R

Rademacher Average

►Rademacher Complexity

Rademacher Complexity

Synonyms
Rademacher average

Definition
Rademacher complexity is a measure used in ►generalization bounds to quantify the "richness" of a class of functions. Letting $\rho_1, \ldots, \rho_n$ denote *Rademacher variables* – independent random variables that take the values ± 1 with equal probability – the *empirical* or *conditional Rademacher complexity* of a class of real-valued functions $\mathcal{F}$ on the points $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ is the conditional expectation

$$\hat{R}_{\mathbf{x}}(\mathcal{F}) = \mathbb{E}_{\mathbf{x},\rho}\left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \rho_i f(x_i) \middle| \mathbf{x}\right].$$

Intuitively, the empirical Rademacher average $R_n(\mathcal{F})$ measures how well functions $f \in \mathcal{F}$ evaluated on $\mathbf{x} \in \mathcal{X}$ can align with randomly chosen labels. The (full) *Rademacher complexity* $R_n(\mathcal{F})$ with respect to a distribution P over $\mathcal{X}$ is the average empirical complexity when the arguments $x_1, \ldots, x_n$ are independent random variables drawn from P. That is,

$$R_n(\mathcal{F}) = \mathbb{E}_{\mathbf{x}}\left[\hat{R}_{\mathbf{x}}(\mathcal{F})\right].$$

There are several properties of the Rademacher average that make it a useful quantity in analysis: for any two classes $\mathcal{F} \subseteq \mathcal{G}$ we have $R_n(\mathcal{F}) \le R_n(\mathcal{G})$; when $c \cdot \mathcal{F} := \{cf : f \in \mathcal{F}\}$ for $c \in \mathbb{R}$ we have $R_n(c \cdot \mathcal{F}) = |c| R_n(\mathcal{F})$; when $\mathcal{F} + g := \{f + g : f \in \mathcal{G}\}$ for some fixed function g we have $R_n(\mathcal{F} + g) = R_n(\mathcal{F})$; and if $\text{conv}(\mathcal{F})$ is the convex hull of $\mathcal{F}$ then $R_n(\text{conv}(\mathcal{F})) = R_n(\mathcal{F})$.

Radial Basis Function Approximation

►Radial Basis Function Networks

Radial Basis Function Networks

M.D. Buhmann
Justus-Liebig University,
Giessen, Germany

Synonyms
Networks with kernel functions; Radial basis function approximation; Radial basis function neural networks; Regularization networks

Definition
Radial basis function networks are a means of approximation by algorithms using linear combinations of translates of a rotationally invariant function, called the radial basis function. The coefficients of these approximations usually solve a minimization problem and can also be computed by interpolation processes. The radial basis functions constitute the so-called reproducing kernels on certain Hilbert-spaces or – in a slightly more general setting – semi-Hilbert spaces. In the latter case, the aforementioned approximation also contains an element from the nullspace of the semi-norm of the semi-Hilbert space. That is usually a polynomial space.

Motivation and Background
Radial basis function networks are a method to approximate functions and data by applying ►kernel methods to ►neural networks. More specifically, approximations

of functions or data via algorithms that make use of networks (or neural networks) can be interpreted as either interpolation or minimization problems using kernels of certain shapes, called radial basis functions in the form in which we wish to consider them in this chapter. In all cases, they are usually *high-dimensional approximations*, that is the number of unknowns n in the argument of the kernel may be very large. On the other hand, the number of learning examples ("data") may be quite small. The name neural networks comes from the idea that this learning process simulates the natural functioning of neurons.

At any rate, the purpose of this approach will be the modelization of the learning process by mathematical methods. In most practical cases of networks, the data from which we will learn in the method are rare, i.e., we have few data "points." We will consider this learning approach as an approximation problem in this description, essentially it is a minimizing (regression) problem.

Structure of the Network/Learning System

To begin with, let $\varphi : \mathbb{R}_+ \to \mathbb{R}$ be a univariate continuous function and $\|\cdot\|$ be the Euclidean norm on $\mathbb{R}^n$ for some $n \in \mathbb{N}$, as used for approximation in the seminal paper by Schoenberg (1938). Here, $\mathbb{R}_+$ denotes the set of nonnegative reals. Therefore,

$$\varphi(\|\cdot\|) : \mathbb{R}^n \to \mathbb{R}, \quad (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \mapsto \varphi\left(\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\right)$$

is a multivariate function and here the number n of unknowns may be very large in practice. This function is rotationally invariant. Incidentally, much of what is going to be said here will work if we replace this function by a general, n-variate function which need no longer be rotationally invariant, but then, strictly speaking, we are no longer talking about radial basis functions. Then other conditions may replace the restriction to radiality. Nonetheless, we stick to the simple case (which is entirely sufficient for many practical applications) when the function really is radially symmetric.

We also require for the time being that this n-variate function be positive definite, that is for all finite sets Ξ of pairwise different the so-called *centers* or data sites $\xi \in \Xi \subset \mathbb{R}^n$, the symmetric matrix

$$A = \{\varphi(\|\xi - \zeta\|)\}_{\xi,\zeta\in\Xi}$$

is a positive definite matrix. The condition of pairwise different data in Ξ may, of course, in practice, not be necessarily met.

This property is usually obtained by requiring that $\varphi(\|\cdot\|)$ be absolutely integrable and its Fourier transform – which thereby exists and is continuous – is positive everywhere ("Bochner's theorem"). An example for such a useful function is the exponential (the "Gauß-kernel") $\varphi(r) = \exp(-c^2 r^2)$, $r \geqslant 0$, where c is a positive parameter. For this the above positive definiteness is guaranteed for all positive c and all n. Another example is the Poisson-kernel $\varphi(r) = \exp(-cr)$. However, we may also take the nonintegrable "inverse multiquadrics" $\varphi(r) = 1/\sqrt{r^2 + c^2}$, which has a Fourier transform in the generalized or distributional sense that is also positive everywhere except at zero. There it has a singularity. Nonetheless, the aforementioned matrices of the form A are still always positive definite for these exponentials and the inverse multiquadrics so long as $c > 0$ and $n = 1, 2, \ldots$

This requirement of positive definiteness guarantees that for all given finite sets Ξ and "data" $f_\xi \in \mathbb{R}$, $\xi \in \Xi$, there is a unique linear combination

$$s(x) = \sum_{\xi\in\Xi} \lambda_\xi \varphi(\|x - \xi\|), \quad x \in \mathbb{R}^n,$$

which satisfies the linear interpolation conditions

$$s(\xi) = f_\xi, \quad \forall\ \xi \in \Xi.$$

This is because the interpolation matrix which is used to compute the coefficients λ_ξ is just the matrix A above which is positive definite, thus regular. The expression in the pen-ultimate display is the network that approximates the data given by the user. Of course the interpolation conditions are just what is meant by learning from examples, the data being the $|\Xi|$ examples. Here as always, $|\Xi|$ denotes the cardinality of the set Ξ. In the learning theory the linear space spanned by the above translates of $\varphi(\|\cdot\|)$ by $\xi \in \Xi$ is called the feature space with φ as activation function.

Incidentally, it is straightforward to generalize the approximation method to an approximation to data in $\mathbb{R}^m$, $m \in \mathbb{N}$, by approximating the data $f_\xi \in \mathbb{R}^m$ componentwise by m such expressions as the above, call them $s_1, s_2, \ldots, s_m$.

Applications

Applications include classification of data, pattern recognition, ▶time series analysis, picture smoothing similar to diffusion methods, and optimization.

Theory/Solution

Returning to interpolation, the problem may also be reinterpreted as a minimization problem. If the weighted L^2-integral is defined as

$$\|g\|_\varphi := \frac{1}{(2\pi)^{n/2}} \sqrt{\int_{\mathbb{R}^n} \frac{1}{\hat{\varphi}(\|x\|)} |\hat{g}(x)|^2 \,\mathrm{d}x},$$

with $\hat{\varphi}$ still being the above positive Fourier transform, for all $g : \mathbb{R}^n \to \mathbb{R}$ for which the Fourier transform in the sense of $L^2(\mathbb{R}^n)$ is well-defined and for which the above integral is finite, we may ask for the approximant to the above data – which still must satisfy the aforementioned interpolation conditions – that minimizes $\|\cdot\|_\varphi$. As Duchon noted, for example, for the thin-plate spline case $\varphi(r) = r^2 \log r$ in this seminal papers this is just the above interpolant, i.e., that linear combination s of translates of radial basis functions, albeit in the thin-plate spline case with a linear polynomial added as we shall see below.

This works immediately both for the two examples of exponential functions and the inverse multiquadrics. Note the fact that the latter has a Fourier transform with a singularity at the origin, does not matter as its reciprocal appears as a weight function in the integral above. The important requirement is that the Fourier transform has no zero. It also works for the positive definite radial basis functions of compact support for instance in Buhmann (1998).

Regularization and Generalizations

Generally, since the interpolation problem to data may be ill-conditioned or unsuitable in the face of ▶noise, smoothing or ▶regularization are appropriate as an alternative. Indeed, the interpolation problem may be replaced by a smoothing problem which is of the form

$$\frac{1}{|\Xi|} \sum_{\xi\in\Xi} (s(\xi) - f_\xi)^2 + \mu \|s\|_\varphi^2 = \min_s !.$$

Here the L^2-integral is still the one used in the description above and μ is a positive smoothing parameter.

However, when there is only a trivial nullspace of the $\|\cdot\|_\varphi$, i.e., $g = 0$ is the only g with $\|g\|_\varphi = 0$, then it is a norm and the solution of this problem will have the form

$$s(x) = \sum_{\xi\in\Xi} \lambda_\xi \varphi(\|x - \xi\|), \quad x \in \mathbb{R}^n.$$

This is where the name regularization network comes from, regularization and smoothing being used synonymously. The form used here in the pen-ultimate display is a classical regularizing network problem or in the spline-terminology a smoothing spline problem. For ▶support vector machines, the square of the residual term $s(\xi) - f_\xi$ should be replaced by another expression, for example, the one by Vapnik (1996)

$$|s(\xi) - f_\xi|_\epsilon := \begin{cases} f_\xi - s(\xi) - \epsilon & \text{if } |f_\xi - s(\xi)| \geq \epsilon, \\ 0 & \text{otherwise,} \end{cases}$$

and for the support vector machines classification by the truncated power function $(\cdot)_+^\nu$ which is a positive power for positive argument and otherwise zero.

In the case of a classical regularizing network, the coefficients of the solution may be found by solving a similar linear system to the standard interpolation linear system mentioned above, namely

$$(A + \mu I)\lambda = f,$$

where f is the vector $(f_\xi)_{\xi\in\Xi}$ in $\mathbb{R}^\Xi$ of the data given, and $\lambda = (\lambda_\xi)_{\xi\in\Xi}$. The I denotes the $|\Xi| \times |\Xi|$ identity matrix and A is still the same matrix as above. Incidentally, scaling mechanisms may also be introduced into the radial basis function by replacing the simple translate $\varphi(\|x - \xi\|)$ by $\varphi(\|x - \xi\|/\delta)$ for a positive δ which may even depend on ξ.

The ideas of regularization and smoothing are of course not new; for instance, regularization goes back to Tichonov et al. (1977) ("Tichonov regularization") and spline smoothing to Wahba (1985), especially when the smoothing parameter is adjusted via cross-validation or generalized cross-validation (GCV).

Now to the case of semi-norms $\|\cdot\|_\varphi$ with nontrivial nullspaces: indeed, the same idea can be carried through for other radial basis functions as well. In particular we are thinking here of those that do not provide positive definite radial basis interpolation matrices but strictly conditionally positive definite ones. We have strictly positive definite radial basis functions of order $k+1$, $k \geq -1$, if the above interpolation matrices A are still positive definite but only on the subspace of those nonzero vectors $\lambda = (\lambda_\xi)$ in $\mathbb{R}^\Xi$ which satisfy

$$\sum_{\xi\in\Xi} \lambda_\xi p(\xi) = 0, \quad \forall\, p \in \mathbb{P}_n^k,$$

where $\mathbb{P}_n^k$ denotes the linear space of polynomials in n variables with total degree at most k. In other words, the quadratic form, $\lambda^{\mathrm{T}} A \lambda$, need only be positive for such $\lambda \neq 0$. For simplicity of the presentation, we shall let $\mathbb{P}_n^{-1}$ denote $\{0\}$. In particular, if the radial basis function is conditionally positive definite of order 0, its interpolation matrices A are always positive definite, i.e., without condition. Also, we have the minimal requirement that the sets of centers Ξ are unisolvent for this polynomial space, i.e., the only polynomial $p \in \mathbb{P}_n^k$ that vanishes identically on Ξ is the zero-polynomial.

The connection of this with a layered neural network is that the approximation above is a weighted sum (weighted by the coefficients λ_ξ) over usually nonlinear activation functions φ. The entries in the sum are the radial basis function neurons and there are usually many of them. The number of nodes in the model is n. The hidden layer of "radial basis function units" consists of $|\Xi|$ nodes, i.e., the number of centers in our radial basis function approximation. The output layer has m responses if the radial basis function approximation above is generalized to m-variate data, then we get $s_1, s_2, \ldots, s_m$ instead of just s, as already described. This network here is of the type of a nonlinear, layered, and feedforward network. More than one hidden layer is unusual. The choice of the radial basis functions (its smoothness for instance) and the flexibility in the positioning of the centers in clusters, grids (Buhmann, 1990, for example) or otherwise provide much of the required freedom for good approximations.

The properties of conditional positive definiteness are fulfilled now for a much larger realm of radial basis functions, which have still nowhere vanishing, generalized Fourier transforms but with higher order singularities at the origin. (Remember that this creates no problem for the well-defineness of $\|\cdot\|_\varphi$.) For instance, the above properties are true for the thin-plate spline function $\varphi(r) = r^2 \log r$, the shifted logarithm $\varphi(r) = (r^2 + c^2)\log(r^2 + c^2)$, and for the multiquadric $\varphi(r) = -\sqrt{r^2 + c^2}$. Here we still have a parameter c which may now be arbitrary real. The order of the above is one for the multiquadric and two for the thin-plate spline. Another commonly used radial basis function which gives rise to conditional positive definiteness is the $\varphi(r) = r^3$.

Hence the norm becomes a semi-norm with nullspace $\mathbb{P}_n^k$ but it still has the same form as a square-integral with the reciprocal of the Fourier transform of the radial basis function as a weight.

Therefore, we have to include a polynomial from the nullspace of the semi-norm to the approximant which becomes

$$s(x) = \sum_{\xi\in\Xi} \lambda_\xi \varphi(\|x-\xi\|) + q(x), \quad x \in \mathbb{R}^n,$$

where $q \in \mathbb{P}_n^k$ and the side conditions on the coefficients

$$\sum_{\xi\in\Xi} \lambda_\xi p(\xi) = 0, \quad \forall\, p \in \mathbb{P}_n^k.$$

If we consider the regularization network problem with the smoothing parameter μ again, then we have to solve the linear system with a smoothing parameter μ

$$(A + \mu I)\lambda + P^{\mathrm{T}} b = f, \quad P\lambda = 0,$$

where $P = (p_i(\xi))_{i=1,\ldots,L,\xi\in\Xi}$, and p_i form a basis of $\mathbb{P}_n^k$, b_i being the components of b, and $q(x) = \sum_{i=1}^L b_i p_i(x)$ is the expression of the polynomial added to the radial basis function sum. So in particular P is a matrix with as many rows as the dimension $L = \binom{n+k}{n}$ of $\mathbb{P}_n^k$ is and $|\Xi|$ columns.

In all cases, the radial basis functions composed with the Euclidean norm can be regarded as reproducing kernels in the semi-Hilbert spaces defined by the set X of distributions g for which $\|g\|_\varphi$ is finite and the semi-inner product

$$(h,g) = \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} \frac{1}{\hat\varphi(\|x\|)} \hat h(x) \overline{\hat g(x)}\, \mathrm{d}x, \quad h, g, \in X.$$

In particular, $\|g\|_\varphi^2 = (g,g)$. If the evaluation functional is continuous (bounded) on that space X, there exists a reproducing kernel, i.e., there is a $K : X \times X \to \mathbb{R}$ such that

$$g(x) = (g, K(\cdot, x)), \quad \forall\, x \in \mathbb{R}^n,\ g \in X,$$

see, for example, Wahba (1990). If the semi-inner product is actually an inner product, then the reproducing kernel is unique. The kernel gives rise to positive definite matrices $\{K(\xi,\zeta)\}_{\xi,\zeta\in\Xi}$ if and only if it is a positive operator. For the spaces X defined by our radial basis functions, it turns out that $K(x,y) := \varphi(\|x-y\|)$, see, e.g., the overview in Buhmann (2003). Then the matrices A are positive definite if $\hat{\varphi}(\|\cdot\|)$ is well-defined and positive, but if it has a singularity at zero, the A may be only conditionally positive definite. Note here that $\hat{\varphi}(\|\cdot\|)$ denotes the n-variate Fourier transform of $\varphi(\|\cdot\|)$, both being radially symmetric.

Advantages of the Approach

Why are we interested in using radial basis functions for networks? The radial basis functions have many excellent approximation properties which make them useful as general tools for approximation. Among them are the variety of more or less smoothness as required (e.g., multiquadrics is C^∞ for positive c and just continuous for $c = 0$), the fast evaluation and computation methods available (see, e.g., Beatson & Powell, 1994), the aforementioned nonsingularity properties and their connection with the theory of reproducing kernel Hilbert spaces, and finally their excellent convergence properties (see, e.g., Buhmann, 2003). Generally, neural networks are a tried and tested approach to approximation, modeling, and smoothing by methods from learning theory.

Limitations

The number of applications where the radial basis function approach has been used is vast. Also, the solutions may be computed efficiently by far field expansions, approximated Lagrange functions, and multipole methods. However, there are still some limitations with these important computational methods when the dimension n is large. So far, most of the multipole and far field methods have been implemented only for medium-sized dimensions.

Cross References

►Neural Networks
►Regularization
►Support Vector Machines

Recommended Reading

Beatson, R. K., & Powell, M. J. D. (1994). An iterative method for thin plate spline interpolation that employs approximations to Lagrange functions. In D. F. Griffiths & G. A. Watson (Eds.), *Numerical analysis 1993* (pp. 17–39). Burnt Mill: Longman.

Broomhead, D., & Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks, *Complex Systems, 2*, 321–355.

Buhmann, M. D. (1990). Multivariate cardinal-interpolation with radial-basis functions. *Constructive Approximation, 6*, 225–255.

Buhmann, M. D. (1998). Radial functions on compact support. *Proceedings of the Edinburgh Mathematical Society, 41*, 33–46.

Buhmann, M. D. (2003). *Radial basis functions: Theory and implementations*. Cambridge: Cambridge University Press.

Duchon, J. (1976). Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. *RAIRO, 10*, 5–12.

Evgeniou, T., Poggio, T., & Pontil, M. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics, 13*, 1–50.

Hardy, R. L. (1990). Theory and applications of the multiquadric-biharmonic method. *Computers and Mathematics with Applications, 19*, 163–208.

Micchelli, C. A. (1986). Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation, 1*, 11–22.

Pinkus, A. (1996). TDI-subpaces of $C(\mathbb{R}^d)$ and some density problems from neural networks. *Journal of Approximation Theory, 85*, 269–287.

Schoenberg, I. J. (1938). Metric spaces and completely monotone functions. *Annals of Mathematics, 39*, 811–841.

Tichonov, A. N., & Arsenin, V. Y. (1977). *Solution of ill-posed problems*. Washington, DC: V.H. Winston.

Vapnik, V. N. (1996). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1985). A comparison of GCV and GML for choosing the smoothing parameter in the generalized splines smoothing problem. *Annals of Statistics, 13*, 1378–1402.

Wahba, G. (1990). *Spline models for observational data. Series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Radial Basis Function Neural Networks

►Radial Basis Function Networks

Random Decision Forests

►Random Forests

Random Forests

Synonyms
Random decision forests

Definition
Random Forests is an ►ensemble learning technique. It is a hybrid of the ►Bagging algorithm and the ►random subspace method, and uses ►decision trees as the base classifier. Each tree is constructed from a bootstrap sample from the original dataset. An important point is that the trees are not subjected to pruning after construction, enabling them to be partially overfitted to their own sample of the data. To further diversify the classifiers, at each branch in the tree, the decision of which feature to split on is restricted to a *random subset* of size n, from the full feature set. The random subset is chosen anew for each branching point. n is suggested to be $\log_2(N+1)$, where N is the size of the whole feature set.

Random Subspace Method

Synonyms
Random subspaces; RSM

Definition
The random subspace method is an ►ensemble learning technique. The principle is to increase diversity between members of the ensemble by restricting classifiers to work on different random subsets of the full ►feature space. Each classifier learns with a subset of size n, chosen uniformly at random from the full set of size N. Empirical studies have suggested good results can be obtained with the rule-of-thumb to choose $n = N/2$ features. The method is generally found to perform best when there are a large number of features (large N), and the discriminative information is spread across them. The method can underperform in the converse situation, when there are few informative features, and a large number of noisy/irrelevant features. ►Random Forests is an algorithm combining RSM with the ►Bagging algorithm, which can provide significant gains over each used separately.

Random Subspaces

►Random Subspace Method

Randomized Decision Rule

►Markovian Decision Rule

Rank Correlation

Definition
Rank correlation measures the correspondence between two rankings τ and τ' of a set of m objects. Various proposals for such measures have been made, especially in the field of statistics. Two of the best-known measures are Spearman's Rank Correlation and Kendall's tau:
Spearman's Rank correlation calculates the sum of squared rank distances and is normalized such that it evaluates to −1 for reversed and to +1 for identical rankings. Formally, it is defined as follows:

$$(\tau, \tau') \mapsto 1 - \frac{6\sum_{i=1}^{m}(\tau(i) - \tau'(i))^2}{m(m^2-1)} \tag{1}$$

Kendall's tau is the number of pairwise rank inversions between τ and τ', again normalized to the range $[-1, +1]$:

$$(\tau, \tau') \mapsto 1 - \frac{4\,|\{(i,j) \mid i<j,\ \tau(i)<\tau(j) \wedge \tau'(i)>\tau'(j)\}|}{m(m-1)} \tag{2}$$

Cross References
►Preference Learning
►ROC Analysis

Ratio Scale

A **ratio** measurement scale possesses all the characteristics of interval measurement, and there exists a *zero* that, the same as arithmetic *zero*, means "nil" or "nothing." See ►Measurement Scales.

Real-Time Dynamic Programming

Real-Time Dynamic Programming (RTDP) is the same as ►Adaptive Real-Time Dynamic Programming (ARTDP) without the system identification component. It is applicable when an accurate model of the problem is available. It converges to an optimal policy of a stochastic optimal path problem under suitable conditions. RTDP was introduced by Barto, Bradtke, and Singh (1995) in their paper Learning to Act Using RTDP.

Recall

Recall is a measure of information retrieval performance. Recall is the total number of documents retrieved that are elevant/Total number of relevant documents in the database. See ►Precision and Recall.

Cross References

►Sensitivity

Receiver Operating Characteristic Analysis

►ROC Analysis

Recognition

►Classification

Recommender Systems

Prem Melville, Vikas Sindhwani
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA

Definition

The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them. Suggestions for books on Amazon, or movies on Netflix, are real-world examples of the operation of industry-strength recommender systems. The design of such recommendation engines depends on the domain and the particular characteristics of the data available. For example, movie watchers on Netflix frequently provide ratings on a scale of 1 (disliked) to 5 (liked). Such a data source records the quality of interactions between users and items. Additionally, the system may have access to user-specific and item-specific profile attributes such as demographics and product descriptions, respectively. Recommender systems differ in the way they analyze these data sources to develop notions of affinity between users and items, which can be used to identify well-matched pairs. ►Collaborative Filtering systems analyze historical interactions alone, while ►Content-based Filtering systems are based on profile attributes; and hybrid techniques attempt to combine both of these designs. The architecture of recommender systems and their evaluation on real-world problems is an active area of research.

Motivation and Background

Obtaining recommendations from trusted sources is a critical component of the natural process of human decision making. With burgeoning consumerism buoyed by the emergence of the web, buyers are being presented with an increasing range of choices while sellers are being faced with the challenge of personalizing their advertising efforts. In parallel, it has become common for enterprises to collect large volumes of transactional data that allows for deeper analysis of how a customer base interacts with the space of product offerings. Recommender systems have evolved to fulfill the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis.

The term "collaborative filtering" was introduced in the context of the first commercial recommender system, called Tapestry (Goldberg, Nichols, Oki, & Terry, 1992), which was designed to recommend documents drawn from newsgroups to a collection of users. The motivation was to leverage social collaboration in order to prevent users from getting inundated by a large volume of streaming documents. Collaborative filtering, which analyzes usage data across users to find well-matched user-item pairs, has since been juxtaposed against the older methodology of content filtering, which had its original roots in information retrieval. In content filtering, recommendations are not "collaborative" in the sense that suggestions made to a user do not explicitly utilize information across the entire user-base. Some early successes of collaborative filtering on related domains included the GroupLens system (Resnick, Neophytos, Bergstrom, Mitesh, & Riedl, 1994b).

As noted in Billsus and Pazzani (1998), initial formulations for recommender systems were based on straightforward correlation statistics and predictive modeling, not engaging the wider range of practices in statistics and machine learning literature. The collaborative filtering problem was mapped to classification, which allowed dimensionality reduction techniques to be brought into play to improve the quality of the solutions. Concurrently, several efforts attempted to combine content-based methods with collaborative filtering, and to incorporate additional domain knowledge in the architecture of recommender systems.

