G

Gaussian Distribution

Xinhua Zhang
NICTA, Australian National University, Canberra, ACT, Australia
School of Computer Science, Australian National University, Canberra, ACT, Australia
NICTA London Circuit, Canberra, ACT, Australia

Abstract
Gaussian distributions are one of the most important distributions in statistics. It is a continuous probability distribution that approximately describes some mass of objects that concentrate about their mean. The probability density function is bell shaped, peaking at the mean. Its popularity also arises partly from the central limit theorem, which says the average of a large number of independent and identically distributed random variables is approximately Gaussian distributed. Moreover, under some reasonable conditions, posterior distributions become approximately Gaussian in the large data limit. Therefore, the Gaussian distribution has been used as a simple model for many theoretical and practical problems in statistics, natural science, and social science.

Synonyms

Normal distribution

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

Definition

The simplest form of Gaussian distribution is the one-dimensional standard Gaussian distribution, which can be described by the probability density function (*pdf*):

$$p(x) = \phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

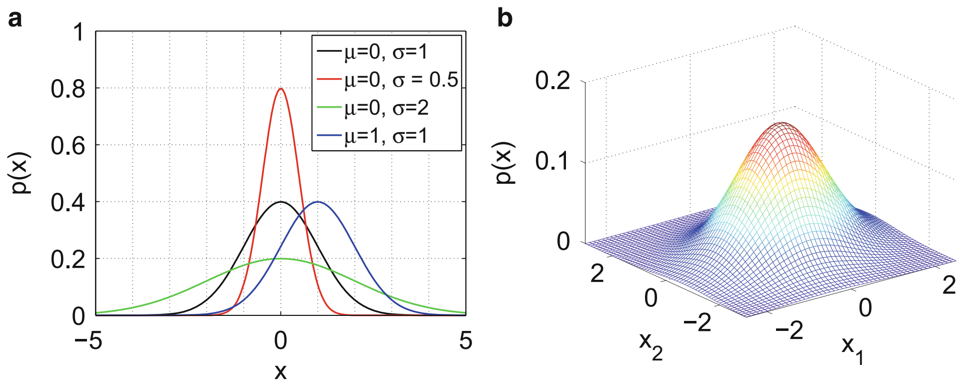
where $\frac{1}{\sqrt{2\pi}}$ ensures the normalization, i.e., $\int_{\mathbb{R}} p(x)\mathrm{d}x = 1$. This distribution centers around $x = 0$, and the rate of decay or "width" of the curve is 1.

More generally, we can apply translation and scaling to obtain a Gaussian distribution that centers on arbitrary $\mu \in \mathbb{R}$ and with arbitrary width $\sigma > 0$. The *pdf* is

$$\begin{aligned} p(x) &= \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \end{aligned}$$

Technically, μ is called the mean and σ^2 is called the variance. Obviously, μ is the peak/mode of the density and is also the mean and median of the distribution due to the symmetry of the density around μ. If a random variable X has this density, then we write

$$X \sim \mathcal{N}(\mu, \sigma^2).$$



Gaussian Distribution, Fig. 1 Gaussian probability density functions. (**a**) One dimension. (**b**) Two dimension

Example density functions are plotted in Fig. 1a.

As an extension to multivariate random variables, the multivariate Gaussian distribution is a distribution on d-dimensional column vector $\mathbf{x}$ with mean column vector $\boldsymbol{\mu}$ and positive definite variance matrix Σ. This gives

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}\det^{1/2}\Sigma}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

and is denoted by $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. An example *pdf* for the two-dimensional case is plotted in Fig. 1b.

Motivation and Background

In history, Abraham de Moivre first introduced this distribution in 1733 under the name "normal distribution" (of course, he did not call it Gaussian distribution since Gauss had not yet been born). Then Laplace used it to analyze experiment errors, based on which Legendre invented the least squares in 1805. Carl Friedrich Gauss rigorously justified it in 1809 and determined the formula of its probability density function. Finally this distribution is named the Gaussian distribution after Gauss. The name "normal distribution" is also widely used, meaning it is a typical, common, or usual distribution. It was coined by Peirce, Galton, and Lexis around 1875 and made popular by Karl Pearson near the inception of the twentieth century.

