h \leftarrow a_1, \ldots, a_n$ is a disjunction of literals $h \vee \neg a_1 \vee \cdots \vee \neg a_n$ allows us to write it in set notation as $\{h, \neg a_1, \ldots, \neg a_n\}$. The soundness and completeness of propositional subsumption then follow from

$$g \vdash_a s \text{ if and only if } g \subseteq s \text{ if and only if } g \models s, \tag{5}$$

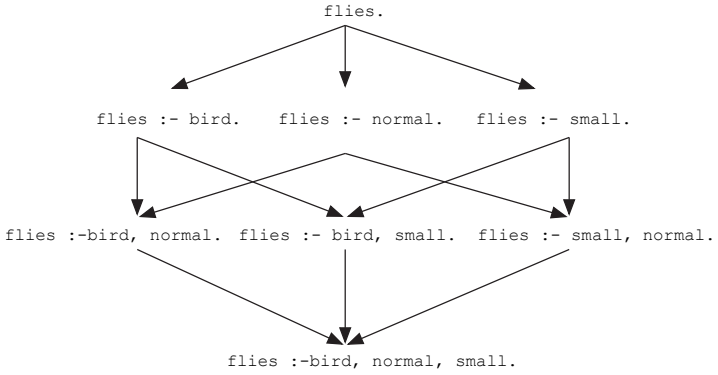
which also states that g subsumes s if and only if $g \subseteq s$.

The propositional subsumption relation induces a complete lattice on the space of possible clauses. A *complete lattice* is a partial order – a reflexive, antisymmetric, and transitive relation – where every two elements posses a unique least upper and greatest lower bound. An example lattice for rules defining the predicate `flies` in terms of `bird`, `normal`, and `small` is illustrated in the Hasse diagram depicted in Fig. 1.

The Hasse diagram also visualizes the different operators that can be used. The *generalization* operator ρ_g maps a clause to the set of its parents in the diagram, whereas the *specialization* operator ρ_s maps a clause to the set of its children. So far, we have defined such operators *implicitly* through their corresponding inference rules. In the literature, they are often defined *explicitly*:

$$\rho_g(h \leftarrow a_1, \ldots, a_n) = \{h \leftarrow a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n | i = 1, \ldots, n\}. \tag{6}$$

In addition to using the inference rules directly, some systems such as Golem (Muggleton & Feng, 1990)



Logic of Generality. Figure 1. The Hasse diagram for the predicate `flies`

also exploit the properties of the underlying lattice by computing the least upper bound of two formulae. The least upper bound operator is known under the name of least general generalization (lgg) in the machine learning literature. It returns the least common ancestor in the Hasse diagram. Using a set notation for clauses, the definition of the lgg is:

$$\text{lgg}(c_1, c_2) = c_1 \cap c_2. \tag{7}$$

The least general generalization operator is used by machine learning systems that follow a cautious generalization strategy. They take two clauses corresponding to positive examples and minimally generalize them.

θ-Subsumption

The most popular framework for generality within inductive logic programming is θ-subsumption (Plotkin, 1970). It provides a generalization relation for clausal logic and it extends propositional subsumption to first order logic.

A *definite clause* is an expression of the form $h \leftarrow a_1, \ldots, a_n$ where h and the a_i are logical atoms. An *atom* is an expression of the form $p(t_1, \ldots, t_m)$ where p is a *predicate name* (or, the name of a relation) and the t_i are terms. A *term* is either a constant (denoting an object in the domain of discourse), a variable, or a structured term of the form $f(u_1, \ldots, u_k)$ where f is a functor symbol (denoting a function in the domain of discourse) and the u_i are terms, see Flach (1994) for more details. Consider for instance the clauses

```
likes(X,Y) :- neighbours(X,Y).
likes(X,husbandof(Y)) :- likes(X,Y).
likes(X,tom) :- neighbours(X,tom),
                male(X).
```

The first clause states that `X likes Y` if `X` is a `neighbour` of `Y`. The second one that `X likes` the `husband` of `Y` if `X likes Y`. The third one that all `male neighbours` of `tom` like `tom`.

θ-Subsumption is based not only on the adding condition rule (2) but also on the *substitution rule*:

$$\frac{g}{g\theta}. \tag{8}$$

The substitution rule applies a substitution θ to the definite clause g. A substitution $\{V_1/t_1, \ldots, V_n/t_n\}$ is an assignment of terms to variables. Applying a substitution to a clause c yields the instantiated clause, where all variables are simultaneously replaced by their corresponding terms.