Further research was spurred by the public availability of datasets on the web, and the interest generated due to direct relevance to e-commerce. Netflix, an online streaming video and DVD rental service, released a large-scale dataset containing 100 million ratings given by about half-a-million users to thousands of movie titles, and announced an open competition for the best collaborative filtering algorithm in this domain. Matrix Factorization (Bell, Koren, & Volinsky, 2009) techniques rooted in numerical linear algebra and statistical matrix analysis emerged as a state-of-the-art technique.

Currently, recommender systems remain an active area of research, with a dedicated ACM conference, intersecting several subdisciplines of statistics, machine learning, data mining, and information retrievals. Applications have been pursued in diverse domains ranging from recommending webpages to music, books, movies, and other consumer products.

Structure of Learning System

The most general setting in which recommender systems are studied is presented in Fig. 1. Known user

$$\begin{array}{cc|c|c|c|c|c|c|}
 & & \multicolumn{6}{c}{\textit{Items}} \\
 & & 1 & 2 & \dots & i & \dots & m \\
\cline{3-8}
 & 1 & 5 & 3 & & 1 & 2 & \\
\cline{3-8}
 & 2 & & 2 & & & & 4 \\
\cline{3-8}
\textit{Users} & \vdots & & & 5 & & & \\
\cline{3-8}
 & u & 3 & 4 & & 2 & 1 & \\
\cline{3-8}
 & \vdots & & & & & 4 & \\
\cline{3-8}
 & n & & & 3 & 2 & & \\
\cline{3-8}
\multicolumn{8}{c}{} \\
\cline{3-8}
 & a & 3 & 5 & & ? & 1 & \\
\cline{3-8}
\end{array}$$

Recommender Systems. Figure 1. User ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user *u* for item *i*. The task is to predict the missing rating $r_{a,i}$ for the active user *a*

preferences are represented as a matrix of n users and m items, where each cell $r_{u,i}$ corresponds to the rating given to item i by the user u. This *user ratings matrix* is typically sparse, as most users do not rate most items. The *recommendation task* is to predict what rating a user would give to a previously unrated item. Typically, ratings are predicted for all items that have not been observed by a user, and the highest rated items are presented as recommendations. The user under current consideration for recommendations is referred to as the *active user*.

The myriad approaches to recommender systems can be broadly categorized as:

- *Collaborative Filtering (CF)*: In CF systems, a user is recommended items based on the past ratings of all users collectively.
- *Content-based recommending*: These approaches recommend items that are similar in content to items the user has liked in the past, or matched to pre-defined attributes of the user.
- *Hybrid approaches*: These methods combine both collaborative and content-based approaches.

Collaborative Filtering

Collaborative filtering (CF) systems work by collecting user feedback in the form of ratings for items in a given domain and exploiting similarities in rating behavior amongst several users in determining how to recommend an item. CF methods can be further subdivided into *neighborhood-based* and *model-based* approaches. Neighborhood-based methods are also commonly referred to as *memory-based* approaches (Breese, Heckerman, & Kadie, 1998).

Neighborhood-based Collaborative Filtering In neighborhood-based techniques, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. Most of these approaches can be generalized by the algorithm summarized in the following steps:

1. Assign a weight to all users with respect to similarity with the active user.
2. Select k users that have the highest similarity with the active user – commonly called the *neighborhood*.
3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

In step 1, the weight $w_{a,u}$ is a measure of similarity between the user u and the active user a. The most commonly used measure of similarity is the Pearson correlation coefficient between the ratings of the two users (Resnick, Iacovou, Sushak, Bergstrom, & Reidl, 1994a), defined below:

$$w_{a,u} = \frac{\sum_{i\in I}(r_{a,i}-\bar{r}_a)(r_{u,i}-\bar{r}_u)}{\sqrt{\sum_{i\in I}(r_{a,i}-\bar{r}_a)^2\sum_{i\in I}(r_{u,i}-\bar{r}_u)^2}} \tag{1}$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\bar{r}_u$ is the mean rating given by user u.

In step 3, predictions are generally computed as the weighted average of deviations from the neighbor's mean, as in:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u\in K}(r_{u,i}-\bar{r}_u)\times w_{a,u}}{\sum_{u\in K} w_{a,u}} \tag{2}$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

Similarity based on Pearson correlation measures the extent to which there is a linear dependence between two variables. Alternatively, one can treat the ratings of two users as a vector in an m-dimensional space, and compute similarity based on the cosine of the angle between them, given by:

$$\begin{aligned} w_{a,u} &= \cos(\boldsymbol{r}_a,\boldsymbol{r}_u) = \frac{\boldsymbol{r}_a\cdot\boldsymbol{r}_u}{\|\boldsymbol{r}_a\|_2\times\|\boldsymbol{r}_u\|_2} \\ &= \frac{\sum_{i=1}^m r_{a,i}r_{u,i}}{\sqrt{\sum_{i=1}^m r_{a,i}^2}\sqrt{\sum_{i=1}^m r_{u,i}^2}} \end{aligned} \tag{3}$$

When computing cosine similarity, one cannot have negative ratings, and unrated items are treated as having a rating of zero. Empirical studies (Breese et al., 1998) have found that Pearson correlation generally performs better. There have been several other similarity measures used in the literature, including *Spearman rank correlation*, *Kendall's τ correlation*, *mean squared differences*, *entropy*, and *adjusted cosine similarity* (Herlocker, Konstan, Borchers, & Riedl, 1999; Su & Khoshgoftaar, 2009).

Several extensions to neighborhood-based CF, which have led to improved performance are discussed below.

Item-based Collaborative Filtering: When applied to millions of users and items, conventional neighborhood-based CF algorithms do not scale well, because of the computational complexity of the search for similar users. As a alternative, Linden, Smith, and York (2003) proposed *item-to-item* collaborative filtering where rather than matching similar users, they match a user's rated items to similar items. In practice, this approach leads to faster online systems, and often results in improved recommendations (Linden et al., 2003; Sarwar, Karypis, Konstan, & Reidl, 2001).

In this approach, similarities between pairs of items i and j are computed off-line using Pearson correlation, given by:

$$w_{i,j} = \frac{\sum_{u\in U}(r_{u,i}-\bar{r}_i)(r_{u,j}-\bar{r}_j)}{\sqrt{\sum_{u\in U}(r_{u,i}-\bar{r}_i)^2}\sqrt{\sum_{u\in U}(r_{u,j}-\bar{r}_j)^2}} \tag{4}$$

where U is the set of all users who have rated both items i and j, $r_{u,i}$ is the rating of user u on item i, and $\bar{r}_i$ is the average rating of the ith item across users.

Now, the rating for item i for user a can be predicted using a simple weighted average, as in:

$$p_{a,i} = \frac{\sum_{j\in K} r_{a,j}w_{i,j}}{\sum_{j\in K}|w_{i,j}|} \tag{5}$$

where K is the neighborhood set of the k items rated by a that are most similar to i.

For item-based collaborative filtering too, one may use alternative similarity metrics such as *adjusted cosine similarity*. A good empirical comparison of variations of item-based methods can be found in Sarwar et al. (2001).

Significance Weighting: It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) items. These neighbors based on a small number of overlapping items tend to be bad predictors. One approach to tackle this problem is to multiply the similarity weight by a *significance weighting* factor, which devalues the correlations based on few co-rated items (Herlocker et al., 1999).

Default Voting: An alternative approach to dealing with correlations based on very few co-rated items is to assume a default value for the rating for items that have not been explicitly rated. In this way one can now compute correlation (Eq. 1) using the union of items rated by users being matched as opposed to the intersection. Such a *default voting* strategy has been shown to improve collaborative filtering by Breese et al. (1998).

Inverse User Frequency: When measuring the similarity between users, items that have been rated by all (and universally liked or disliked) are not as useful as less common items. To account for this Breese et al. (1998) introduced the notion of *inverse user frequency*, which is computed as $f_i = \log n/n_i$, where n_i is the number of users who have rated item i out of the total number of n users. To apply inverse user frequency while using similarity-based CF, the original rating is transformed for i by multiplying it by the factor f_i. The underlying assumption of this approach is that items that are universally loved or hated are rated more frequently than others.

Case Amplification: In order to favor users with high similarity to the active user, Breese et al. (1998) introduced *case amplification* which transforms the original weights in Eq. (2) to

$$w'_{a,u} = w_{a,u} \cdot |w_{a,u}|^{\rho-1}$$

where ρ is the amplification factor, and $\rho \geq 1$.

Other notable extensions to similarity-based collaborative filtering include *weighted majority prediction* (Nakamura & Abe, 1998) and *imputation-boosted CF* (Su, Khoshgoftaar, Zhu, & Greiner, 2008).

Model-based Collaborative Filtering Model-based techniques provide recommendations by estimating parameters of statistical models for user ratings. For example, Billsus and Pazzani (1998) describe an early approach to map CF to a classification problem, and build a classifier for each active user representing items as features over users and available ratings as labels, possibly in conjunction with dimensionality reduction techniques to overcome data sparsity issues. Other predictive modeling techniques have also been applied in closely related ways.

More recently, ►latent factor and matrix factorization models have emerged as a state-of-the-art methodology in this class of techniques (Bell et al., 2009). Unlike neighborhood based methods that generate recommendations based on statistical notions of similarity between users, or between items, latent factor models assume that the similarity between users and items is simultaneously induced by some hidden lower-dimensional structure in the data. 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 where users and items are simultaneously represented as unknown feature vectors (column vectors) $w_u, h_i \in \mathbb{R}^k$ along k latent dimensions. These feature vectors are learnt so that inner products $w_u^T h_i$ approximate the known preference ratings $r_{u,i}$ with respect to some loss measure. The squared loss is a standard choice for the loss function, in which case the following objective function is minimized,

$$J(W,H) = \sum_{(u,i)\in L} \left(r_{u,i} - w_u^T h_i\right)^2 \tag{6}$$

where $W = [w_1 \ldots w_n]^T$ is an $n \times k$ matrix, $H = [h_1 \ldots h_m]$ is a $k \times m$ matrix, and L is the set of user-item pairs for which the ratings are known. In the impractical limit where all user-item ratings are known, the above objective function is $J(W,H) = \|R - WH\|^2_{fro}$

where R denotes the $n \times m$ fully known user-item matrix. The solution to this problem is given by taking the truncated SVD of R, $R = UDV^T$ and setting $W = U_k D_k^{\frac{1}{2}}, H = D_k^{\frac{1}{2}} V_k^T$ where U_k, D_k, V_k contain the k largest singular triplets of R. However, in the realistic setting where the majority of user-item ratings are unknown and insufficient number of matrix entries are observed, such a nice globally optimal solution cannot in general be directly obtained, and one has to explicitly optimize the non-convex objective function $J(W,H)$. Note that in this case, the objective function is a particular form of weighted loss, that is, $J(W,H) = \|S \odot (R - WH)\|_{fro}^2$ where $\odot$ denotes elementwise products, and S is a binary matrix that equals one over known user-item pairs L, and 0 otherwise. Therefore, weighted low-rank approximations are pertinent to this discussion (Srebro & Jaakkola, 2003). Standard optimization procedures include gradient-based techniques, or procedures like alternating least squares where H is solved keeping W fixed and vice versa until a convergence criterion is satisfied. Note that fixing either W or H turns the problem of estimating the other into a weighted ►linear regression task. In order to avoid learning a model that overfits, it is common to minimize the objective function in the presence of ►regularization terms, $J(W,H) + \gamma\|W\|^2 + \lambda\|H\|^2$, where γ, λ are regularization parameters that can be determined by cross-validation. Once W, H are learnt, the product WH provides an approximate reconstruction of the rating matrix from where recommendations can be directly read off.

Different choices of loss functions, regularizers, and additional model constraints have generated a large body of literature on matrix factorization techniques. Arguably, for discrete ratings, the squared loss is not the most natural loss function. The maximum margin matrix factorization (Rennie & Srebro, 2005) approach uses margin-based loss functions such as the hinge loss used in ►SVM classification, and its ordinal extensions for handling multiple ordered rating categories. For ratings that span over K values, this reduces to finding $K - 1$ thresholds that divide the real line into consecutive intervals specifying rating bins to which the output is mapped, with a penalty for insufficient margin of separation. Rennie and Srebro (2005) suggest a nonlinear conjugate gradient algorithm to minimize a smoothed version of this objective function.

Another class of techniques is the nonnegative matrix factorization popularized by the work of Lee and Seung (1999) where nonnegativity constraints are imposed on W, H. There are weighted extensions of NMF that can be applied to recommendation problems. The rating behavior of each user may be viewed as being a manifestation of different roles, for example, a composition of prototypical behavior in clusters of users bound by interests or community. Thus, the ratings of each user are an additive sum of basis vectors of ratings in the item space. By disallowing subtractive basis, nonnegativity constraints lend a "part-based" interpretation to the model. NMF can be solved with a variety of loss functions, but with the generalized KL-divergence loss defined as follows,

$$J(W,H) = \sum_{u,i \in L} r_{u,i} \log \frac{r_{u,i}}{w_u^T h_i} - r_{u,i} + w_u^T h_i$$

NMF is in fact essentially equivalent to probabilistic latent semantic analysis (pLSA) which has also previously been used for collaborative filtering tasks (Hofmann, 2004).

The recently concluded million-dollar Netflix competition has catapulted matrix factorization techniques to the forefront of recommender technologies in collaborative filtering settings (Bell et al., 2009). While the final winning solution was a complex ensemble of different models, several enhancements to basic matrix factorization models were found to lead to improvements. These included:

1. The use of additional user-specific and item-specific parameters to account for systematic biases in the ratings such as popular movies receiving higher ratings on average.
2. Incorporating temporal dynamics of rating behavior by introducing time-dependent variables.

In many settings, only implicit preferences are available, as opposed to explicit like–dislike ratings. For example, large business organizations, typically, meticulously record transactional details of products purchased by their clients. This is a one-class setting since the business domain knowledge for negative examples – that a client has no interest in buying a product ever in the future – is typically not available explicitly in corporate databases. Moreover, such knowledge is difficult to gather and maintain in the first place, given the rapidly changing business environment. Another example is recommending TV shows based on watching habits of users, where preferences are implicit in what the users chose to see without any source of explicit ratings. Recently, matrix factorization techniques have been advanced to handle such problems (Pan & Scholz, 2009) by formulating confidence weighted objective function, $J(W,H) = \sum_{(u,i)} c_{u,i}\left(r_{u,i} - w_u^T h_i\right)^2$, under the assumption that unobserved user-item pairs may be taken as negative examples with a certain degree of confidence specified via $c_{u,i}$.

The problem of recovering missing values in a matrix from a small fraction of observed entries is also known as the Matrix Completion problem. Recent work by Candès & Tao (2009) and Recht (2009) has shown that under certain assumptions on the singular vectors of the matrix, the matrix completion problem can be solved exactly by a convex optimization problem provided with a sufficient number of observed entries. This problem involves finding among all matrices consistent with the observed entries, the one with the minimum nuclear norm (sum of singular values).

Content-based Recommending

Pure collaborative filtering recommenders only utilize the user ratings matrix, either directly, or to induce a collaborative model. These approaches treat all users and items as atomic units, where predictions are made without regard to the specifics of individual users or items. However, one can make a better personalized recommendation by knowing more about a user, such as demographic information (Pazzani, 1999), or about an item, such as the director and genre of a movie (Melville, Mooney, & Nagarajan, 2002). For instance, given movie genre information, and knowing that a user liked "Star Wars" and "Blade Runner," one may infer a predilection for science fiction and could hence recommend "Twelve Monkeys." Content-based recommenders refer to such approaches, that provide recommendations by comparing representations of content describing an item to representations of content that interests the user. These approaches are sometimes also referred to as *content-based filtering*.

Much research in this area has focused on recommending items with associated *textual* content, such as web pages, books, and movies; where the web pages themselves or associated content like descriptions and user reviews are available. As such, several approaches have treated this problem as an information retrieval (IR) task, where the content associated with the user's preferences is treated as a query, and the unrated documents are scored with relevance/similarity to this query (Balabanovic & Shoham, 1997). In NewsWeeder (Lang, 1995), documents in each rating category are converted into *tf-idf* word vectors, and then averaged to get a prototype vector of each category for a user. To classify a new document, it is compared with each prototype vector and given a predicted rating based on the cosine similarity to each category.

An alternative to IR approaches, is to treat recommending as a classification task, where each example represents the content of an item, and a user's past ratings are used as labels for these examples. In the domain of book recommending, Mooney and Roy (2000) use text from fields such as the title, author, synopses, reviews, and subject terms, to train a multinomial ►naïve Bayes classifier. Ratings on a scale of 1 to k can be directly mapped to k classes (Melville et al., 2002), or alternatively, the numeric rating can be used to weight the training example in a probabilistic binary classification setting (Mooney & Roy, 2000). Other classification algorithms have also been used for purely content-based recommending, including ►k-nearest neighbor, ►decision trees, and ►neural networks (Pazzani & Billsus, 1997).

Hybrid Approaches

In order to leverage the strengths of content-based and collaborative recommenders, there have been several hybrid approaches proposed that combine the two. One simple approach is to allow both content-based and collaborative filtering methods to produce separate ranked lists of recommendations, and then merge their results to produce a final list (Cotter & Smyth, 2000). Claypool, Gokhale, and Miranda (1999) combine the two predictions using an adaptive weighted average, where the weight of the collaborative component increases as the number of users accessing an item increases.

Melville et al. (2002) proposed a general framework for *content-boosted collaborative filtering*, where content-based predictions are applied to convert a sparse user ratings matrix into a full ratings matrix, and then a CF method is used to provide recommendations. In particular, they use a Naïve Bayes classifier trained on documents describing the rated items of each user, and replace the unrated items by predictions from this classifier. They use the resulting *pseudo ratings matrix* to find neighbors similar to the active user, and produce predictions using Pearson correlation, appropriately weighted to account for the overlap of actually rated items, and for the active user's content predictions. This approach has been shown to perform better than pure collaborative filtering, pure content-based systems, and a linear combination of the two. Within this content-boosted CF framework, Su, Greiner, Khoshgoftaar, and Zhu (2007) demonstrated improved results using a stronger content-predictor, TAN-ELR, and unweighted Pearson collaborative filtering.

Several other hybrid approaches are based on traditional collaborative filtering, but also maintain a content-based profile for each user. These content-based profiles, rather than co-rated items, are used to find similar users. In Pazzani's approach (Pazzani, 1999), each user-profile is represented by a vector of weighted words derived from positive training examples using the Winnow algorithm. Predictions are made by applying CF directly to the matrix of user-profiles (as opposed to the user-ratings matrix). An alternative approach, Fab (Balabanovic & Shoham, 1997), uses ►relevance feedback to simultaneously mold a personal filter along with a communal "topic" filter. Documents are initially ranked by the topic filter and then sent to a user's personal filter. The user's relevance feedback is used to modify both the personal filter and the originating topic filter. Good et al. (1999) use collaborative filtering along with a number of personalized information filtering agents. Predictions for a user are made by applying CF on the set of other users and the active user's personalized agents.

Several hybrid approaches treat recommending as a classification task, and incorporate collaborative elements in this task. Basu, Hirsh, and Cohen (1998) use *Ripper*, a ►rule induction system, to learn a function that takes a user and movie and predicts whether the movie will be liked or disliked. They combine collaborative and content information, by creating features such as *comedies liked by user* and *users who liked movies of genre X*. In other work, Soboroff and Nicholas (1999) multiply a *term-document matrix* representing all item content with the user-ratings matrix to produce a *content-profile matrix*. Using latent semantic Indexing, a rank-*k* approximation of the content-profile matrix is computed. Term vectors of the user's relevant documents are averaged to produce a user's profile. Then, new documents are ranked against each user's profile in the LSI space.

Some hybrid approaches attempt to directly combine content and collaborative data under a single probabilistic framework. Popescul, Ungar, Pennock, and Lawrence (2001) extended Hofmann's *aspect model* (Hofmann, 1999) to incorporate a three-way co-occurrence data among users, items, and item content. Their generative model assumes that users select latent topics, and documents and their content words are generated from these topics. Schein, Popescul, Ungar, and Pennock (2002) extend this approach, and focus on making recommendations for items that have not been rated by any user.

Evaluation Metrics

The quality of a recommender system can be evaluated by comparing recommendations to a test set of known user ratings. These systems are typical measured using *predictive accuracy metrics* (Herlocker, Konstan, Terveen, & Riedl, 2004), where the predicted ratings are directly compared to actual user ratings. The most commonly used metric in the literature is ►Mean Absolute Error (MAE) – defined as the average absolute difference between predicted ratings and actual ratings, given by:

$$MAE = \frac{\sum_{\{u,i\}} |p_{u,i} - r_{u,i}|}{N} \tag{7}$$

Where $p_{u,i}$ is the predicted rating for user u on item i, $r_{u,i}$ is the actual rating, and N is the total number of ratings in the test set.

A related commonly used metric, ►Root Mean Squared Error (RMSE), puts more emphasis on larger absolute errors, and is given by:

$$RMSE = \sqrt{\frac{\sum_{\{u,i\}} (p_{u,i} - r_{u,i})^2}{N}} \tag{8}$$

Predictive accuracy metrics treat all items equally. However, for most recommender systems the primary concern is accurately predict the items a user will like. As such, researchers often view recommending as predicting *good*, that is, items with high ratings versus *bad* or poorly rated items. In the context of information retrieval (IR), identifying the good from the background of bad items can be viewed as discriminating between "relevant" and "irrelevant" items; and as such, standard IR measures, like ►Precision, ►Recall and ►Area Under the ROC Curve (AUC) can be utilized. These, and several other measures, such as *F1-measure*, *Pearson's product-moment correlation*, *Kendall's* τ, *mean average precision*, *half-life utility*, and *normalized distance-based performance measure* are discussed in more detail by Herlocker et al. (2004).

Challenges and Limitations

This section, presents some of the common hurdles in deploying recommender systems, as well as some research directions that address them.

Sparsity: Stated simply, most users do not rate most items and, hence, the user ratings matrix is typically very sparse. This is a problem for collaborative filtering systems, since it decreases the probability of finding a set of users with similar ratings. This problem often occurs when a system has a very high item-to-user ratio, or the system is in the initial stages of use. This issue can be mitigated by using additional domain information (Melville et al., 2002; Su et al., 2007) or making assumptions about the data generation process that allows for high-quality imputation (Su et al., 2008).

The Cold-Start Problem: New items and new users pose a significant challenge to recommender systems. Collectively these problems are referred to as the *cold-start problem* (Schein et al., 2002). The first of these problems arises in collaborative filtering systems, where an item cannot be recommended unless some user has rated it before. This issue applies not only to new items, but also to obscure items, which is particularly detrimental to users with eclectic tastes. As such the *new-item problem* is also often referred to as the *first-rater problem*. Since content-based approaches (Mooney & Roy, 2000; Pazzani & Billsus, 1997) do not rely on ratings from other users, they can be used to produce recommendations for *all* items, provided attributes of the items are available. In fact, the content-based predictions of similar users can also be used to further improve predictions for the active user (Melville et al., 2002).

The *new-user problem* is difficult to tackle, since without previous preferences of a user it is not possible to find similar users or to build a content-based profile. As such, research in this area has primarily focused on effectively selecting items to be rated by a user so as to rapidly improve recommendation performance with the least user feedback. In this setting, classical techniques from ►active learning can be leveraged to address the task of item selection (Harpale & Yang, 2008; Jin & Si, 2004).

Fraud: As recommender systems are being increasingly adopted by commercial websites, they have started to play a significant role in affecting the profitability of sellers. This has led to many unscrupulous vendors engaging in different forms of fraud to game recommender systems for their benefit. Typically, they attempt to inflate the perceived desirability of their own products (*push attacks*) or lower the ratings of their competitors (*nuke attacks*). These types of attack have been broadly studied as *shilling attacks* (Lam & Riedl, 2004) or *profile injection attacks* (Burke, Mobasher, Bhaumik, & Williams, 2005). Such attacks usually involve setting up dummy profiles, and assume different amounts of knowledge about the system. For instance, the *average attack* (Lam & Riedl, 2004) assumes knowledge of the average rating for each item; and the attacker assigns values randomly distributed around this average, along with a high rating for the item being *pushed.* Studies have shown that such attacks can be quite detrimental to predicted ratings, though *item-based* collaborative filtering tends to be more robust to these attacks (Lam & Riedl, 2004). Obviously, content-based methods, which only rely on a users past ratings, are unaffected by profile injection attacks.

While pure content-based methods avoid some of the pitfalls discussed above, collaborative filtering still has some key advantages over them. Firstly, CF can perform in domains where there is not much content associated with items, or where the content is difficult for a computer to analyze, such as ideas, opinions, etc. Secondly, a CF system has the ability to provide serendipitous recommendations, that is, it can recommend items that are relevant to the user, but do not contain content from the user's profile.

Recommended Reading

Good surveys of the literature in the field can be found in Adomavicius and Tuzhilin (2005); Bell et al. (2009); Su and Khoshgoftaar (2009). For extensive empirical comparisons on variations of Collaborative Filtering refer to Breese (1998), Herlocker (1999), Sarwar et al. (2001).

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*(6), 734–749.

Balabanovic, M., & Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the Association for Computing Machinery, 40*(3), 66–72.

Basu, C., Hirsh, H., & Cohen, W. (July 1998). Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the fifteenth national conference on artificial intelligence (AAAI-98), Madison, Wisconsin* (pp. 714–720).

Bell, R., Koren, Y., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer 42*(8): 30–37.

Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the fifteenth international conference on machine learning (ICML-98), Madison, Wisconsin* (pp. 46–54). San Francisco: Morgan Kaufmann.

Breese, J. S., Heckerman, D., & Kadie, C. (July 1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the fourteenth conference on uncertainty in artificial intelligence, Madison, Wisconsin*.

Burke, R., Mobasher, B., Bhaumik, R., & Williams, C. (2005). Segment-based injection attacks against collaborative filtering recommender systems. In *ICDM '05: Proceedings of the fifth IEEE international conference on data mining* (pp. 577–580). Washington, DC: IEEE Computer Society. Houston, Texas.

Candès, E. J., & Tao, T. (2009). The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inform. Theory, 56*(5), 2053–2080.

Claypool, M., Gokhale, A., & Miranda, T. (1999). Combining content-based and collaborative filters in an online newspaper. In *Proceedings of the SIGIR-99 workshop on recommender systems: algorithms and evaluation*.

Cotter, P., & Smyth, B. (2000). PTV: Intelligent personalized TV guides. In *Twelfth conference on innovative applications of artificial intelligence, Austin, Texas* (pp. 957–964).

Goldberg, D., Nichols, D., Oki, B., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the Association of Computing Machinery, 35*(12), 61–70.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J., et al. (July 1999). Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on artificial intelligence (AAAI-99), Orlando, Florida* (pp. 439–446).

Harpale, A. S., & Yang, Y. (2008). Personalized active learning for collaborative filtering. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval, Singapore* (pp. 91–98). New York: ACM.

Herlocker, J., Konstan, J., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of 22nd international ACM SIGIR conference on research and development in information retrieval, Berkeley, California* (pp. 230–237). New York: ACM.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems, 22*(1), 5–53.

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 30-August 1, 1999 Morgan Kaufmann.

Hofmann, T. (2004). Latent semantic analysis for collaborative filtering. *ACM Transactions on Information Systems, 22*(1), 89–115.

Jin, R., & Si, L. (2004). A Bayesian approach toward active learning for collaborative filtering. In *UAI '04: Proceedings of the 20th conference on uncertainty in artificial intelligence, Banff, Canada* (pp. 278–285). Arlington: AUAI Press.

Lam, S. K., & Riedl, J. (2004). Shilling recommender systems for fun and profit. In *WWW '04: Proceedings of the 13th international conference on World Wide Web, New York* (pp. 393–402). New York: ACM.

Lang, K. (1995). NewsWeeder: Learning to filter netnews. In *Proceedings of the twelfth international conference on machine learning (ICML-95)* (pp. 331–339). San Francisco. Tahoe City, CA, USA. Morgan Kaufmann, ISBN 1-55860-377-8.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature, 401*, 788.

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*(1), 76–80.

Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the eighteenth national conference on artificial intelligence (AAAI-02), Edmonton, Alberta* (pp. 187–192).

Mooney, R. J., & Roy, L. (June 2000). Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on digital libraries, San Antonio, Texas* (pp. 195–204).

Nakamura, A., & Abe, N. (1998). Collaborative filtering using weighted majority prediction algorithms. In *ICML '98: Proceedings of the fifteenth international conference on machine learning* (pp. 395–403). San Francisco: Morgan Kaufmann. Madison, Wisconsin.

Pan, R., & Scholz, M. (2009). Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *15th ACM SIGKDD conference on knowledge discovery and data mining (KDD), Paris, France.*

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review, 13* (5–6), 393–408.

Pazzani, M. J., & Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning, 27*(3), 313–331.

Popescul, A., Ungar, L., Pennock, D. M., & Lawrence, S. (2001). Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the seventeenth conference on uncertainity in artificial intelligence*. University of Washington, Seattle.

Recht, B. (2009). A Simpler Approach to Matrix Completion. Benjamin Recht. (to appear in Journal of Machine Learning Research).

Rennie, J., & Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *International conference on machine learning, Bonn, Germany*.

Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., & Reidl, J. (1994a). GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 computer supported cooperative work conference, New York*. New York: ACM.

Resnick, P., Neophytos, I., Bergstrom, P., Mitesh, S., & Riedl, J. (1994b). Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW94 – Conference on computer supported cooperative work, Chapel Hill* (pp. 175–186). Addison-Wesley.

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the tenth international conference on World Wide Web* (pp. 285–295). New York: ACM. Hong Kong.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 253–260). New York: ACM. Tampere, Finland.

Soboroff, I., & Nicholas, C. (1999). Combining content and collaboration in text filtering. In T. Joachims (Ed.), *Proceedings of the IJCAI'99 workshop on machine learning in information filtering* (pp. 86–91).

Srebro, N., & Jaakkola, T. (2003). Weighted low-rank approximations. In *International conference on machine learning (ICML)*. Washington DC.

Su, X., Greiner, R., Khoshgoftaar, T. M., & Zhu, X. (2007). Hybrid collaborative filtering algorithms using a mixture of experts. In *Web intelligence* (pp. 645–649).

Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence, 2009*, 1–20.

Su, X., Khoshgoftaar, T. M., Zhu, X., & Greiner, R. (2008). Imputation-boosted collaborative filtering using machine learning classifiers. In *SAC '08: Proceedings of the 2008 ACM symposium on applied computing* (pp. 949–950). New York: ACM.

Record Linkage

►Entity Resolution

Recurrent Associative Memory

►Hopfield Network

Recursive Partitioning

►Divide-and-Conquer Learning

Reference Reconciliation

►Entity Resolution

Regression

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

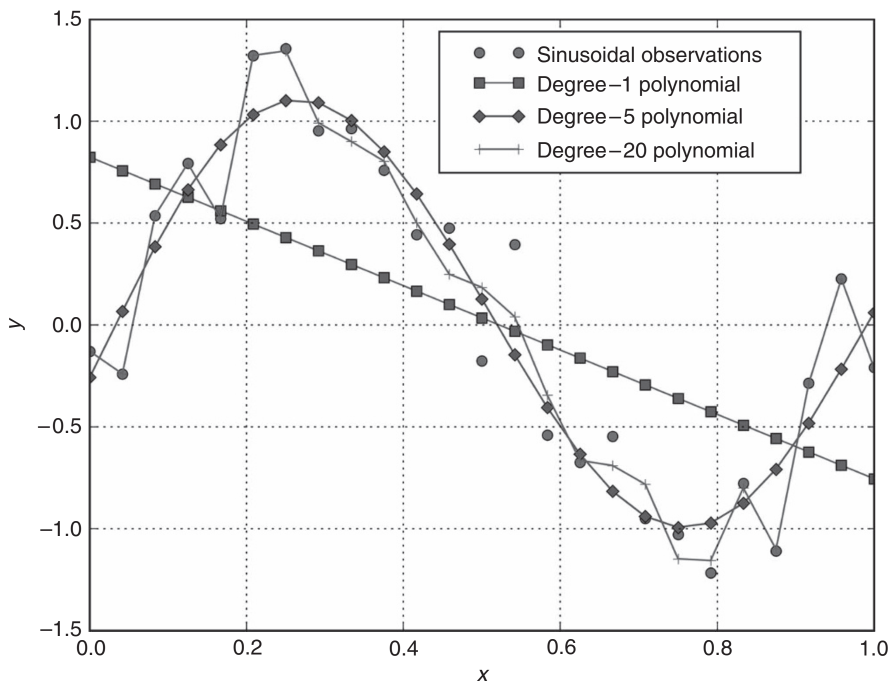
Definition

Regression is a fundamental problem in statistics and machine learning. In regression studies, we are typically interested in inferring a real-valued function (called a regression function) whose values correspond to the mean of a dependent (or response or output) variable conditioned on one or more independent (or input) variables. Many different techniques for estimating this regression function have been developed, including parametric, semi-parametric, and nonparametric methods.

Motivation and Background

Assume that we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y. An example is given in Fig. 1. The task of regression is to learn a hidden functional relationship between x and y from observed and possibly noisy data points. In Fig. 1, the input–output relationship is a Gaussian corrupted sinusoidal relationship, that is $y = \sin(2\pi x) + \epsilon$ where ϵ is normally distributed noise. Various lines show the inferred relationship based on a linear parametric regression model with polynomial basis functions. The higher the degree of the polynomial, the more complex is the inferred relationship, as shown in Fig. 1, as the function tries to better fit the observed data points.

While the most complex polynomial here is an almost perfect reconstruction of observed data points (it has "low bias"), it gives a very poor representation of



Regression. Figure 1. 25 data points (one-dimensional input *x* and output *y* variables) with a Gaussian corrupted sinusoidal input–output relationship, $y = \sin(2\pi x) + \epsilon$ where ϵ is normally distributed noise. The task is to learn the functional relationship between *x* and *y*. Various lines show the inferred relationship based on a linear regression model with polynomial basis functions having various degrees

the true underlying function $\sin(2\pi x)$ that can change significantly with the change of a few data points (it has "high variance"). This phenomemon is called the ▶*bias-variance dilemma*, and selecting a complex model with too high a variance is called ▶*over-fitting*. Complex parametric models (like polynomial regression) lead to low bias estimators with a high variance, while simple models lead to low variance estimators with high bias. To sidestep the problem of trying to estimate or select the model complexity represented for instance by the degree of the polynomial, so-called nonparametric methods allow a rich variety of functions from the outset (*i.e.*, a function class not finitely parameterizable) and usually provide a hyperparameter that tunes the regularity, curvature, or complexity of the function.

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 ▶Linear Regression, we represent the regression function $f()$ by a parameter $w \in \mathbb{R}^H$ in the form $f(x_i) := \langle \phi(x_i), w \rangle$ for H fixed basis functions $\{\phi_h(x_i)\}_{h=1}^H$. 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 non-linear with regard to the input variables although still linear with regard to the parameters.

In regression, many more methods are possible. Some variations on these standard linear models are piecewise linear models, trees, and splines (roughly, piecewise polynomial models joined up smoothly) (Hastie, Tibshirani, & Friedman, 2003). These are called semi-parametric models, because they have a linear parametric component as well as a nonparametric component.

Fitting In general, regression fits a model to data using an objective function or quality criterion in a form such as

R

$$E(f) = \sum_{i=1}^{N} \epsilon(y_i, f(x_i)),$$

where smaller $E(f)$ implies better quality. This might be derived as an error/loss function, or as a negative log likelihood or log probability. The squared error function is the most convenient (leading to a least squares calculation), but many possibilities exist. In general, methods are distinguished by three aspects, (1) the representation of the function $f()$, (2) the form of the term $\epsilon(y_i, f(x_i))$, and (3) the penalty term discussed next.

Regularized/Penalized Fitting The issue of over-fitting, as mentioned already in the section Motivation and Background, is usually addressed by introducing a regularization or penalty term to the objective function. The regularized objective function is now in the form of

$$E_{\text{reg}} = E(f) + \lambda R(f). \tag{1}$$

Here, $E(f)$ measures the quality of the solution for $f()$ on the observed data points, $R(f)$ penalizes complexity of $f()$, and λ is called the regularization parameter which controls the relative importance between the two. Measures of function curvature, for instance, can be used for $R(f)$. In standard ▶Support Vector Machines, the term $E(f)$ measures the hinge loss, and penalty $R(f)$ is the sum of squares of the parameters, also used in ridge regression (Hastie et al., 2003).

Bias-Variance Dilemma

As we have seen in the previous section, the introduction of the regularization term can help avoid over-fitting. However, this raises the question of determining an optimal value for the regularization parameter λ. The specific choice of λ controls the bias-variance tradeoff (Geman, Bienenstock, & Doursat, 1992).

Recall that we try to infer a latent regression function $f(x)$ based on N observed training data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$. The notation $f(x;\mathcal{D})$ explicitly shows the dependence of f on the data $\mathcal{D}$. The mean squared error (MSE) which measures the effectiveness of f as a predictor of y is

$$\mathbf{E}[(y - f(x;\mathcal{D}))^2 | x, \mathcal{D}] = \mathbf{E}[(y - \mathbf{E}[y|x])^2 | x, \mathcal{D}] + (f(x;\mathcal{D}) - \mathbf{E}[y|x])^2 \tag{2}$$

where $\mathbf{E}[.]$ means expectation with respect to a conditional distribution $p(y|x)$. The first term of (2) does not depend on $f(x;\mathcal{D})$ and it represents the intrinsic noise on the data. The MSE of f as an estimator of the regression $\mathbf{E}[y|x]$ is

$$\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}[y|x])^2] \tag{3}$$

where $\mathbf{E}_{\mathcal{D}}$ means expectation with respect to the training set $\mathcal{D}$. The estimation error in (3) can be decomposed into a bias and a variance terms, that is

$$\begin{aligned}
&\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}[y|x])^2] \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] + \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2] \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2] + (\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2 \\
&\quad + 2\mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])](\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x]) \\
&= \mathbf{E}_{\mathcal{D}}[(f(x;\mathcal{D}) - \mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})])^2] + (\mathbf{E}_{\mathcal{D}}[f(x;\mathcal{D})] \\
&\quad - \mathbf{E}[y|x])^2 \\
&= \text{variance} + \text{bias}^2.
\end{aligned}$$

The bias term measures the difference between the average predictor over all datasets and the desired regression function. The variance term measures the adaptability of the predictor to a particular dataset. There is a tradeoff between the bias and variance contributions to the estimation error, with very flexible models having low bias but high variance (over-fitting) and relatively rigid models having low variance but high bias (under-fitting). Typically, variance is reduced through "smoothing," that is an introduction of the regularization term. This, however, will introduce bias as peaks and valleys of the regression function will be blurred. To achieve an optimal predictive capability, an estimator with the best balance between bias and variance is chosen by varying the regularization parameter λ. It is crucial to note that bias-variance decomposition albeit powerful is based on averages of datasets, however in practice only a single dataset is observed. In this regard, a Bayesian treatment of regression, such as Gaussian process regression which will avoid over-fitting problem of maximum likelihood and which will also lead to automatic methods of determining model complexity using the training data alone could be an attractive alternative.

Nonparametric Regression

In the parametric approach, an assumption on the mathematical form of the functional relationship between input x and output y such as linear, polynomial, exponential, or combination of them needs to be chosen a priori. Subsequently, parameters are placed on each of the chosen forms and the optimal values learnt from the observed data. This is restrictive both in the fixed functional form and in the ability to vary the model complexity. Nonparametric approaches try to derive the functional relationship directly from the data, that is, they do not parameterize the regression function.

►Gaussian Processes for regression, for instance, is well-developed. Another approach is the *kernel method*, of which a rich variety exists (Hastie et al., 2003). These can be viewed as a regression variant of nearest neighbor classification where the function is made up of a local element for each data point:

$$f(x) = \frac{\sum_i y_i K_\lambda(x_i, x)}{\sum_i K_\lambda(x_i, x)},$$

where the function $K_\lambda(x_i,)$ is a nonnegative "bump" in x space centered at its first argument with diameter approximately given by λ. Thus, the function has a variable contribution from each data point, and λ controls the bias-variance tradeoff.

Generalized Linear Models

The previous discussion about regression focuses on continuous output/dependent variables. While this type of regression problem is ubiquitous, there are however some interests in cases of *restricted* output variables:

1. The output variable consists of two categories (called *binomial* regression).
2. The output variable consists of more than two categories (called *multinomial* regression).
3. The output variable consists of more than two categories which can be ordered in a meaningful way (called *ordinal* regression). and
4. The output variable is a count of the repetition of the occurrence of an event (called *poisson* regression).

Nelder and Wedderburn (1972) introduced the generalized linear model (GLM) by allowing the linear model to be related to the output variables via a link function. This is a way to unify different cases of response variables under one framework, each only differs in the choice of the link function. Specifically, in GLM, each output variable is assumed to be generated from the exponential family of distributions. The mean of this distribution depends on the input variables through

$$\mathbf{E}[y] = g(\mu) = w_0 + w_1\phi_1(x_i) + \ldots + w_D\phi_D(x_i), \quad (4)$$

where $g(\mu)$ is the link function (Table 1). The parameters of the generalized linear model can then be estimated by the maximum likelihood method, which can be found by iterative re-weighted least squares (IRLS), an instance of the expectation maximization (EM) algorithm.

Other Variants of Regression

So far, we have focused on the problem of predicting a single output variable y from an input variable x. Some studies look at predicting multiple output variables simultaneously. The simplest approach for the *multiple outputs* problem would be to model each output variable with a different set of basis functions. The more common approach uses the same set of basis functions to model all of the output variables. Not surprisingly,

Regression. Table 1 A table of Various Link Functions Associated with the Assumed Distribution on the Output Variable

Distribution of Dependent Variable	Name	Link Function
Gaussian	Identity link	$g(\mu) = \mu$
Poisson	Log link	$g(\mu) = \log(\mu)$
Binomial Multinomial	Logit link	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$
Exponential Gamma	Inverse link	$g(\mu) = \mu^{-1}$
Inverse Gaussian	Inverse squared link	$g(\mu) = \mu^{-2}$

the solution to the multiple outputs problem decouples into independent regression problems with shared basis functions.

For some other studies, the focus of regression is on computing several regression functions corresponding to various percentage points or quantiles (instead of the mean) of the conditional distribution of the dependent variable given the independent variables. This type of regression is called *quantile* regression (Koenker, 2005). Sum of tilted absolute loss (called pinball loss) is being optimized for this type of regression. Quantile regression has many important applications within econometrics, data mining, social sciences, and ecology, among other domains.

Instead of inferring one regression function corresponding to the mean of a response variable, k regression functions can be computed with the assumption that the response variable is generated by a mixture of k components. This is called the *mixture of regressions* problem (Gaffney & Smyth, 1999). Applications include trajectory clustering, robot planning, and motion segmentation.

Another important variant is the *heteroscedastic* regression model where the noise variance on the data is a function of the input variable x. The Gaussian process framework can be used conveniently to model this noise-dependent case by introducing a second Gaussian process to model the dependency of noise variance on the input variable (Goldberg, Williams, & Bishop, 1998). There are also attempts to make the regression model more robust to the presence of a few problematic data points called outliers. Sum of absolute loss (instead of sum of squared loss) or student's t-distribution (instead of Gaussian distribution) can be used for *robust* regression.

Cross References

►Gaussian Processes
►Linear Regression
►Support Vector Machines

Recommended Reading

Machine learning textbooks such as Bishop (2006), among others, introduce different regression models. For a more statistical introduction including an extensive overview of the many different semi-parametric methods and non-parametric methods such as kernel methods see Hastie et al. (2003). For a coverage of key statistical issues including nonlinear regression, identifiability, measures of curvature, autocorrelation, and such, see Seber and Wild (1989). For a large variety of built-in regression techniques, refer to R (http://www.r-project.org/).

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

Gaffney, S., & Smyth, P. (1999). Trajectory clustering with mixtures of regression models. In *ACM SIGKDD* (Vol. 62, pp. 63–72). ACM

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1–58.

Goldberg, P., Williams, C., & Bishop, C. (1998). Regression with input-dependent noise: A Gaussian process treatment. In *Neural information processing systems* (Vol. 10). The MIT Press

Hastie, T., Tibshirani, R., & Friedman, J. (2003). *The elements of statistical learning: Data mining, inference, and prediction* (Corrected ed.). Springer.

Koenker, R. (2005). *Quantile regression*. Cambridge University Press.

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A, 135*, 370–384.

Seber, G., & Wild, C. (1989). *Nonlinear regression*. New York: Wiley.

Regression Trees

Luís Torgo
University of Porto, Rua Campo Alegre, Porto, Portugal

Synonyms

Decision trees for regression; Piecewise constant models; Tree-based regression

Definition

Regression trees are supervised learning methods that address multiple regression problems. They provide a tree-based approximation $\hat{f}$, of an unknown regression function $Y = f(\mathbf{x}) + \varepsilon$ with $Y \in \mathbb{R}$ and $\varepsilon \approx N(0, \sigma^2)$, based on a given sample of data $D = \{\langle x_{i,1}, \ldots, x_{i,p}, y_i \rangle\}_{i=1}^{n}$. The obtained models consist of a hierarchy of logical tests on the values of any of the p predictor variables. The terminal nodes of these trees, known as the leaves, contain the numerical predictions of the model for the target variable Y.

Motivation and Background

Work on regression trees goes back to the AID system by Morgan and Sonquist Morgan (1963).

Nonetheless, the seminal work is the book *Classification and Regression Trees* by Breiman and colleagues (Breiman, Friedman, Olshen, & Stone, 1984). This book has established several standards in many theoretical aspects of tree-based regression, including over-fitting avoidance by post-pruning, the notion of surrogate splits for handling unknown variable, and estimating variable importance.

Regression trees have several features that make them a very interesting approach to several multiple regression problems. For example, regression trees provide: (a) automatic variable selection making them highly insensitive to irrelevant variables; (b) computational efficiency that allows addressing large problems; (c) handling of unknown variable values; (d) handling of both numerical and nominal predictor variables; (e) insensitivity to predictors' scales; and (f) interpretable models for most domains. In spite of all these advantages, regression trees have poor prediction accuracy in several domains because of the piecewise constant approximation they provide.

Structure of Learning System

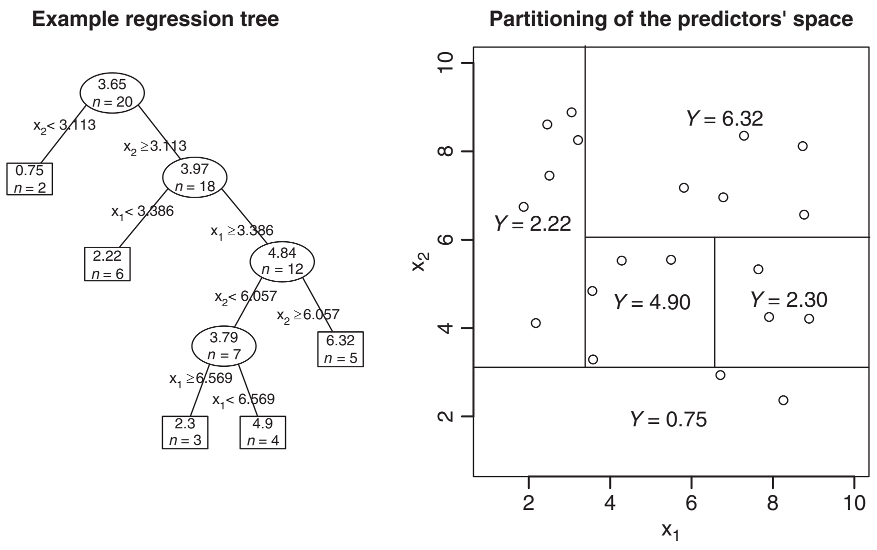
The most common regression trees are binary with logical tests in each node (an example is given on the left graph of Fig. 1). Tests on numerical variables usually take the form $x_i < \alpha$, with $\alpha \in \mathbb{R}$, while tests on nominal variables have the form $x_j \in \{v_1, \ldots, v_m\}$. Each path from the root (top) node to a leaf can be seen as a logical assertion defining a region on the predictors' space. Any regression tree provides a full mutually exclusive partition of the predictor space into L regions with boundaries that are parallel to the predictors' axes due to the form of the tests. Figure 1 illustrates these ideas with a tree and the respective partitioning on the right side of the graph. All observations in a partition are predicted with the same constant value, and that is the reason for regression trees sometimes being referred to as piecewise constant models.

Using a regression tree for obtaining predictions for new observations is straightforward. For each new observation a path from the root node to a leaf is followed, selecting the branches according to the variable values of the observation. The leaf contains the prediction for the observation.

Learning a Regression Tree

A binary regression tree is obtained by a very efficient algorithm known as recursive partitioning (Algorithm 1).

If the termination criterion is not met by the input sample D, the algorithm selects the best logical test on one of the predictor variables according to some criterion. This test divides the current sample into two partitions: the one with the cases satisfying the test, and the remaining. The algorithm proceeds by recursively



Regression Trees. Figure 1. A regression tree and the partitioning it provides

Algorithm 1 Recursive Partitioning.

```
1:  function RecursivePartitioning($D$)
    Input :    $D$, a sample of cases, $\{\langle x_{i,1}, \ldots, x_{i,p}, y_i \rangle\}$
    Output :   $t$, a tree node
2:      if <TERMINATION CRITERION> then
3:          Return a leaf node with <CONSTANT K>
4:      else
5:          $t \leftarrow$ new tree node
6:          $t.split \leftarrow$ <FIND THE BEST TEST ON ONE OF THE VARIABLES>
7:          $t.leftNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \vDash t.split$)
8:          $t.rightNode \leftarrow$ RecursivePartitioning($\mathbf{x} \in D : \mathbf{x} \nvDash t.split$)
9:          Return the node $t$
10:     end if
11: end function
```

applying the same method to these two partitions to obtain the left and right branches of the node. Algorithm 1 has three main components that characterize the type of regression tree we are obtaining: (a) the *termination criterion*; (b) the *constant k*; and (c) the method to find the *best test on one of the predictors*. The choices for these components are related to the preference criteria that are selected to build the trees. The most common criterion is the minimization of the sum of the square errors, known as the least squares (LS) criterion. Using this criterion it can be easily proven (e.g., Breiman et al., 1984) that the constant k should be the average target variable value of the cases in the leaf. With respect to the *termination criterion*, usually very relaxed settings are selected so that an overly large tree is grown. The reasoning is that the trees will be pruned afterward with the goal of overcoming the problem of over-fitting of the training data.

According to the LS criterion the error in a given node is given by,

$$Err(t) = \frac{1}{n_t} \sum_{\langle \mathbf{x}_i, y_i \rangle \in D_t} (y_i - k_t)^2 \qquad (1)$$

where D_t is the sample of cases in node t, n_t is the cardinality of this set and k_t is the average target variable value of the cases in D_t.