Theory/Solution

Canonical Form

The standard definition allows one to easily read off the moments from the *pdf*. Another useful parameterization is called canonical parameterization:

$$p(\mathbf{x}|\boldsymbol{\eta}, \Lambda) = \exp\left(\boldsymbol{\eta}^\top\mathbf{x} - \frac{1}{2}\mathbf{x}^\top\Lambda\mathbf{x} - \frac{1}{2}\left(d\log(2\pi) - \log\det\Lambda + \boldsymbol{\eta}^\top\Lambda\boldsymbol{\eta}\right)\right),$$

where $\eta = \Sigma^{-1}\boldsymbol{\mu}$ and $\Lambda = \Sigma^{-1}$. Λ is often called precision. This parameterization is useful when posing the distribution as a member of the exponential family.

Cumulative Distribution Function

For one-dimensional Gaussian distribution, the cumulative distribution function (*cdf*) is defined by

$$\Phi(x) = \int_{-\infty}^{x}\phi(t)\mathrm{d}t.$$

Formally, it can be conveniently represented by the error function and its complement:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t,$$

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \mathrm{d}t.$$

So

$$\Phi(x) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2}\mathrm{erfc}\left(-\frac{x}{\sqrt{2}}\right).$$

The inverse of the *cdf*, called quantile function, can be written as

$$\Phi^{-1}(s) = \sqrt{2}\mathrm{erf}^{-1}(2s-1), \quad \text{for } s \in (0,1).$$

The *cdf* error function erf() and its inverse $\mathrm{erf}^{-1}()$ do not usually have a closed form and can be computed numerically by functions like `ERF` in Fortran and `double erf(double x)` in C/C++. For the multivariate case, the corresponding *cdf* is highly challenging to compute numerically.

Moments

The first order moment is $\mathbb{E}[X] = \boldsymbol{\mu}$, the variance is $\mathrm{Var}[X] = \Sigma$, and all higher order cumulants are 0. Any central moments with odd terms are 0, i.e., $\mathbb{E}[\Pi_{i=1}^d (x_i - \mu_i)^{p_i}] = 0$ when $\sum_i p_i$ is odd.

Entropy and Kullback-Leibler Divergence

The differential entropy of multivariate Gaussian is

$$h(p) = -\int_{\mathbb{R}^d} p(\mathbf{x}) \ln p(\mathbf{x}) \mathrm{d}\mathbf{x} = \frac{1}{2}\ln\left((2\pi e)^d \det\Sigma\right).$$

The Kullback-Leibler divergence from $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ to $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$ is

$$\mathrm{KL}(\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)||\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)) = \frac{1}{2}\Bigg(\ln\frac{\det\Sigma_2}{\det\Sigma_1} + \mathrm{tr}\Sigma_2^{-1}\Sigma_1 + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \Sigma_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - d\Bigg).$$

Properties Under Affine Transform

Let $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. Suppose A is a linear transform from $\mathbb{R}^d$ to $\mathbb{R}^s$ and $\boldsymbol{c} \in \mathbb{R}^s$, then

$$A\mathbf{x} + \boldsymbol{c} \sim \mathcal{N}(A\boldsymbol{\mu} + \boldsymbol{c}, A\Sigma A^\top)$$

$$\mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})^\top A(\mathbf{x} - \boldsymbol{\mu})] = \mathrm{tr}A\Sigma$$

$$\mathrm{Var}[(\mathbf{x} - \boldsymbol{\mu})^\top A(\mathbf{x} - \boldsymbol{\mu})] = 2\mathrm{tr}A\Sigma A\Sigma$$

where the last two relations require $s = d$.

Conjugate Priors

Conjugate priors where discussed in `<the entry on Prior Probabilities>` (Springer formatters, we want to reference this entry in-line, please format appropriately.). With known variance, the conjugate prior for the mean is again a multivariate Gaussian. With known mean, the conjugate prior for the variance matrix is the Wishart distribution, while the conjugate prior for the precision matrix is the Gamma distribution.