θ-subsumption is then the generality relation induced by the substitution and the adding condition rules. Denoting this set of inference rules by t, we obtain our definition of θ-subsumption:

g θ-subsumes s if and only if $g \vdash_t s$ if and only if
$$\exists\theta : g\theta \subseteq s. \tag{9}$$

For instance, the first clause for `likes` subsumes the third one with the substitution `{Y/tom}`.

θ-subsumption has some interesting properties:

- θ-subsumption is sound.
- θ-subsumption is complete for clauses that are not self-recursive. It is incomplete for self-recursive clauses such as

```
nat(s(X)) :- nat(X)
nat(s(s(Y))) :- nat(Y)
```

for which one can use resolution to prove that the first clause logically entails the second one, even though it does not θ-subsume it.

- Deciding θ-subsumption is an NP-complete problem.

Because θ-subsumption is relatively simple and decidable whereas logical entailment between single clauses is undecidable, it is used as the generality relation by the majority of inductive logic programming systems. These systems typically employ a specialization or refinement operator to traverse the search space. To guarantee systematic enumeration of the search space, the specialization operator ρ_s can be employed. $\rho_s(c)$ is obtained by applying the adding condition or substitution rule with the following restrictions.

- The adding condition rule only adds atoms of the form $p(V_1, \ldots, V_n)$, where the V_i are variables not yet occurring in the clause c.
- The substitution rule only employs *elementary substitutions*, which are of the form
  - $\{X/Y\}$, where X and Y are two variables appearing in c
  - $\{V/ct\}$, where V is a variable in c and ct a constant
  - $\{V/f(V_1, \ldots, V_n)\}$, where V is a variable in c, f a functor of arity n and the V_i are variables not yet occurring in c.

A generalization operator can be obtained by inverting ρ_s, which requires one to invert substitutions. Inverting substitutions is not easy. While applying a substitution θ = `{V/a}` to a clause c replaces all occurrences of `V` by `a` and yields a unique clause $c\theta$, applying the substitution rule in the inverse direction does not necessarily yield a unique clause. If we assume the elementary substitution applied to c with

$$\frac{c}{\texttt{q(a,a)}}. \tag{10}$$

was `{V/a}`, then there are at least three possibilities for c: `q(a,V)`, `q(V,a)`, and `q(V,V)`.

θ-subsumption is reflexive, transitive but unfortunately not anti-symmetric, which can be seen by considering the clauses