Any logical test s divides the cases in D_t into two partitions, D_{t_L} and D_{t_R}. The resulting pooled error is given by,

$$Err(t, s) = \frac{n_{t_L}}{n_t} \times Err(t_L) + \frac{n_{t_R}}{n_t} \times Err(t_R) \qquad (2)$$

where n_{t_L}/n_t (n_{t_R}/n_t) is the proportion of cases going to the left (right) branch of t.

In this context, we can estimate the value of the split s by the respective error reduction, and this can be used to evaluate all candidate split tests for a node,

$$\Delta(s, t) = Err(t) - Err(t, s) \qquad (3)$$

Finding the best split test for a node t involves evaluating all possible tests for this node using Eq. (3). For each predictor of the problem one needs to evaluate all possible splits in that variable. For continuous variables this requires a sorting operation on the values of this variable occurring in the node. After this sorting, a fast incremental algorithm can be used to find the best cut-point value for the test (e.g. Torgo, 1999). With respect to nominal variables, Breiman et al. (1984) have proved a theorem that avoids trying all possible combinations of values, reducing the computational complexity of this task from $O(2^{v-1} - 1)$ to $O(v - 1)$, where v is the number of values of the nominal variable.

Departures from the standard learning procedure described above include, among others: the use of multivariate split nodes (e.g., Breiman et al., 1984; Gama, 2004; Li, Lue, & Chen, 2000) to overcome the axis parallel representation limitation of partitions; the use of different criteria to find the best split node (e.g., Buja & Lee, 2001; Loh, 2002; Robnik-Sikonja & Kononenko, 1996); the use of different preference criteria to guide the tree growth (e.g., Breiman et al., 1984; Buja & Lee, 2001; Torgo, 1999; Torgo & Ribeiro, 2003); and the use of both regression and split nodes (e.g., Lubinsky, 1995; Malerba, Esposito, Ceci, & Appice, 2004).

Pruning Regression Trees

As most nonparametric modeling techniques, regression trees may ►over-fit the training data which will inevitably lead to poor out-of-the-sample predictive performance. The standard procedure to fight this undesirable effect is to grow an overly large tree and then to use some reliable error estimation procedure to find the "best" sub-tree of this large model. This procedure is known as post-pruning a tree (Breiman et al., 1984). An alternative is to stop tree growth sooner in a process known as pre-pruning, which again needs to be guided by reliable error estimation to know when over-fitting is starting to occur. Although more efficient in computational terms, this latter alternative may lead to stop tree growth too soon even with look-ahead mechanisms.

Post-pruning is usually carried out in a three stages procedure: (a) a set of sub-trees of the initial tree is generated; (b) some reliable error estimation procedure is used to obtain estimates for each member of this set; and (c) some method based on these estimates is used to select one of these trees as the final tree model. Different methods exist for each of these steps. A common setup (e.g., Breiman et al., 1984) is to use error-complexity pruning to generate a sequence of nested sub-trees, whose error is then estimated by cross validation. The final tree is selected using the x-SE rule, which starts with the lowest estimated error sub-tree and then selects the smallest tree within the interval of x standard errors of the lowest estimated error tree (a frequent setting is to use one standard error).

Variations on the subject of pruning regression trees include, among others: pre-pruning alternatives (e.g., Breiman & Meisel, 1976; Friedman, 1979); the use of different tree error estimators (see Torgo (1998) for a comparative study and references to different alternatives); and the use of the Minimum Description Length (MDL) principle to guide the pruning (Robnik-Sikonja & Kononenko, 1998).

Cross References

►Model Trees
►Out-of-the-Sample
►Random Forests
►Regression
►Supervised Learning
►Training Sample

Recommended Reading

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees. Statistics/probability series*. Wadsworth & Brooks/Cole Advanced Books & Software.

Breiman, L., & Meisel, W. S. (1976). General estimates of the intrinsic variability of data in nonlinear regression models. *Journal of the American Statistical Association, 71*, 301–307.

Buja, A., & Lee, Y.-S. (2001). Data mining criteria for tree-based regression and classification. In *Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 27–36). San Francisco, California, USA.

Friedman, J. H. (1979). A tree-structured approach to nonparametric multiple regression. In T. Gasser & M. Rosenblatt (Eds.), *Smoothing techniques for curve estimation. Lecture notes in mathematics* (Vol. 757, pp. 5–22). Berlin: Springer.

Gama, J. (2004). Functional trees. *Machine Learning, 55*(3), 219–250.

Li, K. C., Lue, H., & Chen, C. (2000). Interactive tree-structured regression via principal Hessians direction. *Journal of the American Statistical Association, 95*, 547–560.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12*, 361–386.

Lubinsky, D. (1995). Tree structured interpretable regression. In *Proceedings of the workshop on AI & statistics*.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(5), 612–625.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of American Statistical Association, 58*(302), 415–434.

Robnik-Sikonja, M., & Kononenko, I. (1996). Context-sensitive attribute estimation in regression. In *Proceedings of the ICML-96 workshop on learning in context-sensitive domains*. Brighton, UK.

Robnik-Sikonja, M., & Kononenko, I. (1998). Pruning regression trees with MDL. In *Proceedings of ECAI-98*. Brighton, UK.

Torgo, L. (1998). Error estimates for pruning regression trees. In C. Nedellec & C. Rouveirol (Eds.), *Proceedings of the tenth European conference on machine learning. LNAI* (Vol. 1398). London, UK: Springer-Verlag.

Torgo, L. (1999). *Inductive learning of tree-based regression models*. PhD thesis, Department of Computer Science, Faculty of Sciences, University of Porto.

Torgo, L., & Ribeiro, R. (2003). Predicting outliers. In N. Lavrac, D. Gamberger, L. Todorovski, & H. Blockeel (Eds.), *Proceedings of principles of data mining and knowledge discovery (PKDD'03). LNAI* (Vol. 2838, pp. 447–458). Berlin/Heidelberg: Springer-Verlag.

Regularization

Xinhua Zhang
Australian National University, NICTA London Circuit
Canberra, Australia

Definition

Regularization plays a key role in many machine learning algorithms. Exactly fitting a model to the training data is generally undesirable, because it will fit the noise in the training examples (►overfitting), and is doomed to predict (generalize) poorly on unseen data. In contrast, a simple model that fits the training data well is more likely to capture the regularities in it and generalize well. So a regularizer is introduced to quantify the complexity of a model, and many successful machine learning algorithms fall in the framework of regularized risk minimization:

$$\text{(How well the model fits the training data)} \tag{1}$$
$$+\lambda \cdot \text{(complexity/regularization of the model)}, \tag{2}$$

where the positive real number λ controls the tradeoff.

There is a variety of regularizers, which yield different statistical and computational properties. In general, there is no universally best regularizer, and a regularization approach must be chosen depending on the dataset.

Motivation and Background

The main goal of machine learning is to induce a model from the observed data, and use this model to make predictions and decisions. This is also largely the goal of general natural science, and is commonly called inverse problems ("forward problem" means using the model to generate observations). Therefore, it is no surprise that regularization had been well studied before the emergence of machine learning.

Inverse problems are typically ill-posed, e.g., having only a finite number of samples drawn from an uncountable space, or having a finite number of measurements in an infinite dimensional space. In machine learning, we often need to induce a classifier for the whole feature-label space, while only a finite number of feature-label pairs are available for training. In practice, the set of candidate models is often flexible enough to precisely fit all the training examples. However, this can lead to significant overfitting when the training data is noisy, and the real challenge is how to generalize well on the unseen data in the whole feature-label space.

Many techniques have been proposed to tackle ill-posed inverse problems. Almost all of them introduce an additional measure on how much a model is preferred *a priori* (i.e., without observing the training data). This extra belief on the desirable form of the model reflects the external knowledge of the model designer. It cannot be replaced by the data itself according to the "no free lunch theorem," which states that if there is no assumption on the mechanism of labeling, then it is impossible to generalize and any model can be inferior to another on some distribution of the feature-label pair (Devroye, Gyor, & Lugosi, 1996).

A commonly used prior is the so-called ►Occam's razor, which prefers "simple" models. It asserts that among all the models which fit the training data well, the simplest one is more likely to capture the "regularities" in it and hence has a larger chance to generalize well to the unseen data. Then an immediate question is how to quantify the complexity of a model, which is often called a regularizer. Intuitively, a regularizer can encode preference for a sparse model (few features are relevant for prediction), a large margin model (two classes have a wide margin), or a smooth model with weak high-frequency components. A general framework of regularization was given by Tikhonov (1943).

Theory

Suppose n feature-label pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are drawn *iid* from a certain joint distribution P on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the spaces of feature and label respectively. Let the marginal distribution on $\mathcal{X}$ and $\mathcal{Y}$ be P_x and P_y respectively. For convenience, let $\mathcal{X}$ be $\mathbb{R}^p$ (Euclidean space). Denote $X := (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathbf{y} := (y_1, \ldots, y_n)^\top$.

An Illustrative Example: Ridge Regression

Ridge regression is illustrative of the use of regularization. It tries to fit the label y by a linear model $\langle \mathbf{w}, \mathbf{x} \rangle$ (inner product). So we need to solve a system of linear equations in $\mathbf{w}$: $(\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \mathbf{w} = \mathbf{y}$, which is equivalent to a linear least square problem: $\min_{\mathbf{w} \in \mathbb{R}^p} \|X^\top \mathbf{w} - \mathbf{y}\|^2$. If the rank of X is less than the dimension of $\mathbf{w}$, then it is overdetermined and the solution is not unique.

To approach this ill-posed problem, one needs to introduce additional assumptions on what models are preferred, i.e., the regularizer. One choice is to pick a matrix Γ and regularize $\mathbf{w}$ by $\|\Gamma \mathbf{w}\|^2$. As a result, we solve $\min_{\mathbf{w} \in \mathbb{R}^p} \|X^\top \mathbf{w} - \mathbf{y}\|^2 + \lambda \|\Gamma^\top \mathbf{w}\|^2$, and the solution has closed form $\mathbf{w}^* = (XX^\top + \lambda \Gamma \Gamma^\top) X\mathbf{y}$. Γ can be simply the identity matrix which encodes our preference for small norm models.

The use of regularization can also be justified from a Bayesian point of view. Treating $\mathbf{w}$ as a multi-variate random variable and the likelihood as $\exp\left(-\|X^\top \mathbf{w} - \mathbf{y}\|^2\right)$, then the minimizer of $\|X^\top \mathbf{w} - \mathbf{y}\|^2$

is just a maximum likelihood estimate of **w**. However, we may also assume a prior distribution over **w**, e.g., a Gaussian prior $p(\mathbf{w}) \sim \exp\left(-\lambda \left\|\Gamma^{\top}\mathbf{w}\right\|^2\right)$. Then the solution of the ridge regression is simply the maximum a posterior estimate of **w**.

Examples of Regularization

A common approach to regularization is to penalize a model by its complexity measured by some real-valued function, e.g., a certain "norm" of **w**. We list some examples below:

***L_1 regularization*:** L_1 regularizer, $\|\mathbf{w}\|_1 := \sum_i |w_i|$, is a popular approach to finding sparse models, i.e., only a few components of **w** are nonzero and only the corresponding small number of features are relevant to the prediction. A well-known example is the LASSO algorithm (Tibshirani, 1996), which uses a L_1 regularized least square:

$$\min_{\mathbf{w}\in\mathbb{R}^p} \left\|X^{\top}\mathbf{w} - \mathbf{y}\right\|^2 + \lambda \|\mathbf{w}\|_1 .$$

***L_2 regularization*:** The L_2 regularizer, $\frac{1}{2}\|w\|_2^2 := \frac{1}{2}\sum_i w_i^2$, is popular due to its self-dual properties. In all L_p spaces, only the L_2 space is Hilbertian and self-adjoint, so it affords much convenience in studying and exploiting the dual properties of the L_2 regularized models. A well-known example is the support vector machines (SVMs), which minimize the L_2 regularized hinge loss:

$$\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - \langle \mathbf{w}, \mathbf{x}_i\rangle\} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 .$$

***L_p regularization*:** In general, all L_p norms $\|\mathbf{w}\|_p := \left(\sum_i |w_i|^p\right)^{1/p}$ $(p \geq 1)$ can be used for regularization. When $p < 1$, $\left(\sum_i |w_i|^p\right)^{1/p}$ is no longer convex. A specially interesting case is when $p = 0$, and $\|\mathbf{w}\|_0$ is defined as the number of nonzero elements in **w** (the sparseness of **w**). But explicitly optimizing the L_0 norm leads to a combinatorial problem which is hard to solve. In some cases, the L_1 regularizer can approximately recover the solution of L_0 regularization (Candes & Tao, 2005).

***$L_{p,q}$ regularizer*:** The $L_{p,q}$ regularizer is popular in the context of multi-task learning (Tropp, 2006). Suppose there are T tasks, and each training example $\mathbf{x}_i$ has a label vector $\mathbf{y}_i \in \mathbb{R}^T$ with each component corresponding to a task. For each task t, we seek for a linear regressor $\langle \mathbf{w}_t, \mathbf{x}\rangle$ such that for each training example $\mathbf{x}_i$, $\langle \mathbf{w}_t, \mathbf{x}_i\rangle$ fits the tth component of $\mathbf{y}_i$. Of course, the $\mathbf{w}_t$ could be determined independently from each other. But in many applications, the T tasks are somehow related, and it will be advantageous to learn them as a whole. Stack $\mathbf{w}_t$'s into a matrix $W := (\mathbf{w}_1, \ldots, \mathbf{w}_T)$ where each column corresponds to a task and each row corresponds to a feature. Then the intuition of multitask learning can be concretized by regularizing W with the $L_{p,q}$ compositional norm $(p, q \geq 1)$:

$$\|W\|_{p,q} := \left(\sum_i \left(\sum_t |w_{it}|^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}},$$

where w_{it} is the ith component of $\mathbf{w}_t$. When $q = 1$, it becomes the L_1 norm of the L_p norm of the rows, and the sparse inducing property of L_1 norm encourages the rows to have L_p norm 0, i.e., the corresponding feature is not used by *any* task. Other choices of p and q are also possible.

***Entropy regularizer*:** The entropy regularizer is useful in boosting, and it works in a slightly different way from the above regularizers. Boosting aims to find a convex combination of hypotheses, such that the training data is accurately classified by the ensemble. At each step, the boosting algorithm maintains a distribution $\mathbf{d}$ $(d_i > 0$ and $\sum_i d_i = 1)$ over the training examples, feeds $\mathbf{d}$ to an oracle which returns a new hypothesis, and then updates $\mathbf{d}$ and go on. As a "simple" ensemble means a small number of weak hypotheses, the boosting algorithm is expected to find an accurate ensemble by taking as few steps as possible. This can be achieved by exponentiated gradient descent (Kivinen & Warmuth, 1997), which stems from the relative entropy regularizer $\sum_i d_i \log \frac{d_i}{1/n}$ applied at each step. It also attracts $\mathbf{d}$ toward the uniform distribution, which helps avoid overfitting the noise, i.e., trying hard to match the (incorrect) label of a few training examples.

***Miscellaneous*:** Instead of using a function that directly measures the complexity of the model **w**, regularization can also be achieved by penalizing the complexity of the *output* of the model on the training data. This is called value regularization (Rifkin & Lippert, 2007). It not only yields neat derivations of standard algorithms, but also provides much convenience in studying the learning theory and optimization.

Furthermore, the regularized risk minimization framework in (1) is not the only approach to regularization. For example, in ▶online learning where the model is updated iteratively, early stopping is an effective form of regularization and it has been widely used in training neural networks. Suppose the available dataset is divided into a training set and a validation set, and the model is learned online from the training set. Then the algorithm terminates when the performance of the model on the validation set stops improving.

Measuring the Capacity of Model Class

Besides penalizing the complexity of the model, we can restrict the complexity of the model class $\mathcal{F}$ in the first place. For example, ▶linear regression is intrinsically "simpler" than quadratic regression. ▶Decision stumps are "simpler" than linear classifiers. In other words, regularization can be achieved by restricting the capacity of the model class, and the key question is how to quantify this capacity. Some commonly used measures in the context of binary classification are the following.

***VC dimension*:** The Vapnik–Chervonenkis dimension (▶VC dimension) quantifies how many data points can be arbitrarily labeled by using the functions in $\mathcal{F}$ (Vapnik & Chervonenkis, 1971). $\mathcal{F}$ is said to *shatter* a set of data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ if for any assignment of labels to these points, there exists a function $f \in \mathcal{F}$ which yields this labeling. The VC dimension of $\mathcal{F}$ is the maximum n such that there exist n data points that can be shattered by $\mathcal{F}$. For example, decision stumps have VC dimension 2, and linear classifiers (with bias) in a p-dimensional space have VC dimension $p+1$.

***Covering number*:** The idea of covering number (Guo, Bartlett, Shawe-Taylor & Williamson 1999) is to characterize the inherent "dimension" of $\mathcal{F}$, in a way that follows the standard concept of vector dimension. Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we may endow the model class $\mathcal{F}$ with the following metric:

$$d_n(f,g) := \frac{1}{n}\sum_{i=1}^{n} \delta(f(\mathbf{x}_i) \neq g(\mathbf{x}_i)), \quad \forall f, g \in \mathcal{F},$$

where $\delta(\cdot) = 1$ if $\cdot$ is true and 0 otherwise. A set of functions $f_1, \ldots, f_m$ is said to be a cover of $\mathcal{F}$ at radius ϵ if for any function $f \in \mathcal{F}$, there exists an f_i such that $d_n(f, f_i) < \epsilon$. Then the covering number of $\mathcal{F}$ at radius $\epsilon > 0$ with respect to d_n is the minimum size of a cover of radius ϵ.

To understand the motivation of the definition, consider the unit ball in $\mathbb{R}^p$. To cover it by ϵ radius balls, one needs order $N(\epsilon, p) = \epsilon^{-p}$ balls. Then the dimension p can be estimated from the rate of growth of $\log N(\epsilon, p) = -p \log \epsilon$ with respect to ϵ. The covering number is an analogy of $N(\epsilon, p)$, and the dimension of $\mathcal{F}$ can be estimated in the same spirit.

***Rademacher average*:** The Rademacher average is a soft variant of the VC dimension. Instead of requiring the model class to shatter n data points, it allows that the labels be violated at some cost. Let $\sigma_i \in \{-1, 1\}$ be an arbitrary assignment of the labels, and assume all functions in $\mathcal{F}$ range in $\{-1, 1\}$ (this restriction can be relaxed). Then a model $f \in \mathcal{F}$ is considered as the most consistent with $\{\sigma_i\}$ if it maximizes $\frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)$. This term equals 1 if $\mathcal{F}$ does contain a model consistent with $\{\sigma_i\}$. Then we take an average over all possible assignments of σ_i, i.e., treating σ_i as a binary random variable with $P(\sigma_i = 1) = P(\sigma_i = -1) = 0.5$, and calculating the expectation of the best compatibility over $\{\sigma_i\}$:

$$\mathcal{R}_n(\mathcal{F}) = \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Rademacher complexity often leads to tighter generalization bounds than VC dimension thanks to its dependency on the observed data. Furthermore, we may take expectation over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ drawn independently and identically from P_x:

$$\mathcal{R}(\mathcal{F}) = \mathop{\mathbb{E}}_{\mathbf{x}_i \sim P_x} \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)\right].$$

Therefore, similar to VC dimension, the Rademacher average is high if the model class $\mathcal{F}$ is "rich," and can match most assignments of $\{\sigma_i\}$.

Applications

In many applications such as bioinformatics, the training examples are expensive and the number of features p is much higher than the number of labeled examples n. In such cases, regularization is crucial, (e.g., Zhang, Zhang, & Wells, 2008).

L_1 regularization has gained much popularity recently in the field of compressed sensing, and it has been widely used in imaging for radar, astronomy, medical diagnosis, and geophysics. See an ensemble of publications at http://dsp.rice.edu/cs

The main spirit of regularization, namely a preference for models with lower complexity, has been used by some ►model evaluation techniques. Examples include Akaike information criterion (AIC), Bayesian information criterion (BIC), ►minimum description length (MDL), and the ►minimum message length (MML).

Cross References

►Minimum Description Length
►Model Evaluation
►Occam's Razor
►Overfitting
►Statistical Learning Theory
►Support Vector Machines
►VC Dimension

Recommended Reading

Regularization lies at the heart of statistical machine learning, and it is indispensable in almost every learning algorithm. A comprehensive statistical analysis from the computational learning theory perspective can be found in Bousquet, Boucheron, & Lugosi (2005) and Vapnik (1998). Abundant resources on compressed sensing including both theory and applications are available at http://dsp.rice.edu/cs. Regularizations related to SVMs and kernel methods are discussed in detail by Schölkopf & Smola (2002) and Shawe-Taylor & Cristianini (2004). Anthony & Bartlett (1999) provide in-depth theoretical analysis for neural networks.

Anthony, M., & Bartlett, P. L. (1999). *Neural network learning: Theoretical foundations*. Cambridge: Cambridge University Press.

Bousquet, O., Boucheron, S., & Lugosi, G. (2005). Theory of classification: A survey of recent advances. *ESAIM: Probability and Statistics, 9*, 323–375.

Candes, E., & Tao, T. (2005). Decoding by linear programming. *IEEE Transactions on Information Theory, 51*(12), 4203–4215.

Devroye, L., Györ, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition, vol. 31 of applications of mathematics*. New York: Springer.

Guo, Y., Bartlett, P. L., Shawe-Taylor, J., & Williamson, R. C. (1999). Covering numbers for support vector machines. In *Proceedings of the Annual Conference Computational Learning Theory*.

Kivinen, J., & Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation, 132*(1), 1–64.

Rifkin, R. M., & Lippert, R. A. (2007). Value regularization and Fenchel duality. *Journal of Machine Learning Research, 8*, 441–479.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). Kernel methods for pattern analysis. Cambridge: Cambridge University Press.

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

Tikhonov, A. N. (1943). On the stability of inverse problems. *Doklady Akademii nauk SSSR, 39*(5), 195–198.

Tropp, J. A. (2006). Algorithms for simultaneous sparse approximation, part ii: Convex relaxation. *Signal Processing, 86*(3), 589C–602.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley: New York

Vapnik, V., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications, 16*(2), 264–281.

Zhang, M., Zhang, D., & Wells, M. T. (2008). Variable selection for large p small n regression models with incomplete data: Mapping QTL with epistases. *BMC Bioinformatics, 9*, 251.

Regularization Networks

►Radial Basis Function Networks

Reinforcement Learning

Peter Stone
The University of Texas at Austin, Austin, TX, USA

Reinforcement learning describes a large class of learning problems characteristic of autonomous agents interacting in an environment: sequential decision-making problems with delayed reward. Reinforcement learning algorithms seek to learn a policy (mapping from states to actions) that maximize the reward received over time.

Unlike in ►supervised learning problems, in reinforcement-learning problems, there are no labeled examples of correct and incorrect behavior. However, unlike ►unsupervised learning problems, a reward signal can be perceived.

Many different algorithms for solving reinforcement-learning problems are covered in other entries. This entry provides just a brief high-level classification of the algorithms.

Perhaps the most well-known approach to solving reinforcement-learning problems, as covered in detail by Sutton and Barto (1998), is based on learning a value function, which represents the long-term expected reward of each state the agent may encounter, given a particular policy. This approach typically assumes that the environment is a ▶Markov decision process in which rewards are discounted over time, though it is also possible to optimize for average reward per time step as in ▶average-reward reinforcement learning. If a complete model of the environment is available, ▶dynamic programming, or specifically ▶value iteration, can be used to compute an optimal value function, from which an optimal policy can be derived.

If a model is not available, an optimal value function can be learned from experience via model-free techniques such as ▶temporal difference learning, which combine elements of dynamic programming with Monte Carlo estimation. Partly due to Watkins' elegant proof that ▶Q-learning converges to the optimal value function (Watkins, 1989), temporal difference methods are currently among the most commonly used approaches for reinforcement-learning problems.

Watkins' convergence proof relies on executing a policy that visits every state infinitely often. In practice, Q-learning does converge in small, discrete domains. However in larger, and particularly in continuous domains, the learning algorithm must generalize the value function across states, a process known as ▶value function approximation. Examples include ▶instance-based reinforcement learning, ▶Gaussian process reinforcement learning, and ▶relational reinforcement learning.

Even when combined with value function approximation, the most basic value-free methods, such as Q-learning and SARSA are very inefficient with respect to experience: they are not sample-efficient. With the view that experience is often more costly than computation, much research has been devoted to making more efficient use of experience, for instance via ▶hierarchical reinforcement learning, ▶reward shaping, or ▶model-based reinforcement learning in which the experience is used to learn a domain model, which can then be solved via dynamic programming.

Though these methods make efficient use of the experience that is presented to them, the goal of optimizing sample efficiency also motivates the study of ▶efficient exploration in reinforcement learning. The study of exploration methods can be isolated from the full reinforcement-learning problem by removing the notion of temporally delayed reward as is done in ▶associative reinforcement learning or by removing the notion of states altogether as is done in ▶k-armed bandits. k-Armed bandit algorithms focus entirely on the exploration versus exploitation challenge, without having to worry about generalization across states or delayed rewards. Back in the context of the full RL problem, ▶Bayesian reinforcement learning enables optimal exploration given prior distributions over the parameters of the learning problem. However, its computational complexity has limited its use so far to very small domains.

Although most of the methods above revolve around learning a value function, reinforcement-learning problems can also be solved without learning value functions, by directly searching the space of potential policies via policy search. Effective ways of conducting such a search include ▶policy gradient reinforcement learning, ▶least squares reinforcement learning methods, and evolutionary reinforcement learning.