Parameter Estimation

Given n *iid* observations $X_1, \ldots, X_n$, the maximum likelihood estimator of the mean is simply the sample mean

$$\tilde{\boldsymbol{\mu}} = \bar{X} = \frac{1}{n}\sum_{i=1}^n X_i.$$

The maximum likelihood estimator of the covariance matrix is

$$\tilde{\Sigma} = \frac{1}{n}\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^\top.$$

This estimator is biased, and its expectation is $\mathbb{E}[\tilde{\Sigma}] = \frac{n-1}{n}\Sigma$. An unbiased estimator is

$$\tilde{\Sigma} = S = \frac{1}{n-1}\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^\top.$$

Distributions Induced by the Gaussian

If $X \sim \mathcal{N}(0, \Sigma)$, then $X^\top \Sigma^{-1} X$ has a Gamma distribution Gamma$(d/2, 2)$.

Let $X_1, X_2 \sim \mathcal{N}(0,1)$ and they are independent. Their ratio is the standard Cauchy distribution, $X_1/X_2 \sim \text{Cauchy}(0,1)$.

Given n independent univariate random variables $X_i \sim \mathcal{N}(0,1)$, the random variable $Z := \sqrt{\sum_i X_i^2}$ has a χ distribution with degree of freedom n. And Z^2 has a χ^2 distribution with degree of freedom n.

Using Basu's theorem or Cochran's theorem, one can show that the sample mean of $X_1, \ldots, X_n$ and the sample standard deviation are independent. Their ratio

$$t := \frac{\bar{X}}{S} = \frac{\frac{1}{n}(X_1 + \ldots + X_n)}{\sqrt{\frac{1}{n-1}\left[(X_1 - \bar{X})^2 + \ldots + (X_n - \bar{X})^2\right]}}$$

has the student's t-distribution with degree of freedom $n-1$.

Applications

This section discusses some applications and properties of the Gaussian.

Central Limit Theorem

Given n independent and identically distributed observations drawn from a distribution whose variance is finite, the average of the observations is asymptotically Gaussian distributed when n tends to infinity. Under certain conditions, the requirement for identical distribution can be relaxed, and asymptotic normality still holds.

Approximate Gaussian Posterior

Consider n independent and identically distributed observations drawn from a distribution $p(X_i|\boldsymbol{\theta})$, so the data set is $\mathbf{X} = (X_1, \ldots, X_n)^\top$. Under certain conditions, saying roughly that the posterior on $\boldsymbol{\theta}$ converges in probability to a single interior point in its domain as $n \to \infty$, the posterior for $\boldsymbol{\theta}$ is approximately Gaussian for large n, $\boldsymbol{\theta}|\mathbf{X} \approx \mathcal{N}\left(\widehat{\boldsymbol{\theta}}, I\left(\widehat{\boldsymbol{\theta}}\right)\right)$, where $\widehat{\boldsymbol{\theta}}$ is the maximum likelihood or aposterior value for $\boldsymbol{\theta}$ and $I(\boldsymbol{\theta})$ is the *observed (Fisher) information*, the negative of the second derivative (Hessian) of the likelihood w.r.t. the parameters $\boldsymbol{\theta}$.

The Gaussian approximation to the posterior, while a poor approximation in many cases, serves as a useful insight into the nature of asymptotic reasoning. It is justified based on the multidimensional Taylor expansion of the log likelihood at the maximum likelihood or a posterior value, together with its asymptotic convergence property.

3-σ Rule

For standard Gaussian distribution, 99.7 % of the probability mass lie within the three standard deviations $[-3\sigma, 3\sigma]$, i.e., $\int_{-3\sigma}^{3\sigma} \phi(x)\mathrm{d}x > 0.997$. About 95 % mass lies within two standard deviations and about 68 % within one standard deviation. This empirical rule is called 3-σ rule and can be easily extended to general one-dimensional Gaussian distributions.