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- father(X,Y),
               father(U,V).
```

The first clause clearly subsumes the second one because it is a subset. The second one subsumes the first with the substitution {`X/U, V/Y`}. The two clauses are therefore equivalent under θ-subsumption, and hence also logically equivalent. The loss of the anti-symmetry complicates the search process. The naive application of the specialization operator ρ_s may yield syntactic specializations that are logically equivalent. This is illustrated above where the second clause for `parent` is a refinement of the first one using the adding condition rule. In this way, useless clauses are generated, and if the resulting clauses are further refined, there is a danger that the search will end up in an infinite loop.

Plotkin (1970) has studied the quotient set induced by θ-subsumption and proven various interesting properties. The quotient set consists of classes of clauses that are equivalent under θ-subsumption. The class of clauses equivalent to a given clause c is denoted by

$$[c] = \{c' | c' \text{ is equivalent with } c \text{ under } \theta\text{-subsumption}\}. \tag{11}$$

Plotkin proved that

- The quotient set is well-defined w.r.t. θ-subsumption.
- There is a representative, a canonical form, of each equivalence class, the so-called *reduced clause*. The reduced clause of an equivalence class is the shortest clause belonging to class. It is unique up to variable renaming. For instance, in the `parent` example above, the first clause is in reduced form.
- The quotient set forms a complete lattice, which implies that there is a least general generalization of two equivalence classes. In the inductive logic programming literature, one often talks about the least general generalization of two clauses.

Several variants of θ-subsumption have been developed. One of the most important ones is that of *OI*-subsumption (Esposito, Laterza, Malerba, & Semeraro, 1996). For functor-free clauses, it modifies the substitution rule by disallowing substitutions that unify two variables or that substitute a variable by a constant already appearing in the clause. The advantage is that the resulting relation is anti-symmetric, which avoids some of the above mentioned problems with refinement operators. On the other hand, the minimally general generalization of two clauses is not necessary unique, and hence, there exists no least general generalization operator.

Inverse Resolution

Applying resolution is a sound deductive inference rule and therefore realizes specialization. Reversing it yields inductive inference rules or generalization operators (Muggleton, 1987; Muggleton & Buntine, 1988). This is typically realized by combining the resolution principle with a copy operator. The resulting rules are called *absorption* (12) and *identification* (13). They start from the clauses below and induce the clause above the line. They are shown here only for the propositional case, as the first order case requires one to deal with substitutions as well as inverse substitutions.

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}, \quad (12)$$

$$\frac{h \leftarrow g, a_1, \ldots, a_n \text{ and } g \leftarrow b_1, \ldots, b_m}{h \leftarrow b_1, \ldots, b_m, a_1, \ldots, a_n \text{ and } h \leftarrow g, a_1, \ldots, a_n}. \quad (13)$$

Other interesting inverse resolution operators perform predicate invention, that is, they introduce new predicates that were not yet present in the original data. These operators invert two resolution steps. One such operator is the *intra-construction* operator (14). Applying this operator from bottom to top introduces the new predicate q that was not present before.

$$\frac{q \leftarrow l_1, \ldots, l_k \text{ and } p \leftarrow k_1, \ldots, k_n, q \text{ and } q \leftarrow l_1', \ldots, l_m'}{p \leftarrow k_1, \ldots, k_n, l_1, \ldots, l_k \text{ and } p \leftarrow k_1, \ldots, k_n, l_1', \ldots, l_m'}. \quad (14)$$

The idea of inverting the resolution operator is very appealing because it aims at inverting the most popular deductive inference operator, but is also rather complicated due to the non-determinism and the need to invert substitutions. Due to these complications, there are only few systems that employ inverse resolution operators.

Background Knowledge

Inductive logic programming systems employ background knowledge during the learning process. Background knowledge typically takes the form of a set of clauses B, which is then used by the covers relation. When learning from entailment in the presence of background knowledge B an example e is covered by a hypothesis h if and only if $B \cup h \models e$. This notion of coverage is employed in most of the work on inductive logic programming. In the intial `flies` example, the two clauses defining `bird` would typically be considered background knowledge.

The incorporation of background knowledge in the induction process has resulted in the frameworks for generality *relative* to a background theory. More formally, a hypothesis g is more general than a hypothesis s relative to the background theory B if and only if $B \cup g \models s$. The only inference rules that deal with multiple clauses are those based on (inverse) resolution. The other frameworks can be extended to cope with this generality relation following the logical theory of generalization. Various frameworks have been developed along these lines. Some of the most important ones are relative subsumption (Plotkin, 1971) and generalized subsumption (Buntine, 1998), which extend θ-subsumption and the notion of least general generalization toward the use of background knowledge. Computing the least general generalization of two clauses relative to the background theory is realized by first computing the most specific clauses covering the examples with regard to the background theory and then generalizing them using the least general generalization operator of θ-subsumption.

The first step is the most interesting one, and has been tackled under the name of *saturation* (Rouveirol, 1994) and *bottom-clauses* (Muggleton, 1995). We illustrate it within the framework of inverse entailment due to Muggleton (1995). The bottom clause $\bot(c)$ of a clause c with regard to a background theory B is the most specific clause $\bot(c)$ such that

$$B \cup \bot(c) \models c. \quad (15)$$

If B consist of

```
polygon :-  rectangle.
rectangle :- square.
oval :- circle.
```

and the example c is