As typically formulated, the goal of a reinforcement-learning algorithm is to learn an optimal (or high-performing) policy based on knowledge of, or experience of, a reward function (and state transition function). However, it is also possible to take the opposite perspective that of trying to learn the reward function based on observation of the optimal policy. This problem formulation is known as ▶inverse reinforcement learning.

Leveraging this large body of theory and algorithms, a current focus in the field is deploying large-scale, successful applications of reinforcement learning. Two such applications treated herein are ▶autonomous helicopter flight using reinforcement learning and ▶robot learning.

Cross References

▶Associative Reinforcement Learning

▶Autonomous Helicopter Flight Using Reinforcement Learning

►Average-Reward Reinforcement Learning
►Bayesian Reinforcement Learning
►Dynamic Programming
►Efficient Exploration in Reinforcement Learning
►Gaussian Process Reinforcement Learning
►Hierarchical Reinforcement Learning
►Instance-Based Reinforcement Learning
►Inverse Reinforcement Learning
►Least Squares Reinforcement Learning Methods
►Model-Based Reinforcement Learning
►Policy Gradient Methods
►Q-Learning
►Relational Reinforcement Learning
►Reward Shaping
►Symbolic Dynamic Programming
►Temporal Difference Learning
►Value Function Approximation

Recommended Reading

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT.
Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK.

Reinforcement Learning in Structured Domains

►Relational Reinforcement Learning

Relational

The adjective *relational* can have two different meanings in machine learning. The ambiguity comes from an ambiguity in database terminology.

►Relational Data Mining refers to relational database, and is sometimes denoted *multi-relational* data mining. Indeed a relational database typically involves several relations (a relation is the formal name of a table). Those tables are often linked to each other, but the "relational" adjective does not refer to those relationships.

On the other hand, ►Relational Learning focuses on those relationships and intends to learn whether a relationship exists between some given entities.

Cross References

►Propositionalization
►Relational Data Mining
►Relational Learning

Relational Data Mining

►Inductive Logic Programming

Relational Dynamic Programming

►Symbolic Dynamic Programming

Relational Learning

Jan Struyf, Hendrik Blockeel
Katholieke Universiteit Leuven, Heverlee, Belgium

Problem Definition

Relational learning refers to learning in a context where there may be relationships between learning examples, or where these examples may have a complex internal structure (i.e., consist of multiple components and there may be relationships between these components). In other words, the "relational" may refer to both an internal or external relational structure describing the examples. In fact, there is no essential difference between these two cases, as it depends on the definition of an example whether relations are internal or external to it. Most methods, however, are clearly set in one of these two contexts.

Learning from Examples with External Relationships

This setting considers learning from a set of examples where each example itself has a relatively simple description, for instance in the attribute-value format, and relationships may be present among these examples.

Example 1. Consider the task of web-page classification. Each web-page is described by a fixed set of attributes, such as a bag of words representation of the page. Web-pages may be related through hyperlinks, and the class label of a given page typically depends on the labels of the pages to which it links.

Example 2. Consider the Internet Movie Database (www.imdb.com). Each movie is described by a fixed set of attributes, such as its title and genre. Movies are related to other entity types, such as *Studio*, *Director*, *Producer*, and *Actor*, each of which is in turn described by a different set of attributes. Note that two movies can be related through the other entity types. For example, they can be made by the same studio or star the same well-known actor. The learning task in this domain could be, for instance, predicting the opening weekend box office receipts of the movies.

If relationships are present among examples, then the examples may not be independent and identically distributed (i.i.d.), an assumption made by many learning algorithms. Relational data that violates this assumption can be detrimental to learning performance as Jensen and Neville (2002) show. Relationships among examples can, on the other hand, also be exploited by the learning algorithm. ►Collective classification techniques (Jensen, Neville, & Gallagher, 2004), for example, take the class labels of related examples into account when classifying a new instance.

Learning from Examples with a Complex Internal Structure

In this setting, each example may have a complex internal structure, but no relationships exist that relate different examples to one another. Learning algorithms typically use individual-centered representations in this setting, such as logical interpretations or strongly typed terms (Lloyd, 2003), which store together all data of a given example. An important advantage of individual-centered representations is that they scale better to large datasets. Special cases of this setting include applications where the examples can be represented as graphs, trees, or sequences.

Example 3. Consider a database of candidate chemical compounds to be used in drugs. The molecular structure of each compound can be represented as a graph where the vertices are atoms and the edges are bonds. Each atom is labeled with its element type and the bonds can be single, double, triple, or aromatic bonds. Compounds are classified as active or inactive with regard to a given disease and the goal is to build models that are able to distinguish active from inactive compounds based on their molecular structure. Such models can, for instance, be used to gain insight in the common substructures, such as binding sites, that determine a compound's activity.

Approaches to Relational Learning

Many different kinds of learning tasks have been defined in relational learning, and an even larger number of approaches have been proposed for tackling these tasks. We give an overview of different learning settings that can be considered instances of relational learning.

Inductive Logic Programming

In ►inductive logic programming (ILP), the input and output knowledge of a learner are described in (variants of) first-order predicate logic. Languages based on first-order logic are highly expressive from the point of view of knowledge representation, and indeed, a language such as Prolog (Bratko, 2000) can be used without adaptations to represent objects and the relationships between them, as well as background knowledge that one may have about the domain.

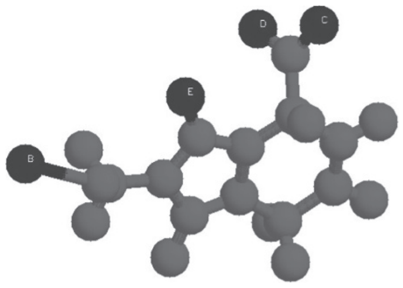
Example 4. This example is based on the work by Finn, Muggleton, Page, and Srinivasan (1998). Consider a data set that describes chemical compounds. The active compounds in the set are ACE inhibitors, which are used in treatments for hypertension. The molecular structure of the compounds is represented as a set of Prolog facts, such as: atom(m1, a1, o).

```
atom(m1, a2, c).
...
bond(m1, a1, a2, 1).
...
coord(m1, a1, 5.91, −2.44, 1.79).
coord(m1, a2, 0.57, −2.77, 0.33).
...
```

which states that molecule m1 includes an oxygen atom a1 and a carbon atom a2 that are single bonded. The coord/5 predicate lists the 3D coordinates of the

```
ACE_inhibitor(A) :-
     zincsite(A, B),
     hacc(A, C),
     dist(A, B, C, 7.9, 1.0),
     hacc(A, D),
     dist(A, B, D, 8.5, 1.0),
     dist(A, C, D, 2.1, 1.0),
     hacc(A, E),
     dist(A, B, E, 4.9, 1.0),
     dist(A, C, E, 3.1, 1.0),
     dist(A, D, E, 3.8, 1.0).
```

a



c

Molecule A is an ACE inhibitor if:
molecule A can bind to zinc at site B, and
molecule A contains a hydrogen acceptor C, and
the distance between B and C is 7.9 ± 1.0Å, and
molecule A contains a hydrogen acceptor D, and
the distance between B and D is 8.5 ± 1.0Å, and
the distance between C and D is 2.1 ± 1.0Å, and
molecule A contains a hydrogen acceptor E, and
the distance between B and E is 4.9 ± 1.0Å, and
the distance between C and E is 3.1 ± 1.0Å, and
the distance between D and E is 3.8 ± 1.0Å.

b

Relational Learning. Figure 1. (a) Prolog clause modeling the concept of an ACE inhibitor in terms of the background knowledge predicates zincsite/2, hacc/2, and dist/5. (b) The inductive logic programming system Progol automatically translates (a) into the "Sternberg English" rule, which can be easily read by human experts. (c) A molecule with the active site indicated by the atoms B, C, D, and E. (Image courtesy of Finn et al. (1998).)

atoms in the given conformer. Background knowledge, such as the concepts zinc site, hydrogen donor, and the distance between atoms, are defined by means of Prolog clauses. Figure 1 shows a clause learned by the inductive logic programming system PROGOL (Džeroski & Lavrač, 2001, Ch. 7) that makes use of these background knowledge predicates. This clause is the description of a pharmacophore, that is, a submolecular structure that causes a certain observable property of a molecule.

More details on the theory of inductive logic programming and descriptions of algorithms can be found in the entry on ▶inductive logic programming in this encyclopedia, or in references (De Raedt, 2008; Džeroski & Lavrač, 2001).

Learning from Graphs

A graph is a mathematical structure consisting of a set of nodes V and a set of edges $E \subseteq V^2$ between those nodes. The set of edges is by definition a binary relation defined over the nodes. Hence, for any learning problem where the relationships between examples can be described using a single binary relation, the training set can be represented straightforwardly as a graph. This setting covers a wide range of relational learning tasks, for example, web mining (the set of links between pages is a binary relation), social network analysis, etc. Non-binary relationships can be represented as hypergraphs; in a hypergraph, edges are defined as subsets of V of arbitrary size, rather than elements of V^2.

In graph-based learning systems, there is a clear distinction between approaches that learn from examples with external relationships, where the whole data set is represented as a single graph and each node is an example, and individual-centered approaches, where each example by itself is a graph. In the first kind of approaches, the goal is often to predict properties of existing nodes or edges, to predict the existence or non-existence of edges ("▶link discovery"), to predict whether two nodes actually refer to the same object ("node identification"), detection of subgraphs that frequently occur in the graph, etc. When learning from multiple graphs, a typical goal is to learn a model for classifying the graphs, to find frequent substructures (where frequency is defined as the number of graphs a subgraphs occurs in), etc.

Compared to other methods for relational learning, graph-based methods typically focus more on the structure of the graph, and less on properties of single nodes. They may take node and edge labels into account, but typically do not allow for more elaborate information to be associated with each node.

▶Graph mining methods are often more efficient than other relational mining methods because they avoid certain kinds of overhead, but are typically still NP-complete, as they generally rely on subgraph isomorphism testing. Nevertheless, researchers have been able to significantly improve efficiency or even avoid NP-completeness by looking only for linear or tree-shaped patterns, or by restricting the graphs analyzed to a relatively broad subclass. As an example, Horváth et al. (2006) show that a large majority of molecules belong to the class of outerplanar graphs, and propose an efficient algorithm for subgraph isomorphism testing in this class.

More information about mining graph data can be found in the ▶graph mining entry in this encyclopedia, or in (Cook & Holder, 2007; Washio & Motoda, 2003).

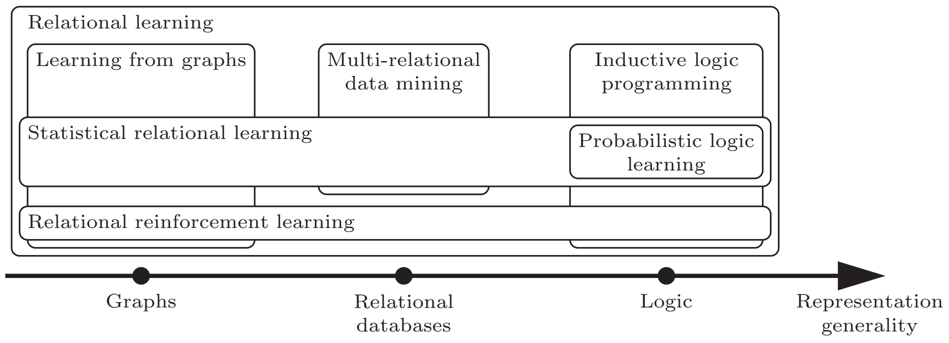
Multi-relational Data Mining

Multi-relational data mining approaches relational learning from the relational database point of view. The term "multi-relational" refers to the fact that from the database perspective, one learns from information spread over multiple tables or relations, as opposed to ▶attribute-value learning, where one learns from a single table.

Multi-relational data mining systems tightly integrate with relational databases. Mainly rule and decision tree learners have been developed in this setting. Because practical relational databases may be huge, most of these systems pay much attention to efficiency and scalability, and use techniques such as sampling and pre-computation (e.g., materializing views). An example of a scalable and efficient multi-relational rule learning system is CrossMine (Yin, Han, Yang, & Yu, 2006).

An alternative approach to relational learning and multi-relational data mining is ▶propositionalization. Propositionalization consists of automatically converting the relational representation into an attribute-value representation and then using attribute-value data mining algorithms on the resulting representation. An important line of research within multi-relational data mining investigates how database approaches can be used to this end. Database oriented propositionalization creates a view in which each example is represented by precisely one row. Information from related entities is incorporated into this row by adding derived attributes, computed by means of aggregation. In the movie database (Example 2), the view representing movies could include aggregated attributes such as the number of actors starring in the movie. A comparison of propositionalization approaches is presented by Krogel et al. (2003), and a discussion of them is also included in this volume.

Finally, note that most inductive logic programming systems are directly applicable to multi-relational data mining by representing each relational table as a predicate. This is possible because the relational representation is essentially a subset of first-order logic (known as datalog). Much research on multi-relational


Relational learning
Learning from graphs
Multi-relational data mining
Inductive logic programming
Statistical relational learning
Probabilistic logic learning
Relational reinforcement learning
Graphs
Relational databases
Logic
Representation generality

data mining was developed within the ILP community (Džeroski & Lavrač, 2001).

Statistical Relational Learning/Probabilistic Logic Learning

Research on relational learning, especially in the beginning, has largely focused on how to handle the relational structure of the data, and ignored aspects such as uncertainty. Indeed, the databases handled in multi-relational data mining, or the knowledge assumed given in inductive logic programming, are typically assumed to be deterministic. With the rise of probabilistic representations and algorithms within machine learning has come an increased interest in enabling relational learners to cope with uncertainty in the input data. This goal has been approached from at least two different directions: statistical learning approaches have been extended toward the relational setting, giving rise to the area of ▶statistical relational learning, whereas inductive logic programming researchers have investigated how to extend their knowledge representation and learning algorithms to cater for probabilistic information, referring to this research area as ▶probabilistic logic learning. While there are some differences in terminology and approaches, both research areas essentially address the same research question, namely how to integrate relational and probabilistic learning.

Among the best known approaches for statistical relational learning is the learning of probabilistic relational models (PRMs, Džeroski & Lavrač, 2001, Chap. 13). PRMs extend Bayesian networks to the relational representation used in relational databases. PRMs model the joint probability distribution over the non-key attributes in a relational database schema. Similar to Bayesian networks, PRMs are ▶graphical models. Each attribute corresponds to a node and direct dependencies are modeled by directed edges. Such edges can connect attributes from different entity types that are (indirectly) related (such a relationship is called a "slot chain"). Inference in PRMs occurs by constructing a ▶Bayesian network by instantiating the PRM with the data in the database and performing the inference in the latter. To handle 1:N relationships in the Bayesian network, PRMs make use of aggregation, similar to the propositionalization techniques mentioned above.

Bayesian logic programs (BLPs) (Kersting, 2006) aim at combining the inference power of Bayesian networks with that of first-order logic reasoning. Similar to PRMs, the semantics of a BLP is defined by translating it to a Bayesian network. Using this network, the probability of a given interpretation or the probability that a given query yields a particular answer can be computed.

The acyclicity requirement of Bayesian networks carries over to representations such as PRMs and BLPs. Markov logic networks (MLNs) (Richardson & Domingos, 2006) upgrade ▶Markov networks to first-order logic and allow networks with cycles. MLNs are defined as sets of weighted first-order logic formulas. These are viewed as "soft" constraints on logical interpretations: the fewer formulas a given interpretation violates, the higher its probability. The weight determines the contribution of a given formula: the higher its weight, the greater the difference in log probability between an interpretation that satisfies the formula and one that does not, other things being equal. The Alchemy system implements structure and parameter learning for MLNs.

More specific statistical learning techniques such as Naïve Bayes and Hidden Markov Models have also been upgraded to the relational setting. More information about such algorithms and about statistical relational learning in general can be found in (Getoor & Taskar, 2007; Kersting, 2006).

In probabilistic logic learning, two types of semantics are distinguished (De Raedt & Kersting, 2003): the model theoretic semantics and the proof theoretic semantics. Approaches that are based on the model theoretic semantics define a probability distribution over interpretations and extend probabilistic attribute-value techniques, such as Bayesian networks and Markov networks, while proof theoretic semantics approaches define a probability distribution over proofs and upgrade, e.g., stochastic context free grammars.

Example 5. Consider the case where each example is a sentence in natural language. In this example, a model theoretic approach would define a probability distribution directly over sentences. A proof theoretic approach would define a probability distribution over "proofs," in this case possible parse trees of the sentence (each sentence may have several possible parses). Note that the proof theoretic view is more general in the sense that the distribution over sentences can be computed from the distribution over proofs.

Stochastic logic programs (SLPs) (Muggleton, 1996) follow most closely the proof theoretic view and upgrade stochastic context free grammars to first-order logic. SLPs are logic programs with probabilities attached to the clauses such that the probabilities of clauses with the same head sum to 1.0. The probability of a proof is then computed as the product of the probabilities of the clauses that are used in the proof. PRISM (Sato & Kameya, 1997) follows a related approach where the probabilities are defined on ground facts.

Like with standard graphical models, learning algorithms may include both parameter learning (estimating the probabilities) and structure learning (learning the program). For most frameworks mentioned above, such techniques have been or are being developed.

For a more detailed treatment of statistical relational learning and probabilistic logic learning, we refer to the entry on statistical relational learning in this volume, and to several reference works (De Raedt & Kersting, 2003; Getoor & Taskar, 2007; Kersting, 2006; De Raedt, Frasconi, Kersting, & Muggleton, 2008).

Relational Reinforcement Learning

Relational reinforcement learning (RRL) (Džeroski, De Raedt, & Driessens, 2001; Tadepalli, Givan, & Driessens, 2004) is reinforcement learning upgraded to the relational setting. Reinforcement learning is concerned with how an agent should act in a given environment to maximize its accumulated reward. In RRL, both the state of the environment and the actions are represented using a relational representation, typically in the form of a logic program.

Much research in RRL focuses on Q-learning, which represents the knowledge of the agent by means of a Q-function mapping state–action pairs to real values. During exploration, the agent selects in each state the action that is ranked highest by the Q-function. The Q-function is typically represented using a relational regression technique. Several techniques, such as relational regression trees, relational instance based learning, and relational kernel based regression have been considered in this context. Note that the regression algorithms must be able to learn incrementally: each time the agent receives a new reward, the Q-function must be incrementally updated for the episode (sequence of state-action pairs) that led to the reward. Due to the use of relational regression techniques, the agent is able to generalize over states: it will perform similar actions in similar states and therefore scales better to large application domains.

More recent topics in RRL include how expert knowledge can be provided to the agent in the form of guidance, and how learned knowledge can be transferred to related domains ("transfer learning"). More details on these techniques and more specific information on the topic of relational reinforcement learning can be found in its corresponding encyclopedia entry and in the related entry on ►symbolic dynamic programming, as well as in references (Džeroski et al., 2001; Tadepalli et al., 2004).

Cross References

►Inductive Logic Programming
►Multi-Relational Data Mining
►Relational Reinforcement Learning

Recommended Reading

Most of the topics covered in this entry have more detailed entries in this encyclopedia, namely "Inductive Logic Programming," "Graph Mining," "Relational Data Mining," and "Relational Reinforcement Learning." These entries provide a brief introduction to these more specific topics and appropriate references for further reading.

Direct relevant references to the literature include the following. A comprehensive introduction to ILP can be found in De Raedt's book (De Raedt, 2008) on logical and relational learning, or in the collection edited by Džeroski and Lavrač (2001) on relational data mining. Learning from graphs is covered by Cook and Holder (2007). Džeroski and Lavrač (2001) is also a good starting point for reading about multi-relational data mining, together with research papers on multi-relational data mining systems, for instance, Yin et al. (2006), who present a detailed description of the CrossMine system. Statistical relational learning in general is covered in the collection edited by Getoor & Taskar (2007), while De Raedt & Kersting (2003) and De Raedt et al. (2008) present overviews of approaches originating in logic-based learning. An overview of relational reinforcement learning can be found in Tadepalli et al. (2004).

Bratko, I. (2000). *Prolog programming for artificial intelligence*. Reading, MA: Addison-Wesley (3rd ed.).
Cook, D. J., & Holder, L. B. (2007). *Mining graph data*. Hoboken, NJ: Wiley.
De Raedt, L. (2008). *Logical and relational learning*. Berlin: Springer.
De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (2008). *Probabilistic inductive logic programming*. Berlin: Springer.
De Raedt, L., & Kersting, K. (2003). Probabilistic logic learning. *SIGKDD Explorations, 5*(1), 31–48.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*, 7–52.
Džeroski, S., & Lavrač, N., (Eds.). (2001). *Relational data mining*. Berlin: Springer.
Finn, P., Muggleton, S., Page, D., & Srinivasan, A. (1998). Pharmacophore discovery using the inductive logic programming system PROGOL. *Machine Learning, 30*, 241–270.
Getoor, L., & Taskar, B. (2007). *Introduction to statistical relational learning*. Cambridge: MIT Press.
Horváth, T., Ramon, J., & Wrobel, S. (2006). Frequent subgraph mining in outerplanar graphs. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 197–206). New York: ACM.
Jensen, D., & Neville, J. (2002). Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceeding of the 19th International Conference on Machine Learning*, University of New South Wales, Sydney (pp. 259–266). San Francisco, CA: Morgan Kaufmann.
Jensen, D., Neville, J., & Gallagher, B. (2004). Why collective inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD international conference on knowledge discovery and data mining*, Philadelphia, PA (pp. 593–598). New York: ACM.
Kersting, K. (2006). *An inductive logic programming approach to statistical relational learning*. Amsterdam: IOS Press.
Krogel, M.-A., Rawles, S., Železný, F., Flach, P., Lavrač, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th international conference on inductive logic programming*, Szeged, Hungary (pp. 194–217). Berlin: Springer-Verlag.
Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.
Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 254–264). Amsterdam: IOS Press.
Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*(1–2), 107–136.
Sato, T., & Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proceedings of the 15th International joint conference on artificial intelligence (IJCAI 97)*, Nagoya, Japan (pp. 1330–1335). San Francisco, CA: Morgan Kaufmann.
Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proceeding of the ICML'04 Workshop on relational reinforcement learning*, Banff, Canada (pp. 1–9).
Washio, T., & Motoda, H. (2003). State of the art of graph-based data mining. *SIGKDD Explorations, 5*(1), 59–68.
Yin, X., Han, J., Yang, J., & Yu, P. S. (2006). Efficient classification across multiple database relations: A CrossMine approach. *IEEE Transactions on Knowledge and Data Engineering, 18*(6), 770–783.

Relational Regression Tree

►First-Order Regression Tree

Relational Reinforcement Learning

Kurt Driessens
Universiteit Leuven, Celestijnenlaan, Belgium

Synonyms

Learning in worlds with objects; Reinforcement learning in structured domains

Definition

Relational reinforcement learning is concerned with learning behavior or control policies based on a numerical feedback signal, much like standard reinforcement learning, in complex domains where states (and actions) are largely characterized by the presence of objects, their properties, and the existing relations between those objects. Relational reinforcement learning uses approaches similar to those used for standard reinforcement learning, but extends these with methods that can abstract over specific object identities and exploit the structural information available in the environment.

Motivation and Background

►Reinforcement learning is a very attractive machine learning framework, as it tackles – in a sense – the whole artificial intelligence problem at a small scale: an agent acts in an unknown environment and has to learn how to behave optimally by reinforcement, i.e., through rewards and punishment. Reinforcement learning has produced some impressive and promising results. However, the applicability of reinforcement learning has been greatly limited by its problem of dealing with large problem spaces and its inability to generalize the learned knowledge to new but related problem domains.

While standard reinforcement learning methods represent the learning environment as a set of unrelated states or, when using ►attribute-value representations, as a vector space consisting of a fixed number of independent dimensions, humans tend to think about their environment in terms of objects, their properties, and the relations between them. Examples of objects in everyday life are chairs, people, streets, trees, etc. This representation allows people to treat or use most of the new objects that they encounter correctly, without requiring training time to learn how to use them. For example, people are able to drink their coffee from any cup that will hold it, even if they have never encountered that specific cup before, because they already have experience with drinking their coffee from other cup-type objects. Standard reinforcement learning agents do not have this ability. Their state and action representations do not allow them to abstract away from specific object-identities and recognize them as a type of object they are already accustomed to.

Relational reinforcement learning tries to overcome this problem by representing states of the learning agent's environment as sets of objects, their properties, and the relationships between them, similar to the approaches used in ►relational learning and ►inductive logic programming. These structural representations make it possible for the relational reinforcement learning agent to abstract away from specific identities of objects and often also from the amount of objects present, the exact learning environment, or even the specific task to be performed.

The term "Relational reinforcement learning" was introduced by Džeroski, De Raedt, and Blockeel (1998) when they first teamed the Q-learning algorithm with a first-order regression algorithm. Since then, relational reinforcement learning has gained an increasing amount of interest.

Structure of the Learning System

In principle, the structure of a relational reinforcement learning system is very similar to that of standard reinforcement learning systems Fig. 3. At a high level, the learning agent interacts with an environment by performing actions that influence that environment, and the environment provides the learning agent with a description of its current state and a numerical feedback of the behavior of the agent. The goal of the agent is to maximize some cumulative form of this feedback signal. The major difference between standard reinforcement learning and relational reinforcement learning is the representation of the state–action–space. Relational reinforcement learning works on ►Markov decision processes where states and actions have been relationally factored, so-called relational Markov decision processes (RMDPs).

An RMDP can be defined as follows:

Definition 1 (**Relational Markov Decision Process**) *Let P_S be a set of state related predicates, P_A a set of action related predicates and C a set of constants in a logic Λ. Let $\mathcal{B}$ be a theory defined in that logic.*
An RMDP
is defined as $< S, A, T, R >$, where $S \equiv \{s \subset H^{P_S \cup C} | s \models \mathcal{B}\}$ represents the set of states, $A \equiv \{a \subset H^{P_A \cup C} | a \models \mathcal{B}\}$ represents the set of actions, in which H^X is the set of facts that can be constructed given the symbols in X, and T and R represent the transition probabilities and reward function respectively: $T : S \times A \times S \rightarrow [0,1]$ and $R : S \rightarrow \mathbb{R}$.

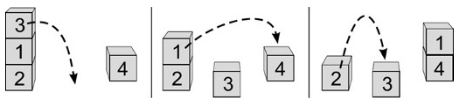
In less formal language, this means that the states and actions in an RMDP are represented using a set of constants C and a set of predicates P_S and P_A respectively and constrained by a background theory $\mathcal{B}$. This means that the background theory $\mathcal{B}$ defines which states are possible in the domain and which actions can be executed in which states.

The following example illustrates these concepts. Consider the blocks world depicted in Fig. 1. To represent this environment in first-order logic, one could use:

- State related predicates: $P_S = \{\text{on/2, clear/1}\}$
- Action related predicate: $P_A = \{\text{move/2}\}$
- Constants: $C = \{1,2,3,4,\text{floor}\}$

The set of facts $H^{P_S \cup C}$ would then include, for example: $on(1,2)$, $on(4, floor)$ and $clear(2)$ but also $on(3,3)$ and $on(floor, 2)$. To constrain the possible states to those that actually make sense in a standard, i.e., real-world view of the blocks world, the theory $\mathcal{B}$ can include rules to make states that include these kinds of facts impossible. For example, to make sure that a block cannot be on top of itself, $\mathcal{B}$ could include the following rule:

$$\texttt{false} \leftarrow on(X,X).$$



Relational Reinforcement Learning. Figure 1. Example state–action pairs in the Blocks World

One can also include more extensive rules to define the exact physics of the blocks world that one is interested in. For example, including

$$\texttt{false} \leftarrow on(Y,X), on(Z,X), X \neq floor, Y \neq Z$$

as part of the theory $\mathcal{B}$, one can exclude states where two blocks are on top of the same block. The action space given by $H^{P_A \cup C}$ consists of facts such as $move(3,2)$ and $move(floor,1)$ and can be constrained by rules such as:

$$\texttt{false} \leftarrow move(floor, X).$$

which makes sure that the floor cannot be placed on top of a block.

The leftmost state–action pair of Fig. 1 can be fully specified by the following set of facts (state description on the left, action on the right):

on(2,floor).	clear(3).
on(1,2).	clear(4).
on(3,1).	clear(floor).
on(4,floor).	

One can easily generalize over specific states and create abstract states (or state–action pairs) that represent sets of states (or state–action pairs) by using variables instead of constants and by listing only those parts of states and actions that hold for each element of the abstract state (or state–action pair). For example, the abstract state "$on(1,2), on(2,floor)$" represents all states in which block 1 is on top of block 2, which in turn is on the floor. The abstract state does not specify the locations of any other blocks. Of the three states depicted in Fig. 1, the set of states represented by the abstract state would include the left and middle states. Abstract states can also be represented by using variables when one does not want to specify the location of any specific block, but wants to focus on structural aspects of the states and actions. The abstract state–action pair "$move(X,Y), on(Y,floor)$" represents all state–action pairs where a block is moved on top of another block that is on the floor, for example the middle and right state–action pairs of Fig. 1.

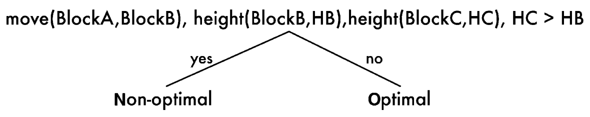
Added Benefits of Relational Reinforcement Learning

We already stated that the real world is made up out of interacting objects, or at least that humans often think about the real world as such. Relational reinforcement learning allows this same representation to be used by reinforcement learning agents, which in turn leads to more human-interpretable learning results.

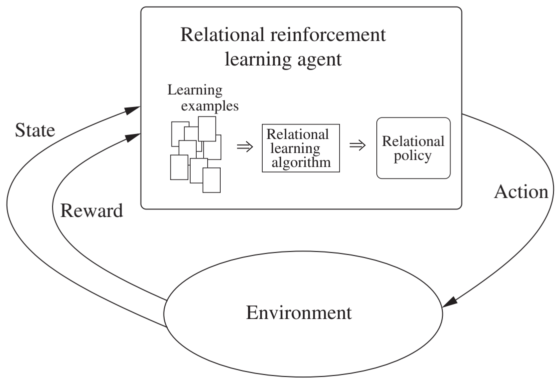
As a consequence of the used logical or relational representation of states and actions, the results learned by a relational reinforcement learning agent can be re-used more easily when some of the parameters of the learning task change. Because relational reinforcement learning algorithms try to solve the problem at hand at an abstract level, the solutions will often carry over to different instantiations of that abstract problem. For example, the resulting policies learned by the RRL system (Driessens, 2004) discussed below, a very simple example of which is shown in Fig. 2, often generalize over domains with a varying number of objects. If only actions which lead to the "*optimal*" leaf are executed, the shown policy tree will organize any number of blocks into a single stack.

As another example of this, the relational approximate policy iteration approach (Fern, Yoon, & Givan, 2006), also discussed below, is able to learn task specific control knowledge from random walks in the environment. By treating the resulting state of such a random walk as a goal state and generalizing over the specifics of that goal (and the rest of the random walk) relational approximate policy iteration can learn domain specific, but goal independent policies. This generalization of the policy is accomplished by parametrization of the goal and focusing on the relations between objects in the goal, states and actions when representing the learned policy.

Another practical benefit of relational reinforcement learning lies in the field of inductive transfer. Transfer learning is concerned with the added benefits of having experience with a related task when being confronted with a new one. Because of the structural representation of learned results, the transfer of knowledge learned by relational reinforcement learning agents can be accomplished by recycling those parts


move(BlockA,BlockB), height(BlockB,HB),height(BlockC,HC), HC > HB
yes
no
Non-optimal
Optimal

Relational Reinforcement Learning. Figure 2. Simple relational policy for stacking any number of blocks


Relational reinforcement learning agent
Learning examples
Relational learning algorithm
Relational policy
State
Reward
Action
Environment

Relational Reinforcement Learning. Figure 3. Structure of the RRL system

of the results that still hold valid information for the new task. Depending on the relation between the two tasks, this can yield substantial benefits concerning the required training experience.

The use of first-order logic as a representational language in relational reinforcement learning also allows the integration of reasoning methods with traditional reinforcement learning approaches. One example of this is ►Symbolic Dynamic Programming, which uses logical regression to compute necessary preconditions that allow an agent to reach certain goals. This same integration allows the use of search or planning knowledge as background information to extend the normal description of states and actions.

Example Relational Reinforcement Learning Approaches

Relational Q-Learning Relational reinforcement learning was introduced with the development of the RRL-system (Džeroski et al., 1998). This is a Q-learning system that employs a relational regression algorithm to generalize the Q-table used by standard Q-learning algorithms into a Q-function. The differences with a standard Q-learning agent are mostly located inside the learning agent. One important difference is the agent's representation of the current state. In relational reinforcement learning, this representation contains structural or relational information about the environment.

Inside the learning agent, the information consisting of encountered states, chosen actions, and the connected rewards is translated into learning examples. These examples are then processed by a relational learning system that produces a relational Q-function and/or policy as a result. The relational representation of the Q-function allows the RRL-system to use the structural properties of states and actions when assigning a Q-value to them.

Several relational regression approaches have been developed and applied in this context. While the original approach used an of-the-shelve relational regression algorithm that processed the learning examples in batch and had to be restarted to be able to process newly available learning experiences, a number of incremental algorithms have been developed for use in relational reinforcement learning since then. These include an incremental first-order regression tree algorithm, incremental relational instance based regression, kernel based regression that uses Gaussian processes, and graph-kernels and algorithms that include combinations of the above (Driessens, 2004).

It is possible to translate the learned Q-function approximations into a function that directly represents its policy. Using the values predicted by the learned Q-function, one can generate learning examples that represent state–action pairs and label them as either part of the learned policy or not. This results in a binary classification problem that can be handled by a supervised relational learning algorithms. This technique is known as P-learning (Džeroski, De Raedt, & Driessens, 2001). It exhibits better generalization performance across related learning problems than the Q-learning approach described above. Other than the first-order decision trees mentioned above, rule-based learners have also been applied to this kind of policy learning.

Non-parametric Policy Gradients Non-parametric policy gradients (Kersting & Driessens, 2008), also a model-free approach, apply Friedmann's gradient boosting (Friedman, 2001) in an otherwise standard policy gradient approach for reinforcement learning. To avoid having to represent policies using a fixed number of parameters, policies are represented as a weighted sum of regression models grown in a stage-wise optimization (This allows the number of parameters to grow as the experience of the learner increases, hence the name non-parametric.). While this does not make non-parametric policy gradients a technique specifically designed for relational reinforcement learning, it allows, like the relational Q-learning approach described above, the use of relational regression models and is not constrained to the attribute-value setting of standard policy gradients.

The idea behind the approach is that instead of finding a single, highly accurate policy, it is easier to find many rough rules of thumb of how to change the way the agent currently acts. The learned policy is represented as

$$\pi(s,a) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}} ,$$

where instead of assuming a linear parameterization for Ψ as is done in standard policy gradients, it is assumed that Ψ will be represented by a linear combination of functions. Specifically, one starts with some initial function Ψ_0, e.g., based on the zero potential, and iteratively adds corrections $\Psi_m = \Psi_0 + \Delta_1 + \cdots + \Delta_m$. In contrast to the standard gradient approach, Δ_m here denotes the so-called functional gradient, which is sampled during interaction with the environment and then generalized by an off-the-shelf regression algorithm.

The advantages of policy gradients over value-function techniques are that they can learn non-deterministic policies and that convergence of the learning process can be guaranteed, even when using function approximation (Sutton, McAllester, Singh, & Mansour, 2000). Experimental results show that non-parametric policy gradients have the potential to significantly outperform relational Q-learning (Kersting & Driessens, 2008).

Relational Approximate Policy Iteration A different approach, which also directly learns a policy, is taken in relational approximate policy iteration (Fern et al., 2006). Like standard policy iteration, the approach iteratively improves its policy through interleaving evaluation and improvement steps. In contrast to standard policy iteration, it uses a policy language bias and a generalizing policy function.

Instead of building a value-function approximation for each policy evaluation step, relational approximate policy iteration evaluates the current policy and its closely related neighbors by sampling the state–action–space through a technique called policy roll-out. This technique generates a set of trajectories from a given state, by executing every possible action in that state and following the current policy for a number of steps afterward (It is also possible to improve convergence speed by following the next policy.). These trajectories and their associated costs result in number of learning examples – one for each possible action in each selected state – that can be used, together with the policy language bias to generate the next, improved policy.

Because every possible action in each sampled state needs to be evaluated, this approach does require a model or a resettable simulator of the environment. However, relational approximate policy iteration has been shown to work well for learning domain specific control knowledge and performs very well on planning competition problems.

Symbolic Dynamic Programming In contrast to the previous techniques, ▶symbolic dynamic programming (SDP) does not learn a policy through exploration of the environment. Instead, it is a model-based approach that uses knowledge about preconditions and consequences of actions to compute the fastest way to reach a given goal. Like other dynamic programming techniques, SDP starts from the goal the agent wants to reach and reasons backwards to find the policy that is needed to reach that goal. In contrast to other dynamic programming techniques, it does not solve specific instantiations of the problem domain, but instead solves the problem at an abstract level, thereby solving it for all possible instantiations of the problem at once.

SDP treats the required goal-conditions as an abstract state definition. Because pre- and post-conditions of actions are known, SDP can compute the necessary conditions that allow actions to reach the abstract goal state. These conditions define abstract states from which it is possible to reach a goal state in one step. Starting from these abstract states, the same approach can be used to discover abstract states that allow the goal to be reached in two steps and so on.

This approach was first proposed by Boutilier, Reiter, and Price (2001), implemented as a working system by Kersting, van Otterlo, and De Raedt (2004) and later improved upon by Sanner and Boutilier (2005). This last approach won 2nd place in the probabilistic programming competition at ICAPS in 2006.

Cross References

►Hierarchical Reinforcement Learning
►Inductive Logic Programming
►Model-Based Reinforcement Learning
►Policy Iteration
►Q-learning
►Reinforcement Learning
►Relational Learning
►Symbolic Dynamic Programming
►Temporal Difference Learning

Further Information

The field of relational reinforcement learning has given rise to a number of PhD dissertations in the last few years (Croonenborghs, 2009; Driessens, 2004; Sanner, 2008; van Otterlo, 2008). The dissertation of Martijn van Otterlo resulted in a book (van Otterlo, 2009) which provides a recent and reasonably complete overview of the relational reinforcement learning research field. Other publications that presents an overview of relational reinforcement learning research include the proceedings of the two workshops on representational issues in (relational) reinforcement learning at the International Conferences of Machine Learning in 2004 and 2005 (Driessens, Fern, & van Otterlo, 2005; Tadepalli, Givan, & Driessens, 2004).

Recommended Reading

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 17th international joint conference on artificial intelligence (IJCAI-2001)*, Seattle, WA (pp. 690–700).

Croonenborghs, T. (2009). *Model-assisted approaches for relational reinforcement learning*. PHD thesis, Department of Compute Science, Katholieke Universiteit Leuven.

Driessens, K. (2004). *Relational reinforcement learning*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven.

Driessens, K., Fern, A., & van Otterlo, M. (Eds.). (2005). *Proceedings of ICML-2005 workshop on rich representation for reinforcement learning*, Bonn, Germany.

Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. In *Proceedings of the 15th international conference on machine learning (ICML-1998)* (pp. 136–143). San Francisco, CA: Morgan Kaufmann. Madison, WI, USA.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*, 7–52.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research, 25*, 85–118.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*, 1189–1232.

Kersting, K., & Driessens, K. (2008). Non-parametric policy gradients: A unified treatment of propositional and relational domains. In A. McAllum & S. Roweis (Eds.), *Proceedings of the 25th international conference on machine learning (ICML 2008)*, Helsinki, Finland (pp. 456–463).

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *Proceedings of the twenty-first international conference on machine learning (ICML-2004)*, Banff, Canada (pp. 465–472).

Sanner, S. (2008). *First-order decision-theoretic planning in structured relational environments*. PhD thesis, Department of Compute Science, University of Toronto.

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. In *Proceedings of the 21st conference on Uncertainty in AI (UAI)*, Edinburgh, Scotland.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems 12* (pp. 1057–1063). Cambridge: MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (Eds.). (2004). *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, Banff, Canada.

van Otterlo, M. (2008). *The logic of adaptive learning*. PhD thesis, Centre for Telematics and Information Technology, University of Twente.

van Otterlo, M. (2009). The logic of adaptive behavior: Knowledge representation and algorithms for adaptive sequential decision making under uncertainty in first-order and relational domains. Amsterdam, The Netherlands: IOS Press.

Relational Value Iteration

►Symbolic Dynamic Programming

Relationship Extraction

►Link Prediction

Relevance Feedback

Relevance feedback provides a measure of the extent to which the results of a search match the expectations of the user who initiated the query. Explicit feedback require users to assess relevance by choosing one out of a number of choices, or to rank documents to reflect their perceived degree of relevance. Implicit feedback is obtained by monitoring user's behavior such as time spent browsing a document, amount of scrolling performed while browsing a document, number of times a document is visited, etc. Relevance feedback is one the techniques used to support query reformulation and turn the search into an iterative and interactive process.

Cross References

►Search Engines: Applications of ML

Representation Language

►Hypothesis Language

Reservoir Computing

Risto Miikkulainen
The University of Texas at Austin, Austin, TX, USA

Synonyms

Echo state network; Liquid state machine

Definition

Reservoir computing is an approach to sequential processing where recurrency is separated from the output mapping (Jaeger, 2003; Maass, Natschlaeger, & Markram, 2002). The input sequence activates neurons in a recurrent neural network (a reservoir, where activity propagates as in a liquid). The recurrent network is large, nonlinear, randomly connected, and fixed. A linear output network receives activation from the recurrent network and generates the output of the entire machine. The idea is that if the recurrent network is large and complex enough, the desired outputs can likely be learned as linear transformations of its activation. Moreover, because the output transformation is linear, it is fast to train. Reservoir computing has been successful in particular in speech and language processing and vision and cognitive neuroscience.

Recommended Reading

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems* (Vol. 15, pp. 593–600). Cambridge, MA: MIT Press.

Maass, W., Natschlaeger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*, 2531–2560.

Resolution

►First-Order Logic

Resubstitution Estimate

Resubstitution estimates are estimates that are derived by applying a ►model to the ►training data from which it was learned. For example, *resubstitution error* is the error of a model on the training data.

Cross References

►Model Evaluation

Reward

In most *Markov decision process* applications, the decision-maker receives a *reward* each period. This reward can depend on the current *state*, the *action* taken, and the next state and is denoted by $r_t(s, a, s')$.

Reward Selection

►Reward Shaping

Reward Shaping

Eric Wiewiora
University of California, San Diego

Synonyms

Heuristic rewards; Reward selection

Definition

Reward shaping is a technique inspired by animal training where supplemental rewards are provided to make a problem easier to learn. There is usually an obvious natural reward for any problem. For games, this is usually a win or loss. For financial problems, the reward is usually profit. Reward shaping augments the natural reward signal by adding additional rewards for making progress toward a good solution.

Motivation and Background

Reward shaping is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. It is most often discussed in the ►reinforcement learning framework. Providing feedback is crucial during early learning so that promising behaviors are tried early. This is necessary in large domains, where reinforcement signals may be few and far between.

A good example of such a problem is chess. The objective of chess is to win a match, and an appropriate reinforcement signal should be based on this. If an agent were to learn chess without prior knowledge, it would have to search for a great deal of time before stumbling onto a winning strategy. We can speed up this process by rewarding the agent more frequently. One possibility is to reward the learner for capturing enemy pieces, and punish the learner for losing pieces. This new reward creates a much richer learning environment, but also runs the risk of distracting the agent from the true goal (winning the game).

Another domain where feedback is extremely important is in robotics and other real-world applications. In the real world, learning requires a large amount of interaction time, and may be quite expensive. Mataric noted that in order to mitigate "thrashing" (repeatedly trying ineffective actions) rewards should be supplied as often as possible (Mataric, 1994).

If a problem is inherently described by sparse rewards, it may be difficult to change the reward structure without disrupting progress to the original goal. The behavior that is optimal with a richer reward function may be quite different from the intended behavior, even if relatively small shaping rewards are added. A classic example of this is found in Randlov and Alsrom (1998). While training an agent to control a bicycle simulation, they rewarded an agent whenever it moved toward a target destination. In response to this reward, the agent learned to ride in a tight circle, receiving reward whenever it moved in the direction of the goal.

Theory

We assume a reinforcement learning framework. For every time step t, the learner observes state s_t, takes action a_t, and receives reward r_t. The goal of reinforcement learning is to find a policy $\pi(s)$ that produces actions that optimize some long-term measurement of reward. We define the value function for every state as the expected infinite horizon discounted reward

$$V(s) = \max_{\pi} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_t = \pi(s_t)\right],$$

where γ is the discount rate. A reinforcement learner's goal is to learn a good estimate of $V(s)$, and to use this estimate to choose a good policy.

A natural reward source should be fairly obvious from the problem at hand. Financial problems should use net monetary gain or loss as reward. Games and goal-directed problems should reward winning the game or reaching the goal. It is usually advantageous to augment this natural reward with a shaping reward f_t. We define the augmented value function V' for the reinforcement learning problem with shaping rewards

$$V'(s) = \max_{\pi'} \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t (r_t + f_t) | s_0 = s, a_t = \pi'(s_t)\right].$$

Ideally, the policy that optimizes the augmented value function will differ much from the previous optimal policy.

Constructing an appropriate shaping reward system is inherently a problem-dependent task, though a line of research aids in the implementation of these reward signals. *Potential-based shaping* provides a formal framework for translating imperfect knowledge of the relative value of states and actions into a shaping reward.

Potential-Based Shaping

Ng et al. proposed a method for adding shaping rewards in a way that guarantees the optimal policy maintains its optimality (Ng, Harada, & Russell, 1999). They define a potential function $\Phi()$ over the states. The shaping reward f for transitioning from state s to s' is defined as the discounted change in this state potential:

$$f(s, s') = \gamma\Phi(s') - \Phi(s).$$

This potential-based shaping reward is added to the natural reward for every state transition the learner experiences. Call the augmented reward $r'_t = r_t + f(s_t, s_{t+1})$, and the value function based on this reward $V'(s)$. The potential-based shaping concept can also be applied to actions as well as states. See Wiewiora, Cottrell, & Elkan (2003) for details.

It can be shown that the augmented value function is closely related to the original:

$$V'(s) = V(s) - \Phi(s).$$

An obvious choice for the potential function is $\Phi(s) \approx V(s)$, making $V'()$ close to zero. This intuition is strengthened by results presented by Wiewiora (2003). This paper shows that for most reinforcement learning systems, the potential function acts as an initial estimate of the natural value function $V()$.

However, even if the potential function used for shaping is very close to the true natural value function, learning may still be difficult. Koenig et al. have shown that initial estimates of the value function have a large influence on the efficiency of reinforcement learning (Koenig & Simmons, 1996). With an initial estimate of the value function set below the optimal value, many reinforcement learning algorithms could require learning time exponential in the state and action space in order to find a highly rewarding state. On the other hand, in nonrandom environments, an optimistic initialization the value function creates learning time that is polynomial in the state-action space before a goal is found.

Cross References

►Reinforcement Learning

Recommended Reading

Koenig, S., & Simmons, R. G. (1996). The effect of representation and knowledge on goal directed exploration with reinforcement-learning algorithms. *Machine Learning, 22*(1–3), 227–250.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *International conference on machine learning*, New Brunswick, NJ (pp. 181–189). San Francisco, CA: Morgan Kaufmann.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Machine learning, proceedings of the sixteenth international conference*, Bled, Slovenia (pp. 278–287). San Francisco, CA: Morgan Kaufmann.

Randlov, J., & Alstrom, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the fifteenth international conference on machine learning*, Madison, WI. San Francisco, CA: Morgan Kaufmann.

Wiewiora, E., Cottrell, G., & Elkan, C. (2003). Principled methods for advising reinforcement learning agents. In *Machine learning, proceedings of the twentieth international conference*, Washington, DC (pp. 792–799). Menlo Park, CA: AAAI Press.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research, 19*, 205–208.

RIPPER

►Rule Learning

Robot Learning

Jan Peters[1], Russ Tedrake[2], Nicholas Roy[2], Jun Morimoto[3]
[1]Max Planck Institute for Biological Cybernetics, Germany
[2]Massachusetts Institute of Technology, Cambridge, MA, USA
[3]Advanced Telecommunication Research Institute International ATR, Kyoto, Japan

Definition

Robot learning consists of a multitude of machine learning approaches, particularly ►reinforcement learning, ►inverse reinforcement learning and ►regression methods. These methods have been adapted sufficiently to domain to achieve real-time learning in complex robot systems such as helicopters, flapping-wing flight, legged robots, anthropomorphic arms, and humanoid robots.

Robot Skill Learning Problems

In classical artificial intelligence-based robotics approaches, scientists attempted to manually generate a set of rules and models that allows the robot systems to sense and act in the real world. In contrast, ►robot learning has become an interesting problem in robotics as (1) it may be prohibitively hard to program a robot for many tasks, (2) not all situations, as well as goals, may be foreseeable, and (3) real-world environments are often nonstationary (Connell and Mahadevan, 1993). Hence, future robots need to be able to adapt to the real world.

In comparison to many other machine learning domains, robot learning suffers from a variety of complex real-world problems. The real-world training time is limited and, hence, only a few complete executions of a task can ever be generated. These episodes are frequently perceived noisily, have a large variability in the executed actions, do not cover all possible scenarios, and often do not include all reactions to external stimuli. At the same time, high-dimensional data is obtained at a fast rate (e.g., proprioceptive information at 500 Hz to 5 kHz, vision at 30–200 Hz). Hence, domain-appropriate machine learning methods are often needed in this domain.

A straightforward way to categorize robot learning approaches is given by the type of feedback (Connell and Mahadevan, 1993). A scalar performance score such as a reward or cost will often result in a ►reinforcement learning approach. A presented desired action or predicted behavior allows supervised learning approaches such as model learning or direct imitation learning. Feedback in terms of an explanation has become most prominent in apprenticeship learning. These methods will be explained in more detail in the next section. Note that unsupervised learning problems, where no feedback is required can also be found in robotics, see Ham et al. (2005) and Jenkins et al. (2006) but only for special topics.

Note that this overview on ►robot learning focuses on general problems that need to be addressed to teach robots new skills or tasks. Hence, several important specific robotics problems in specialized domains such as ►simultaneous localization and map building (SLAM) for mobile robots (Thrun et al., 2005) and unsupervised sensor fusion approaches for robot perception (Apolloni et al., 2005; Jenkins et al., 2006) are considered beyond the scope of this article.