```
positive :- red, square.
```

Then the bottom-clause $\bot(c)$ is

```
positive :- red, rectangle, square,
            polygon.
```

The bottom-clause is useful because it only lists those atoms that are relevant to the example, and only generalizations (under θ-subsumption) of $\bot(c)$ will cover the example. For instance, in the illustration, the bottom-clause mentions neither `oval` nor `circle` as clauses for `pos` containing these atoms will never cover the example clause c. Once the bottom-clause covering an example has been found the search process continues as if no background knowledge were present. Either specialization operators (typically under θ-subsumption) would search the space of clauses more general than $\bot(c)$, or the least general generalization of multiple bottom-clauses would be computed.

Equation (15) is equivalent to

$$B \cup \neg c \models \neg \bot(c), \tag{16}$$

which explains why the bottom-clause is computed by finding all factual consequences of $B \cup \neg c$ and then inverting the resulting clause again. On the example:

$\neg c$ = {¬ positive, red, square}

and the set of all consequences is

$\neg \bot(c)$ = $\neg c \cup$ {rectangle, polygon}

which then yields $\bot(c)$ mentioned above. When dealing with first order logic, bottom-clauses can become infinite, and therefore, one typically imposes further restrictions on the atoms that appear in bottom-clauses. These restrictions are part of the *language bias*.

The textbook by Nienhuys-Cheng and De Wolf (1997) is the best reference for an in-depth formal description of various frameworks for generality in logic, in particular, for θ-subsumption and some of its variants. The book by De Raedt (2008) contains a more complete introduction to inductive logic programming and relational learning, and also introduces the key frameworks for generality in logic. An early survey of inductive logic programming and the logical theory of generality is contained in Muggleton and De Raedt (1994). Plotkin (1970, 1971) pioneered the use θ-subsumption and relative subsumption (under a background theory) for machine learning. Buntine (1998) extended these frameworks toward generalized subsumption, and Esposito et al. (1996) introduced *OI*-subsumption. Inverse resolution was first used in the system Marvin (Sammut & Banerji, 1986), and then elaborated by Muggleton (1987) for propositional logic and by Muggleton and Buntine (1988) for definite clause logic. Various learning settings are studied by De Raedt (1997) and discussed extensively by De Raedt (2008). They are also relevant to ▶probabilistic logic learning and ▶statistical relational learning.

Recommended Reading

Buntine, W. (1998). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence, 36*, 375–399.

De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence, 95*, 187–201.

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

Semeraro, G., Esposito, F., & Malerba, D. (2006). Ideal Refinement of Datalog Programs. In *Proceedings of the 5th International Workshop on Logic Program Synthesis and Transformation, Lecture notes in computer science* (Vol. 1048, pp. 120–136). Springer.

Flach, P. A. (1994). *Simply logical: Intelligent reasoning by example*. New York: Wiley.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence, 20*(2), 111–161.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *Proceedings of the 10th International Joint conference on Artificial Intelligence* (pp. 287–292). San Francisco: Morgan Kaufmann.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13*(3–4), 245–286.

Muggleton, S., & Buntine, W. (1988). Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning* (pp. 339–351). San Francisco: Morgan Kaufmann.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19/20*, 629–679.

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st conference on Algorithmic Learning Theory* (pp. 368–381). Ohmsma, Tokyo, Japan.

Nienhuys-Cheng, S.-H., & de Wolf, R. (1997). *Foundations of inductive logic programming*. Berlin: Springer.
Plotkin, G. D. (1970). A note on inductive generalization. In *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh: Edinburgh University Press.
Plotkin, G. D. (1971). A further note on inductive generalization. In *Machine Intelligence* (Vol. 6, pp. 101–124). Edinburgh: Edinburgh University Press.
Rouveirol, C. (1994). Flattening and saturation: Two representation changes for generalization. *Machine Learning, 14*(2), 219–232.
Sammut, C., & Banerji, R. B. (1986). Learning concepts by asking questions. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 167–192). San Francisco: Morgan Kaufmann.

Logic Program

A logic program is a set of logical rules or ►clauses. Logic programs are employed to answer queries using the ►resolution inference rule. For example, consider the following logic program:

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

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

father(charles, william).
   mother(diana, william).
father(philip, charles).
   mother(elizabeth, charles).

father(john, diana).
   mother(frances, diana).
```

Using resolution we obtain the following answers to the query `:-grandparent(X,Y)`:

```
X = philip,     Y = william ;
X = john,       Y = william ;
X = elizabeth,  Y = william ;
X = frances,    Y = william.
```

Cross References

►Clause
►First-Order Logic
►Prolog

Logical Consequence

►Entailment

Logical Regression Tree

►First-Order Regression Tree

Logistic Regression

Synonyms

Logit model

Definition

Logistic regression provides a mechanism for applying the techniques of ►linear regression to ►classification problems. It utilizes a linear regression model of the form

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where x_1 to x_n represent the values of the n attributes and β_0 to β_n represent weights. This model is mapped onto the interval [0,1] using

$$P(c_0 \mid x_1 \ldots x_n) = \frac{1}{1 + e^{-z}}$$

where c_0 represents class 0.

Recommended Reading

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). New York: Springer.

Logit Model

►Logistics Regression

Log-Linear Models

►Maximum Entropy Models for Natural Language Processing

Long-Term Potentiation of Synapses

By a suitable induction protocol, the connection between two neurons can be strengthened. If this change persists for hours, the effect is called a long-term potentation.

LOO Error

►Leave-One-Out Error

Loopy Belief Propagation

Loopy belief propagation is a heuristic inference algorithm for ►Bayesian networks. See ►Graphical Models for details.

Loss

Synonyms
Cost

Definition
The *cost* or *loss* of a prediction y', when the correct value is y, is a measure of the relative utility of that prediction given that correct value. A common loss function used with ►classification learning is ►*zero-one loss*. Zero-one loss assigns 0 to loss for a correct classification and 1 for an incorrect classification. ►Cost sensitive classification assigns different costs to different forms of misclassification. For example, misdiagnosing a patient as having appendicitis when he or she does not might be of lower cost than misdiagnosing the patient as not having it when he or she does. A common loss function used with ►regression is ►error squared. This is the square of the difference between the predicted and true values.

Loss Function

Synonyms
Cost function

Definition
A loss function is a function used to determine ►loss.

LWPR

►Locally Weighted Regression for Control

LWR

►Locally Weighted Regression for Control