Robot Learning Systems

As learning has found many applications in robotics, this article can only scratch the surface. It focuses on the key problem of teaching a robot new abilities with methods such as (1) Model Learning, (2) Imitation and Apprenticeship Learning, and (3) Reinforcement Learning.

Model Learning

Model learning is the machine learning counterpart to classical system identification (Farrell and Polycarpou, 2006; Schaal et al., 2003). However, while the classical approaches heavily rely on the structure of physically based models, specification of the relevant state variables and hand-tuned approximations of unknown nonlinearities, model learning approaches avoid many of these labor-intensive steps and the entire process to be more easily automated. Machine learning and system identification approaches often assume an observable state of the system to estimate the mapping from inputs to outputs of the system. However, a learning system is often able to learn this mapping including the statistics needed to cope with unidentified state variables and can hence cope with a larger class of systems. Two types of models are commonly learned, i.e., forward models and inverse models.

Forward models predict the behavior of the system based either on the current state or a history of preceeding observations. They can be viewed as "learned simulators" that may be used for optimizing a policy or for predicting future information. Examples of the application of such learned simulators range from the early work in the late 1980s by Atkeson and Schaal in robot arm-based cartpole swing-ups to Ng's recent extensions for stabilizing an inverted helicopter. Most forward models can directly be learned by **regression**.

Conversely, *inverse models* attempt to predict the input to a system in order to achieve a desired output in the next step, i.e., it uses the model of the system to directly generate control signals. In traditional control, these are often called approximation-based control systems (Farrell and Polycarpou, 2006). Inverse model learning can be solved straightforwardly by **regression** if the system dynamics are uniquely invertible, e.g., as in inverse dynamics learning for a fully actuated system. However, for underactuated or redundantly actuated systems, operational space control, etc., such a unique inverses do not exist and additional optimization is needed.

Imitation and Apprenticeship Learning

A key problem in robotics is to ease the problem of programming a complex behavior. Traditional robot programming approaches rely on accurate, manual modeling of the task and removal of all uncertainities, so that they work well. In contrast to classical robot programming, learning from demonstration approaches aim at recovering the instructions directly from a human demonstration. Numerous unsolved problems exist in this context such as discovering the intent of the teacher or determing the mapping from the teacher's kinematics to the robot's kinematics (often called the correspondence problem). Two different approaches are common in this area: direct imitation learning and apprenticeship learning.

In *imitation learning* (Schaal et al., 2003), also known as ▶behavioral cloning, the robot system directly estimates a policy from a teacher's presentation, and, subsequently, the robot system reproduces the task using this policy. A key advantage of this approach is that it can often learn a task successfully from few demonstrations. In areas where human demonstrations are straightforward to obtain, e.g., for learning racket sports, manipulation, drumming on anthropomorphic systems, direct imitation learning often proved to be an appropriate approach. Its major shortcomings are that it cannot explain why the derived policy is a good one, and it may struggle with learning from noisy demonstrations.

Hence, *apprenticeship learning* (Coates et al., 2009) has been proposed as an alternative, where a reward function is used as an explanation of the teacher's behavior. Here, the reward function is chosen under which the teacher appears to act optimally, and the optimal policy for this reward function is subsequently computed as a solution. This approach transforms the problem of learning from demonstrations onto the harder problem of approximate optimal control or reinforcement learning, hence it is also known as inverse optimal control or ▶inverse reinforcement learning. As a result, it is limited to problems that can be solved by current reinforcement learning methods. Additionally, it often has a hard time dealing with tasks, where only few demonstrations with low variance exist. Hence, inverse reinforcement learning has been particularly successful in areas where it is hard for a human to demonstrate the desired behavior such as for helicopter acrobatics or in robot locomotion.

Further information on learning by demonstration may be found in Coates et al. (2009) and Schaal et al. (2003).

Robot Reinforcement Learning

The ability to self-improve with respect to an arbitrary reward function, i.e., ▶reinforcement learning, is essential for robot systems to become more autonomous. Here, the system learns about its policy by interacting with its environment and receiving scores (i.e., rewards or costs) for the quality of its performance. Few off-the-shelf reinforcement learning methods scale into the domain of robotics both in terms of dimensionality and the number of trials needed to obtain an interesting behavior. Three different but overlapping styles of reinforcement learning can be found in robotics: model-based reinforcement learning, model-free ▶value function approximation methods, and direct policy search (see ▶Markov Decision Process).

Model-based reinforcement learning relies upon a learned forward model used for simulation-based optimization as discussed before. While often highly efficient, it frequently suffers from the fact that learned models are imperfect and, hence, the optimization method can be guaranteed to be biased by the errors in the model. A full Bayesian treatment of model uncertainty appears to be a promising way for alleviating this shortcoming of this otherwise powerful approach.

Value function approximation methods have been the core approach used in reinforcement learning during the 1990s. These techniques rely upon approximating the expected rewards for every possible action in every visited state. Subsequently, the controller chooses the actions in accordance to this value. Such approximation requires a globally consistent value function, where the quality of the policy is determined by the largest error of the value function at any possible state. As a result, these methods have been problematic for anthropomorphic robotics as the high-dimensional domains often defy learning such a global construct. However, it has been highly sucessful in low-dimensional domains such as mobile vehicle control and robot soccer with wheeled robots as well as on well-understood test domains such as cart-pole systems.

Unlike the previous two approaches, *policy search* attempts to directly learn the optimal policy from experience without solving intermediary learning problems. Policies often have significantly fewer parameters than models or value functions. For example, for the control of a prismatic robot optimally with respect to a quadratic reward function, the number of policy parameters grows linearly in the number of state dimensions, while it grows quadratically in the size of the model and value function (this part is well-known as this problem is analytically tractable). In general cases, the number of parameters of value functions does often even grow exponentially in the number of states (which is known as the "Curse of Dimensionality"). This insight has given rise to policy search methods, particularly, ►policy gradient methods and probabilistic approaches to policy search such as the reward-weighted regression or PoWER (Kober and Peters, 2009). To date, application results of direct policy search approaches range from gait optimization in locomotion (Tedrake et al., 2004) to various motor learning examples (e.g., Ball-in-a-Cup, T-Ball, or throwing darts, see e.g., Kober and Peters, 2009).

Further information on reinforcement learning for robotics may be found in Connell and Mahadevan (1993), Kober and Peters (2009), Riedmiller et al. (2009), and Tedrake et al. (2004).

Application Domains

The possible application domains for robot learning have not been fully explored; one could even aggressively state that a huge number of challenges remain to be fully addressed in order to solve the problem of robot learning. Nevertheless, robot learning has been successful in several application domains.

For accurate execution of desired trajectories, model learning has been scaled to learning the full inverse dynamics of a humanoid robot in real time more accurately than achievable with physical models (Schaal et al., 2002). Current work focuses mainly on improving the concurrent execution of tasks as well as control of redundant or underactuated systems.

Various approaches have been successful in task learning. Learning by demonstration approaches are moving increasingly toward industrial grade solutions, where fast training of complex tasks becomes possible. Skills ranging from motor toys, e.g., basic movements, paddling a ball, to complex tasks such as cooking a complete meal, basic table tennis strokes, helicopter acrobatics, or footplacement in locomotion have been learned from human teachers. Reinforcement learning has yielded better gaits in locomotion, jumping behaviors for legged robots, perching with fixed wing flight robots, forehands in table tennis as well as various applications of learning to control motor toys used for the motor development of children.

Cross References

►Behavioral Cloning
►Inverse Reinforcement Learning
►Policy Search
►Reinforcement Learning
►Value Function Approximation

Recommended Reading

Recently, several special issues (Morimoto et al., 2010; Peters and Ng, 2009) and books (Sigaud, 2010) have covered the domain of robot learning. The classical book (Connell and Mahadevan, 1993) is interesting nearly 20 years after its publication. Additional special topics are treated in Apolloni et al. (2005) and Thrun et al. (2005).

Apolloni, B., Ghosh, A., Alpaslan, F. N., Jain, L. C., & Patnaik, S. (2005). *Machine learning and robot perception. Studies in computational intelligence* (Vol. 7). Berlin: Springer.

Coates, A., Abbeel, P., & Ng, A. Y. (2009). Apprenticeship learning for helicopter control. *Communications of the ACM, 52*(7), 97–105.

Connell, J. H., & Mahadevan, S. (1993). *Robot learning.* Dordrecht: Kluwer Academic.

Farrell, J. A., & Polycarpou, M. M. (2006). *Adaptive approximation based control. Adaptive and learning systems for signal processing, communications and control series.* Hoboken: John Wiley.

Ham, J., Lin, Y., & Lee, D. D. (2005). Learning nonlinear appearance manifolds for robot localization. In *International conference on intelligent robots and Systems*, Takamatsu, Japan.

Jenkins, O., Bodenheimer, R., & Peters, R. (2006). Manipulation manifolds: Explorations into uncovering manifolds in sensory-motor spaces (8 pages). In *International conference on development and learning*, Bloomington, IN

Kober, J., & Peters, J. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems* 22. Cambridge: MIT Press.

Morimoto, J., Toussaint, M., & Jenkins, C. (2010). Special issue on robot learning in practice. *IEEE Robotics and Automation Magazine, 17*(2), 17–84.

Peters, J., & Ng, A. (2009). Special issue on robot learning. *Autonomous Robots, 27*(1–2):1–144.

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

Riedmiller, M., Gabel, T., Hafner, R., & Lange, S. (July 2009). Reinforcement learning for robot soccer. *Autonomous Robots, 27*(1):55–73.

Schaal, S., Atkeson, C. G., & Vijayakumar, S. Scalable techniques from nonparameteric statistics for real-time robot learning. *Applied Intelligence, 17*(1):49–60.

Schaal, S., Ijspeert, A., & Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences, 358*(1431):537–547.

Sigaud, O., & Peters, J. (2010). *From motor learning to interaction learning in robots. Studies in computational intelligence* (Vol. 264). Heidelberg: Springer.
Tedrake, R., Zhang, T. W., & Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Proceedings of the IEEE international conference on intelligent robots and systems* (pp. 2849–2854). IROS 2004, Sendai, Japan.
Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge: MIT Press.

ROC Analysis

Peter A. Flach
University of Bristol
Bristol, UK

Synonyms

Receiver operating characteristic analysis

Definition

ROC analysis investigates and employs the relationship between ▶sensitivity and ▶specificity of a binary classifier. *Sensitivity* or ▶*true positive rate* measures the proportion of positives correctly classified; *specificity* or ▶*true negative rate* measures the proportion of negatives correctly classified. Conventionally, the true positive rate (tpr) is plotted against the ▶*false positive rate* (fpr), which is one minus true negative rate. If a classifier outputs a score proportional to its belief that an instance belongs to the positive class, decreasing the ▶decision threshold – above which an instance is deemed to belong to the positive class – will increase both true and false positive rates. Varying the decision threshold from its maximal to its minimal value results in a piecewise linear curve from $(0,0)$ to $(1,1)$, such that each segment has a nonnegative slope (Fig. 1). This ROC curve is the main tool used in ROC analysis. It can be used to address a range of problems, including: (1) determining a decision threshold that minimizes ▶error rate or misclassification cost under given class and cost distributions; (2) identifying regions where one classifier outperforms another; (3) identifying regions where a classifier performs worse than chance; and (4) obtaining calibrated estimates of the class posterior.

Motivation and Background

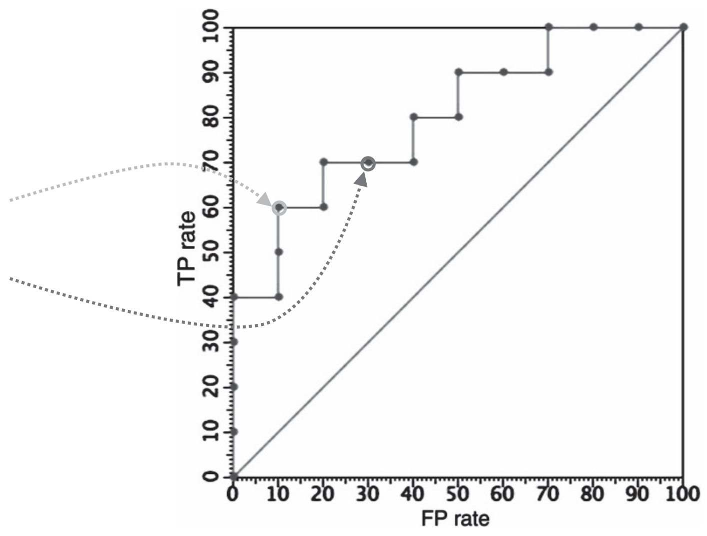
ROC analysis has its origins in *signal detection theory* (Egan, 1975). In its simplest form, a detection problem involves determining the value of a binary signal contaminated with random noise. In the absence of any other information, the most sensible decision threshold would be halfway between the two signal values. If the noise distribution is zero-centered and symmetric, sensitivity and specificity at this threshold have the same expected value, which means that the corresponding *operating point* on the ROC curve is located at the intersection with the descending diagonal tpr + fpr = 1. However, we may wish to choose different operating points, for instance because false negatives and false positives have different costs. In that case, we need to estimate the noise distribution.

A slight reformulation of the signal detection scenario clarifies its relevance in a machine learning setting. Instead of superimposing random noise on a deterministic signal, we can view the resulting noisy signal as coming from a ▶mixture distribution consisting of two component distributions with different means. The detection problem is now to decide, given a received value, from which component distribution it was drawn. This is essentially what happens in a binary ▶classification scenario, where the scores assigned by a trained classifier follow a mixture distribution with one component for each class. The random variations in the data are translated by the classifier into random variations in the scores, and the classifier's performance depends on how well the per-class score distributions are separated. Figure 2 illustrates this for both discrete and continuous distributions. In practice, empirical ROC curves and distributions obtained from a test set are discrete because of the finite resolution supplied by the test set. This resolution is further reduced if the classifier only assigns a limited number of different scores, as is the case with ▶decision trees; the histogram example illustrates this.

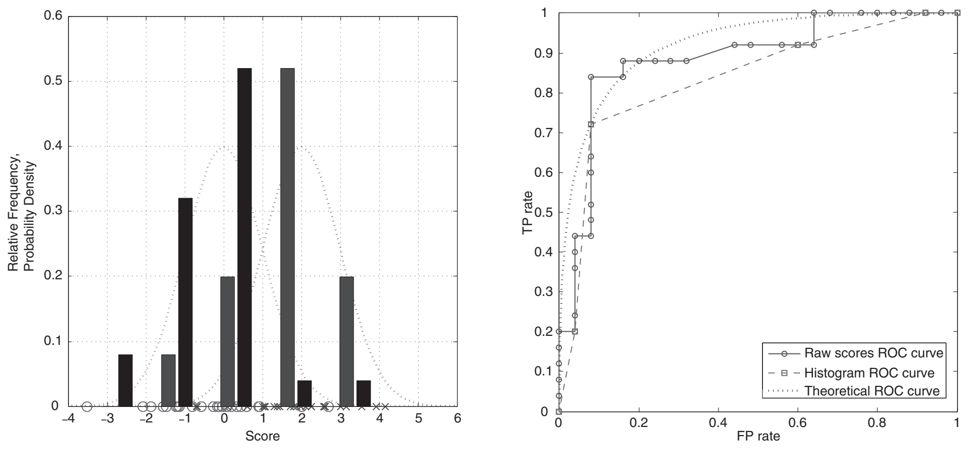
Solutions

For convenience we will assume henceforth that score distributions are discrete, and that decision thresholds always fall between actual scores (the results easily generalize to continuous distributions using probability

Class	Score
+	0.98
+	0.93
+	0.87
+	0.84
–	0.79
+	0.73
+	0.67
–	0.62
+	0.57
–	0.54
–	0.48
+	0.43
–	0.37
+	0.34
–	0.28
–	0.24
+	0.18
–	0.12
–	0.09
–	0.03



ROC Analysis. Figure 1. **The table on the *left* gives the scores assigned by a classifier to ten positive and ten negative examples. Each threshold on the classifier's score results in particular true and false positive rates: e.g., thresholding the score at 0.5 results in three misclassified positives (tpr = 0.7) and three misclassified negatives (fpr = 0.3); thresholding at 0.65 yields tpr = 0.6 and fpr = 0.1. Considering all possible thresholds gives the ROC curve on the right; this curve can also be constructed without explicit reference to scores, by going down the examples sorted on decreasing score and making a step up (to the right) if the example is positive (negative)**



ROC Analysis. Figure 2. **(*left*) Artificial classifier "scores" for two classes were obtained by sampling 25 points each from two ►Gaussian distributions with mean 0 and 2, and unit variance. The figure shows the raw scores on the *x*-axis and normalized histograms obtained by uniform five-bin discretization. (*right*) The jagged ROC curve was obtained by thresholding the raw scores as before. The histogram gives rise to a smoothed ROC curve with only five segments. The dotted line is the theoretical curve obtained from the true Gaussian distributions**

density functions). There is a useful duality between thresholds and scores: decision thresholds correspond to operating points connecting two segments in the ROC curve, and actual scores correspond to segments of the ROC curve connecting two operating points. Let $f(s|+)$ and $f(s|-)$ denote the relative frequency of positive (negative) examples from a test set being assigned score s. (Note that s itself may be an estimate of the likelihood $p(x|+)$ of observing a positive example with feature vector x. We will return to this later.)

Properties of ROC Curves

The first property to note is that the true (false) positive rate achieved at a certain decision threshold t is the proportion of the positive (negative) score distribution to the right of the threshold; that is, $\mathrm{tpr}(t) = \sum_{s>t} f(s|+)$ and $\mathrm{fpr}(t) = \sum_{s>t} f(s|-)$. In Fig. 2, setting the threshold at 1 using the discretized scores gives a true positive rate of 0.72 and a false positive rate of 0.08, as can be seen by summing the bars of the histogram to the right of the threshold. Although the ROC curve does not display thresholds or scores, this allows us to reconstruct the range of thresholds yielding a particular operating point from the score distributions.

If we connect two distinct operating points on an ROC curve by a straight line, the slope of that line segment is equal to the ratio of positives to negatives in the corresponding score interval; that is,

$$\mathrm{slope}(t_1, t_2) = \frac{\mathrm{tpr}(t_2) - \mathrm{tpr}(t_1)}{\mathrm{fpr}(t_2) - \mathrm{fpr}(t_1)} = \frac{\sum_{t_1<s<t_2} f(s|+)}{\sum_{t_1<s<t_2} f(s|-)}$$

Choosing the score interval small enough to cover a single segment of the ROC curve corresponding to score s, it follows that the segment has slope $f(s|+)/f(s|-)$.

This can be verified in Fig. 2: e.g., the top-right segment of the smoothed curve has slope 0 because the leftmost bin of the histogram contains only negative examples. For continuous distributions the slope of the ROC curve at any operating point is equal to the ratio of probability densities at that score.

It can happen that $\mathrm{slope}(t_1, t_2) < \mathrm{slope}(t_1, t_3) < \mathrm{slope}(t_2, t_3)$ for $t_1 < t_2 < t_3$, which means that the ROC curve has a "dent" or *concavity*. This is inevitable when using raw classifier scores (unless the positives and negatives are perfectly separated), but can also be observed in the smoothed curve in the example: the rightmost bin of the histogram has a positive-to-negative ratio of 5, while the next bin has a ratio of 13. Consequently, the two leftmost segments of the ROC curve display a slight concavity. It means that performance can be improved by combining the two bins, leading to one large segment with slope 9. In other words, ROC curve concavities demonstrate locally suboptimal behavior of a classifier. An extreme case of suboptimal behavior occurs if the entire curve is concave, or at least below the ascending diagonal: in that case, performance can simply be improved by assigning all test instances the same score, resulting in an ROC curve that follows the ascending diagonal. A *convex* ROC curve is one without concavities.

The AUC Statistic

The most important statistic associated with ROC curves is the *area under* (ROC) *curve* or AUC. Since the curve is located in the unit square, we have $0 \leq \mathrm{AUC} \leq 1$. $\mathrm{AUC} = 1$ is achieved if the classifier scores every positive higher than every negative; $\mathrm{AUC} = 0$ is achieved if every negative is scored higher than every positive. $\mathrm{AUC} = 1/2$ is obtained in a range of different scenarios, including: (1) the classifier assigns the same score to all test examples, whether positive or negative, and thus the ROC curve is the ascending diagonal; (2) the per-class score distributions are similar, which results in an ROC curve close (but not identical) to the ascending diagonal; and (3) the classifier gives half of a particular class the highest scores, and the other half, the lowest scores. Note that, although a classifier with AUC close to 1/2 is often said to perform randomly, there is nothing random in the third classifier: rather, its excellent performance on some of the examples is counterbalanced by its very poor performance on some others. (Sometimes a linear rescaling $2\mathrm{AUC}-1$ called the Gini coefficient is preferred, which has a related use in the assessment of income or wealth distributions using Lorenz curves: a Gini coefficient close to 0 means that income is approximately evenly distributed. Notice that this Gini coefficient is often called the Gini index, but should not be confused with the impurity measure used in decision tree learning).

AUC has a very useful statistical interpretation: it is the expectation that a (uniformly) randomly drawn positive receives a higher score than a randomly drawn negative. It is a normalized version of the *Wilcoxon–Mann–Whitney sum of ranks test*, which tests the null hypothesis that two samples of ordinal measurements are drawn from a single distribution. The "sum of ranks" epithet refers to one method to compute this statistic, which is to assign each test example an integer rank according to decreasing score (the highest scoring example gets rank 1, the next gets rank 2, etc.); sum up the ranks of the n^- negatives, which have to be high; and subtract $\sum_{i=1}^{n^-} i = n^-(n^- + 1)/2$ to achieve 0 if all negatives are ranked first. The AUC statistic is then obtained by normalizing by the number of pairs of one positive and one negative, n^+n^-. There are several other ways to calculate AUC: for instance, we can calculate, for each negative, the number of positives preceding it, which basically is a columnwise calculation and yields an alternative view of AUC as the expected true positive rate if the operating point is chosen just before a randomly drawn negative.

Identifying Optimal Points and the ROC Convex Hull

In order to select an operating point on an ROC curve, we first need to specify the objective function that we aim to optimize. In the simplest case this will be ►accuracy, the proportion of correctly predicted examples. Denoting the proportion of positives by pos, we can express accuracy as a weighted average of the true positive and true negative rates $\text{pos} \cdot \text{tpr} + (1 - \text{pos})(1 - \text{fpr})$. It follows that points with the same accuracy lie on a straight line with slope $(1 - \text{pos})/\text{pos}$; these parallel lines are the *isometrics* for accuracy (Flach, 2003). In order to find the optimal operating point for a given class distribution, we can start with an accuracy isometric through $(0,1)$ and slide it down until it touches the ROC curve in one or more points (Fig. 3 *(left)*). In the case of a single point this uniquely determines the operating point and thus, the threshold. If there are several points in common between the accuracy isometric and the ROC curve, we can make an arbitrary choice, or interpolate stochastically. We can read off the achieved accuracy by intersecting the accuracy isometric with the descending diagonal, on which $\text{tpr} = 1 - \text{fpr}$ and therefore the true positive rate at the intersection point is equal to the accuracy associated with the isometric.

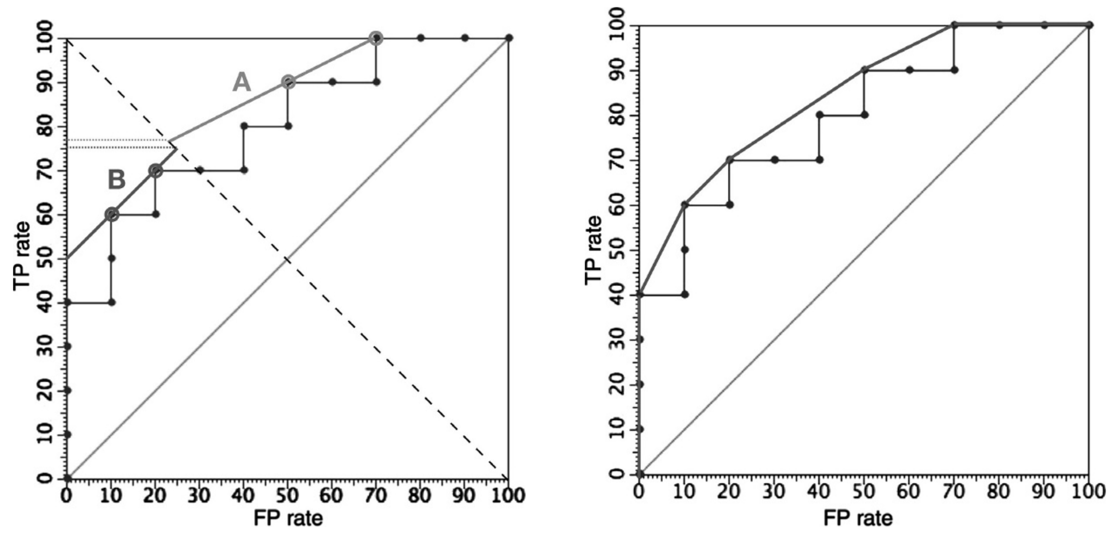
We can generalize this approach to any objective function that is a linear combination of true and false positive rates. For instance, let predicting class i for an instance of class j incur cost $\text{cost}(i|j)$, so for instance the cost of a false positive is $\text{cost}(+|-)$ (profits for correct predictions are modeled as negative costs, e.g., $\text{cost}(+|+) < 0$). Cost isometrics then have slope $(\text{cost}(+|-) - \text{cost}(-|-))/(\text{cost}(-|+) - \text{cost}(+|+))$. Nonuniform class distributions are simply taken into account by multiplying the class and cost ratio, giving a single *skew ratio* expressing the relative importance of negatives compared to positives.

This procedure of selecting an optimal point on an ROC curve can be generalized to select among points lying on more than one curve, or even an arbitrary set of points (e.g., points representing different categorical classifiers). In such scenarios, it is likely that certain points are never selected for any skew ratio; such points are said to be *dominated*. For instance, points on a concave region of an ROC curve are dominated. The non-dominated points are optimal for a given closed interval of skew ratios, and can be joined to form the *convex hull* of the given ROC curve or set of ROC points (Fig. 3 *(right)*). (In multiobjective optimization, this concept is called the *Pareto front*.) This notion of the ROC convex hull (sometimes abbreviated as ROCCH) is extremely useful in a range of situations. For instance, if an ROC curve displays concavities, the convex hull represents a discretization of the scores which achieves higher AUC. Alternatively, the convex hull of a set of categorical classifiers can be interpreted as a hybrid classifier that can reach any point on the convex hull by stochastic interpolation between two neighboring classifiers (Provost & Fawcett, 2001).

Obtaining Calibrated Estimates of the Class Posterior

Recall that each segment of an ROC curve has slope $\text{slope}(s) = f(s|+)/f(s|-)$, where s is the score associated with the segment, and $f(s|+)$ and $f(s|-)$ are the relative frequencies of positives and negatives of assigned score s. Now consider the function $\text{cal}(s) = \text{slope}(s)/(\text{slope}(s) + 1) = f(s|+)/(f(s|+) + f(s|-))$: the *calibration map* $s \mapsto \text{cal}(s)$ adjusts the classifier's scores to reflect the empirical probabilities observed in the test set. If the ROC curve is convex, $\text{slope}(s)$ and $\text{cal}(s)$ are monotonically nonincreasing with decreasing s, and thus replacing the scores s with $\text{cal}(s)$ does not change the ROC curve (other than merging neighboring segments with different scores but the same slope into a single segment).

Consider decision trees as a concrete example. Once we have trained (and possibly pruned) a tree, we can obtain a score in each leaf l by taking the ratio of positive to negative training examples in that leaf: $\text{score}(l) = p(+|l)/p(-|l)$. These scores represent posterior odds, taking into account the class prior in the training set. Each leaf gives rise to a different segment of the ROC curve, which, by the nature of how the scores were calculated, will be convex. The calibrated scores $\text{cal}(\text{score}(l))$ then represent an adjusted estimate of the positive posterior that replaces the training set prior with a uniform prior. To see this, notice that duplicating

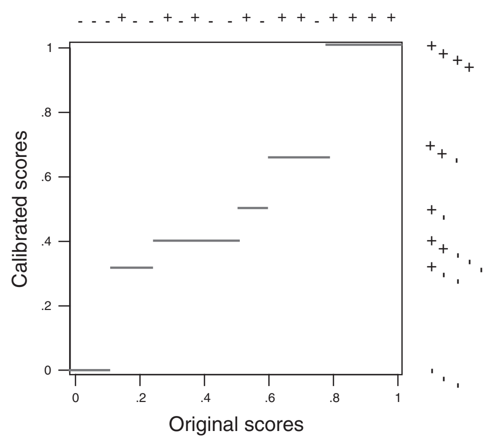


ROC Analysis. Figure 3. (*left*) **The slope of accuracy isometrics reflects the class ratio. Isometric A has slope 1/2: this corresponds to having twice as many positives as negatives, meaning that an increase in true positive rate of *x* is worth a 2*x* increase in false positive rate. This selects two optimal points on the ROC curve. Isometric B corresponds to a uniform class distribution, and selects optimal points which make fewer positive predictions. In either case, the achieved accuracy can be read off on the *y*-axis after intersecting the isometric with the descending diagonal (slightly higher for points selected by A). (*right*) The convex hull selects those points on an ROC curve which are optimal under some class distribution. The slope of each segment of the convex hull gives the class ratio under which the two end points of the segment yield equal accuracy. All points under the convex hull are nonoptimal**

all positive training examples would amplify all uncalibrated scores score(l) with a factor 2, but the ROC curve and therefore the calibrated probability estimates remain unchanged.

If the ROC curve is not convex, the mapping $s \mapsto \mathrm{cal}(s)$ is not monotonic; while the scores $\mathrm{cal}(s)$ would lead to improved performance on the data from which the ROC curve was derived, this is very unlikely to generalize to other data, and thus leads to ►overfitting. This is why, in practice, a less drastic calibration procedure involving the convex hull is applied (Fawcett & Niculescu-Mizil, 2007). Let s_1 and s_2 be the scores associated with the start and end segments of a concavity, i.e., $s_1 > s_2$ and $\mathrm{slope}(s_1) < \mathrm{slope}(s_2)$. Let $\mathrm{slope}(s_1 s_2)$ denote the slope of the line segment of the convex hull that repairs this concavity, which implies $\mathrm{slope}(s_1) < \mathrm{slope}(s_1 s_2) < \mathrm{slope}(s_2)$. The calibration map will then map any score in the interval $[s_1, s_2]$ to $\mathrm{slope}(s_1 s_2)/(\mathrm{slope}(s_1 s_2) + 1)$ (Fig. 4).

This ROC-based calibration procedure, which is also known as *isotonic regression* (Zadrozny & Elkan,



ROC Analysis. Figure 4. The piecewise constant calibration map derived from the convex hull in Fig. 3. The original score distributions are indicated at the top of the figure, and the calibrated distributions are on the right. We can clearly see the combined effect of binning the scores and redistributing them over the interval $[0,1]$

2002), not only produces calibrated probability estimates but also improves AUC. This is in contrast with other calibration procedures such as logistic calibration which do not bin the scores and therefore do not change the ROC curve. ROC-based calibration can be shown to achieve lowest *Brier score* (Brier, 1950), which measures the mean squared error in the probability estimates as compared with the ideal probabilities (1 for a positive and 0 for a negative), among all probability estimators that do not reverse pairwise rankings. On the other hand, being a nonparametric method it typically requires more data than parametric methods in order to estimate the bin boundaries reliably.

Future Directions

ROC analysis in its original form is restricted to binary classification, and its extension to more than two classes gives rise to many open problems. c-class ROC analysis requires $c(c-1)$ dimensions, in order to distinguish each possible misclassification type. Srinivasan proved that basic concepts such as the ROC polytope and its linearly interpolated convex hull generalize to the c-class case (Srinivasan, 1999). In theory, the volume under the ROC polytope can be employed for assessing the quality of a multiclass classifier (Ferri, Hernández-Orallo, & Salido, 2003), but this volume is hard to compute as – unlike the two-class case, where the segments of an ROC curve can simply be enumerated in $O(n \log n)$ time by sorting the n examples on their score (Fawcett, 2006; Flach, 2004) – there is no simple way to enumerate the ROC polytope. Mossman considers the special case of 3-class ROC analysis, where for each class the two possible misclassifications are treated equally (the so-called *one-versus-rest* scenario) (Mossman, 1999). Hand and Till propose the average of all one-versus-rest AUCs as an approximation of the area under the ROC polytope (Hand & Till, 2001). Various algorithms for minimizing a classifier's misclassification costs by reweighting the classes are considered in Bourke, Deng, Scott, Schapire, and Vinodchandran (2008) and Lachiche and Flach (2003).

Other research directions include the explicit visualization of misclassification costs (Drummond & Holte, 2006), and using ROC analysis to study the behavior of machine learning algorithms and the relations between machine learning metrics (Fuernkranz & Flach, 2005).

Cross References

- ►Accuracy
- ►Class Imbalance Problem
- ►Classification
- ►Confusion Matrix
- ►Cost-Sensitive Learning
- ►Error Rate
- ►False Negative
- ►False Positive
- ►Gaussian Distribution
- ►Posterior Probability
- ►Precision
- ►Prior Probability
- ►Recall
- ►Sensitivity
- ►Specificity
- ►True Negative
- ►True Positive

Recommended Reading

Bourke, C., Deng, K., Scott, S., Schapire, R., & Vinodchandran, N. V. (2008). On reoptimizing multi-class classifiers. *Machine Learning, 71*(2–3), 219–242.

Brier, G. (1950). Verification of forecasts expressed in terms of probabilities. *Monthly Weather Review, 78*, 1–3.

Drummond, C., & Holte, R. (2006). Cost curves: An improved method for visualizing classifier performance. *Machine Learning, 65*(1), 95–130.

Egan, J. (1975). *Signal detection theory and ROC analysis. Series in cognitition and perception*. New York: Academic Press.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874.

Fawcett, T., & Niculescu-Mizil, A. (2007). PAV and the ROC convex hull. *Machine Learning, 68*(1), 97–106.

Ferri, C., Hernández-Orallo, J., & Salido, M. (2003). Volume under the ROC surface for multi-class problems. In *Proceedings of the fourteenth (ECML 2003)* (pp. 108–120). Lecture Notes in Computer Science 2837. Berlin: Springer.

Flach, P. (2003). The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *Proceedings of the twentieth international conference on machine learning (ICML 2003)* (pp. 194–201). Washington, DC: AAAI Press.

Flach, P. (2004). *The many faces of ROC analysis in machine learning*. ICML-04 Tutorial. http://www.cs.bris.ac.uk/flach/ICML04tutorial/. Accessed on 16 December 2009.

Fuernkranz, J., & Flach, P. (2005). ROC 'n' Rule learning – towards a better understanding of covering algorithms. *Machine Learning, 58*(1), 39–77.

Hand, D., & Till, R. (2001). A simple generalization of the area under the ROC curve to multiple class classification problems. *Machine Learning, 45*(2), 171–186.

Lachiche, N., & Flach, P. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In *Proceedings of the twentieth international conference on machine learning (ICML'03)* (pp. 416–423). Washington, DC: AAAI Press.

Mossman, D. (1999). Three-way ROCs. *Medical Decision Making, 19*, 78–89.

Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning, 42*(3), 203–231.

Srinivasan, A. (1999). *Note on the location of optimal classifiers in n-dimensional ROC space.* Technical report PRG-TR-2-99. Oxford University Computing Laboratory, Oxford.

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 694–699). New York: ACM.

ROC Convex Hull

The convex hull of an ►ROC curve is a geometric construction that selects the points on the curve that are optimal under some class and cost distribution. It is analogous to the Pareto front in multiobjective optimization. See ►ROC Analysis.

ROC Curve

The ROC curve is a plot depicting the trade-off between the ►true positive rate and the ►false positive rate for a classifier under varying decision thresholds. See ►ROC Analysis.

Rotation Forests

Rotation Forests is an ►ensemble learning technique. It is similar to the ►Random Forests approach to building decision tree ensembles. In the first step, the original feature set is split randomly into K disjoint subsets. Next, ►principal components analysis is used to extract n principal component dimensions from each of the K subsets. These are then pooled, and the original data projected linearly into this new feature space. A tree is then built from this data in the usual manner. This process is repeated to create an ensemble of trees, each time with a different random split of the original feature set.

As the tree learning algorithm builds the classification regions using hyperplanes parallel to the feature axes, a small rotation of the axes may lead to a very different tree. The effect of rotating the axes is that classification regions of high accuracy can be constructed with far fewer trees than in ►Bagging and ►Adaboost.

RSM

►Random Subspace Method

Rule Learning

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Synonyms

AQ; Covering algorithm; CN2; Foil; Laplace estimate; *m*-estimate; OPUS; RIPPER

Definition

Inductive rule learning solves a ►classification problem via the induction of a ►rule set or a ►decision list. The principal approach is the so-called *separate-and-conquer* or *covering* algorithm, which learns one rule at a time, successively removing the covered examples. Individual algorithms within this framework differ primarily in the way they learn single rules. A more extensive survey of this family of algorithms can be found in Fürnkranz (1999).

The Covering Algorithm

Most covering algorithms operate in a ►concept learning framework, that is, they assume a set of positive and negative training examples. Adaptations to the multi-class case are typically performed via ►class binarization, transforming the original problem into a set of binary problems. Some algorithms, most notably CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991), learn multi-class rules directly by optimizing over all possible classes in the head of the rule. In this case, the resulting theory is interpreted as a decision list. In the following, a two-class problem with a positive and a negative class will be assumed.

```
procedure COVERING (Examples, Classifier)
Input: Examples, a set of positive and negative
        examples for a class c.
// initialize the classifier
Classifier = ∅
// loop until no more positive examples are covered
while POSITIVE (Examples) ≠ ∅ do
    // find the best rule for the current examples
    Rule = FINDBESTRULE (Examples)
    // check if we need more rules
    if RULESTOPPINGCRITERION (Classifier,
                              Rule, Examples)
    then    break while
    // remove covered examples and add rule to rule set
    Examples = Examples ∖ COVER (Rule, Examples)
    Classifier = Classifier ∪ Rule
endwhile
// post-process the rule set (e.g., pruning)
Classifier = POSTPROCESSING (Classifier)
Output: Classifier
```

```
procedure FINDBESTRULE (Examples, BestRule)
Input: Examples, a set of positive and negative
        examples for a class c.
InitRule = INITIALIZERULE (Examples)
InitVal = EVALUATERULE (InitRule)
BestRule = <InitVal, InitRule>
Rules = {BestRule}
while Rules ≠ ∅ do
  Candidates = SELECTCANDIDATES(Rules, Examples)
  Rules = Rules ∖ Candidates
  for Candidate ∈ Candidates do
    Refinements = REFINERULE(Candidate, Examples)
      for Refinement ∈ Refinements do
        Evaluation = EVALUATERULE (Refinement,
                                   Examples)
          if STOPPINGCRITERION(Refinement,
                Examples)
          then    next Refinement
          NewRule = <Evaluation, Refinement>
          Rules = INSERTSORT(NewRule, Rules)
          if NewRule > BestRule
          then    BestRule = NewRule
      endfor
  endfor
  Rules = FILTERRULES(Rules, Examples)
endwhile
Output: BestRule
```

The COVERING algorithm starts with an empty theory. If there are any positive examples in the training set it calls the subroutine FINDBESTRULE for learning a single rule that will cover a subset of the positive examples (and possibly some negative examples as well). All covered examples are then separated from the training set, the learned rule is added to the theory, and another rule is learned from the remaining examples. Rules are learned in this way until no positive examples are left or until the RULESTOPPINGCRITERION fires. In the simplest case, the stopping criterion is a check whether there are still remaining positive examples that need to be covered. The resulting theory may also undergo some POSTPROCESSING, for example, a separate pruning and re-induction phase as in RIPPER (Cohen, 1995).

In the following, these components will be discussed in more detail.

Finding the Best Rule

Single rules are typically found by searching the space of possible rules for a rule that optimizes a given quality criterion defined in EVALUATERULE. The value of this heuristic function is the higher the more positive and the less negative examples are covered by the candidate rule. FINDBESTRULE maintains *Rules*, a sorted list of candidate rules, which is initialized by the procedure INITIALIZERULE. New rules will be inserted in appropriate places (INSERTSORT), so that *Rules* will always be sorted in decreasing order of the heuristic evaluations of the rules. At each cycle, SELECTCANDIDATES selects a subset of these candidate rules, which are then refined using the ▸refinement operator REFINERULE. Each refinement is evaluated and inserted into the sorted *Rules* list unless the STOPPINGCRITERION prevents this. If the evaluation of the *NewRule* is better than the best rule found previously, *BestRule* is set to *NewRule*. FILTERRULES selects the subset of the ordered rule list that will be used in subsequent iterations. When all candidate rules have been processed, the best rule is returned.

Different choices of these functions allow the definition of different biases for the separate-and-conquer learner. The ▶search bias is defined by the choice of a search strategy (InitializeRule and RefineRule), a search algorithm (SelectCandidates and FilterRules), and a search heuristic (EvaluateRule). The refinement operator RefineRule constitutes the ▶language bias of the algorithm. An overfitting avoidance bias can be implemented via some StoppingCriterion and/or in a post-processing phase.

For example, InitializeRule and RefineRule may be defined so that they realize a top-down (general-to-specific), a bottom-up (specific-to-general) or a bidirectional search. Exhaustive ▶breadth-first, ▶depth-first, or best-first searches can be realized by appropriate choices of EvaluateRule, and no filtering or candidate selection. FilterRules can, for example, be used to realize a ▶hill-climbing or ▶beam search by maintaining only the best or the *BeamWidth* best rules. Evolutionary algorithms and stochastic local search can also be easily realized.

The most common algorithm for finding the best rule is a top-down hill-climbing algorithm. It basically constructs a rule by consecutively adding conditions to the rule body so that a given quality criterion is greedily optimized. This constitutes a simple greedy hill-climbing algorithm for finding a local optimum in the hypothesis space defined by the feature set. InitializeRule will thus return the most general rule, the rule with the body {`true`}, and RefineRule will return all possible extensions of the rule by a single condition. FilterRules will only let the best refinement pass for the next iteration, so that SelectCandidates will always have only one choice. The search heuristic, the stopping criterion, and the post-processing are discussed in the next sections.

Rule Learning Heuristics

The covering algorithm tries to find a rule set that is as *complete* and *consistent* as possible. Thus, each rule should cover as many positive examples and as few negative examples as possible. The exact trade-off between these two objectives is realized via the choice of a rule learning heuristic. A few important ones are (assume that p out of P positive examples and n out of N negative examples are covered by the rule):

Laplace estimate $\left(\text{Lap} = \frac{p+1}{p+n+2}\right)$ computes the fraction of positive examples in all covered examples, where each class is initialized with one virtual example in order to penalize rules with low coverage.

m-**estimate** $\left(\text{m} = \frac{p+m\cdot P/(P+N)}{p+n+m}\right)$ is a generalization of the Laplace estimate which uses m examples for initialization, which are distributed according to the class distribution in the training set (Cestnik, 1990).

Information gain $\left(\text{ig} = p \cdot \left(\log_2 \frac{p}{p+n} - \log_2 \frac{p'}{p'+n'}\right)\right)$, where p' and n' are the number of positive and negative examples covered by the rule's predecessor) is Quinlan's (1990) adaptation of the information gain heuristic used for decision tree learning. The main difference is that this only focuses on a single branch (a rule), whereas the decision tree version tries to optimize all successors of a node simultaneously.

Correlation and χ^2 $\left(\text{corr} = \frac{p(N-n)-(P-p)n}{\sqrt{PN(p+n)(P-p+N-n)}}\right)$ computes the four-field correlation of covered/uncovered positive/negative examples. It is equivalent to a χ^2 statistic ($\chi^2 = (P+N)\,\text{corr}^2$).

An exhaustive overview and theoretical comparison of various search heuristics in coverage space, a variant of ▶ROC space can be found in Fürnkranz and Flach (2005).

Overfitting Avoidance

It is trivial to find a rule set that is complete and consistent on the training data. To achieve this, one only needs to convert each positive example into a rule. Each of these rules is consistent (provided the data set is not inconsistent), and collectively they cover the entire example set (completeness). However, this is clearly a bad case of ▶overfitting because the theory will not generalize to new positive examples.

Overfitting is to some extent handled by the search heuristics described above, but most algorithms use additional ▶pruning techniques. One can discriminate between ▶*pre-pruning* techniques, where a separate criterion is used to filter out unpromising rules. For example, CN2 computes the *likelihood ratio statistic* $\text{lrs} = 2 \cdot \left(p \log \frac{p}{e_p} + n \log \frac{n}{e_n}\right)$, where $e_p = (p+n)\frac{P}{P+N}$ and $e_n = (p+n)\frac{N}{P+N} = (p+n) - e_p$ are the number of positive and negative examples one could expect if the $p+n$ examples covered by the rule were distributed in the same way as the $P + N$ examples in the full data set. This statistic follows a χ^2 distribution, which allows to filter out rules for which the distribution of the covered examples is not statistically and significantly different from the distribution of examples in the full data set. Other pre-pruning criteria are simple thresholds that define a minimum acceptable value for the search heuristic, or FOIL's ▶minimum description length criterion that relates the length of a rule to the number of examples it covers.

However, it can be shown experimentally that CN2 or FOIL still have a tendency to overfit the data. Instead, state-of-the-art algorithms ▶post-prune a rule right after it has been learned. For this purpose, one third of the training data are reserved for pruning. After a rule has been learned, it is greedily simplified on the pruning set. Simplifications can be the deletion of the last condition, a final sequence of conditions, or an arbitrary condition of the rule. If the simplification does not decrease the accuracy of the rule on the pruning set, it will be performed. This so-called *incremental reduced error pruning* algorithm (Fürnkranz & Widmer, 1994) is used in the rule learning algorithm RIPPER (Cohen, 1995).

A survey and experimental comparison of pruning techniques for rule learning can be found in Fürnkranz (1997).

Alternatives to Covering

An obvious generalization of covering is to not entirely remove covered examples but to reduce their example ▶weights, thus decreasing their importance in subsequent iterations (see, e.g., the SLIPPER algorithm; Cohen & Singer 1999).

Rules can also be learned by alternative strategies. There have been numerous proposals, only the most influential can be mentioned. Each path from the root to a leaf of a ▶decision tree corresponds to a rule and so rules can be learned by first learning a decision tree and then post-processing it (see, e.g., the C4.5RULES algorithm; Quinlan, 1993). It is also possible to use the ▶APriori algorithm for an exhaustive search for classification rules, and to use a subsequent covering algorithm to combine the rules into a rule set (see, e.g., the CBA algorithm; Liu, Hsu, & Ma, 1998). RISE (Domingos, 1996) combines bottom-up generalization with ▶nearest neighbor algorithms to learn a theory via "conquering without separating".

Well-known Rule Learning Algorithms

AQ can be considered as the original covering algorithm. Its original version was conceived by Ryszard Michalski in the 1960s (Michalski, 1969), and numerous versions and variants of the algorithm appeared subsequently in the literature. AQ uses a top-down beam search for finding the best rule. It does not search all possible specializations of a rule, but only considers refinements that cover a particular example, the so-called *seed example*. This idea is basically the same as the use of a ▶bottom clause in ▶inductive logic programming.

CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991) employs a beam search guided by the Laplace estimate, and uses the likelihood ratio significance test to fight overfitting. It can operate in two modes, one for learning rule sets (by modeling each class independently), and one for learning decision lists.

FOIL (Quinlan, 1990) was the first relational learning algorithm that received attention beyond the field of inductive logic programming. It learns a concept with the covering loop and learns individual concepts with a top-down refinement operator, guided by information gain. The main difference to previous systems is that FOIL allowed the use of first-order background knowledge. Instead of only being able to use tests on single attributes, FOIL could employ tests that compute relations between multiple attributes, as well as introduce new variables in the body of a rule.

RIPPER (Cohen, 1995) was the first rule learning system that effectively countered the overfitting problem via *incremental reduced error pruning*, as described above. It also added a post-processing phase for optimizing a rule set in the context of other rules. The key idea is to remove one rule out of a previously learned rule set and try to relearn it not only in the context of previous rules (as would be the case in the regular covering rule), but in the context of a complete theory. RIPPER is still state-of-the-art in inductive rule learning. A freely accessible reimplementation can be found in the WEKA machine learning library under the name of JRIP.

Opus (Webb, 1995) was the first rule learning algorithm to demonstrate the feasibility of a full exhaustive search through all possible rule bodies for finding a rule that maximizes a given quality criterion (or heuristic function). The key idea is the use of *ordered search* that prevents that a rule is generated multiple times. This means that even though there are $l!$ different orders of the conditions of a rule of length l, only one of them can be taken by the learner for finding this rule. In addition, OPUS uses several techniques that prune significant parts of the search space, so that this search method becomes feasible. Follow-up work has shown that this technique is also an efficient alternative for ►association rule discovery, provided that the database to mine fits into the memory of the learning system.

Cross References

►Apriori Algorithm
►Association Rule
►Decision List
►Decision Trees
►Subgroup Discovery

Recommended Reading

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. In L. Aiello (Ed.), *Proceedings of the ninth European conference on artificial intelligence (ECAI-90), Stockholm, Sweden* (pp. 147–150). Pitman, London.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proceedings of the fifth European working session on learning (EWSL-91), Porto, Portugal* (pp. 151–163). London: Springer.

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

Cohen, W. W. (1995). Fast effective rule induction. In A. Prieditis & S. Russell (Eds.), *Proceedings of the 12th international conference on machine learning (ML-95), Lake Tahoe, California* (pp. 115–123). Morgan Kaufmann, San Mateo, CA.

Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. In *Proceedings of the 16th national conference on artificial intelligence (AAAI-99), Orlando* (pp. 335–342). Menlo Park: AAAI/MIT Press.

Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning, 24*, 141–168.

Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning, 27*(2), 139–171.

Fürnkranz, J. (February 1999). Separate-and-conquer rule learning. *Artificial Intelligence Review, 13*(1), 3–54.

Fürnkranz, J., & Flach, P. (2005). ROC 'n' rule learning – Towards a better understanding of covering algorithms. *Machine Learning, 58*(1), 39–77.

Fürnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. In W. Cohen & H. Hirsh (Eds.), *Proceedings of the 11th international conference on machine learning (ML-94), New Brunswick, NJ* (pp. 70–77). Morgan Kaufmann, San Mateo, CA.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In R. Agrawal, P. Stolorz, & G. Piatetsky-Shapiro (Eds.), *Proceedings of the fourth international conference on knowledge discovery and data mining (KDD-98), New York City, NY* (pp. 80–86).

Michalski, R. S. (1969). On the quasi-minimal solution of the covering problem. In *Proceedings of the fifth international symposium on information processing (FCIP-69), Bled, Yugoslavia. Switching circuits* (Vol. A3, pp. 125–128).

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

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

Webb, G. I. (1995). OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research, 5*, 431–465.