Combination of Random Variables

Let d-dimensional random variables $X_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$. If they are independent, then for any set of linear transforms A_i from $\mathbb{R}^d$ to $\mathbb{R}^s$, we have $\sum_i A_i X_i \sim \mathcal{N}(\sum_i A_i \boldsymbol{\mu}_i, \sum_i A_i \Sigma_i A_i^\top)$. The converse is also true by the Cramer's theorem: if X_i are independent and their sum $\sum_i X_i$ is Gaussian distributed, then all X_i must be Gaussian.

Correlations and Independence

In general, independent random variables must be uncorrelated but not vice versa. However, if a multivariate random variable is jointly Gaussian, then any uncorrelated subset of the random variables *must be* independent. Notice the precondition of joint Gaussian. It is possible for two Gaussian random variables to be uncorrelated but not independent, for the reason that they are not jointly Gaussian. For example, let $X \sim \mathcal{N}(0,1)$ and $Y = -X$ if $|X| < c$, and $Y = X$ if $|X| > c$. By properly setting c, Y and X can be made uncorrelated but obviously not independent.

Marginalization, Conditioning, and Agglomeration

Suppose the vector $\mathbf{x}$ can be written as $(\mathbf{x}_1^\top, \mathbf{x}_2^\top)^\top$ and correspondingly the mean and covariance matrix can be written as

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Then the marginal distribution of $\mathbf{x}_1$ is Gaussian $\mathcal{N}(\mu_1, \Sigma_{11})$, and the conditional distribution of $\mathbf{x}_1$ conditioned on $\mathbf{x}_2$ is $\mathcal{N}(\boldsymbol{\mu}_{1|2}, \Sigma_{1|2})$, where

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2),$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

Suppose the multivariate Gaussian vector $\mathbf{x}_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ and a vector $\mathbf{x}_2$ is a linear function of $\mathbf{x}_1$ with Gaussian noise, i.e., $\mathbf{x}_2|\mathbf{x}_1 \sim \mathcal{N}(A\mathbf{x}_1 + \boldsymbol{\mu}_{12}, \Sigma_{12})$. Then the joint distribution of $(\mathbf{x}_1^\top, \mathbf{x}_2^\top)^\top$ is also Gaussian:

$$\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_1 \\ A\boldsymbol{\mu}_1 + \boldsymbol{\mu}_{12} \end{pmatrix}, \begin{pmatrix} \Sigma_{11} + A^\top\Sigma_{12}A & -A^\top\Sigma_{12} \\ -\Sigma_{12}A & \Sigma_{12} \end{pmatrix} \right).$$

Cross-References

▸ Gaussian Processes

Recommended Reading

For a complete treatment of Gaussian distributions from a statistical perspective, see Casella and Berger (2002), and Mardia et al. (1979) provides details for the multivariate case. Bernardo and Smith (2000) shows how Gaussian distributions can be used in the Bayesian theory. Bishop (2006) introduces Gaussian distributions in Chap. 2 and shows how it is extensively used in machine learning. Finally, some historical notes on Gaussian distributions can be found at Miller et al., especially under the entries “NORMAL” and “GAUSS.”

Bernardo JM, Smith AFM (2000) Bayesian theory. Wiley, Chichester/New York
Bishop C (2006) Pattern recognition and machine learning. Springer, New York
Casella G, Berger R (2002) Statistical inference, 2nd edn. Duxbury, Pacific Grove
Mardia KV, Kent JT, Bibby JM (1979) Multivariate analysis. Academic Press, London/New York
Miller J, Aldrich J et al. Earliest known uses of some of the words of mathematics. http://jeff560.tripod.com/mathword.html

Gaussian Process

Novi Quadrianto[1], Kristian Kersting[2,3], and Zhao Xu[4]
[1]Department of Informatics, SMiLe CLiNiC, University of Sussex, Brighton, UK
[2]Technische Universität Dortmund, Dortmund, Germany
[3]Knowledge Discovery, Fraunhofer IAIS, Sankt Augustin, Germany
[4]Fraunhofer IAIS, Sankt Augustin, Germany

Synonyms

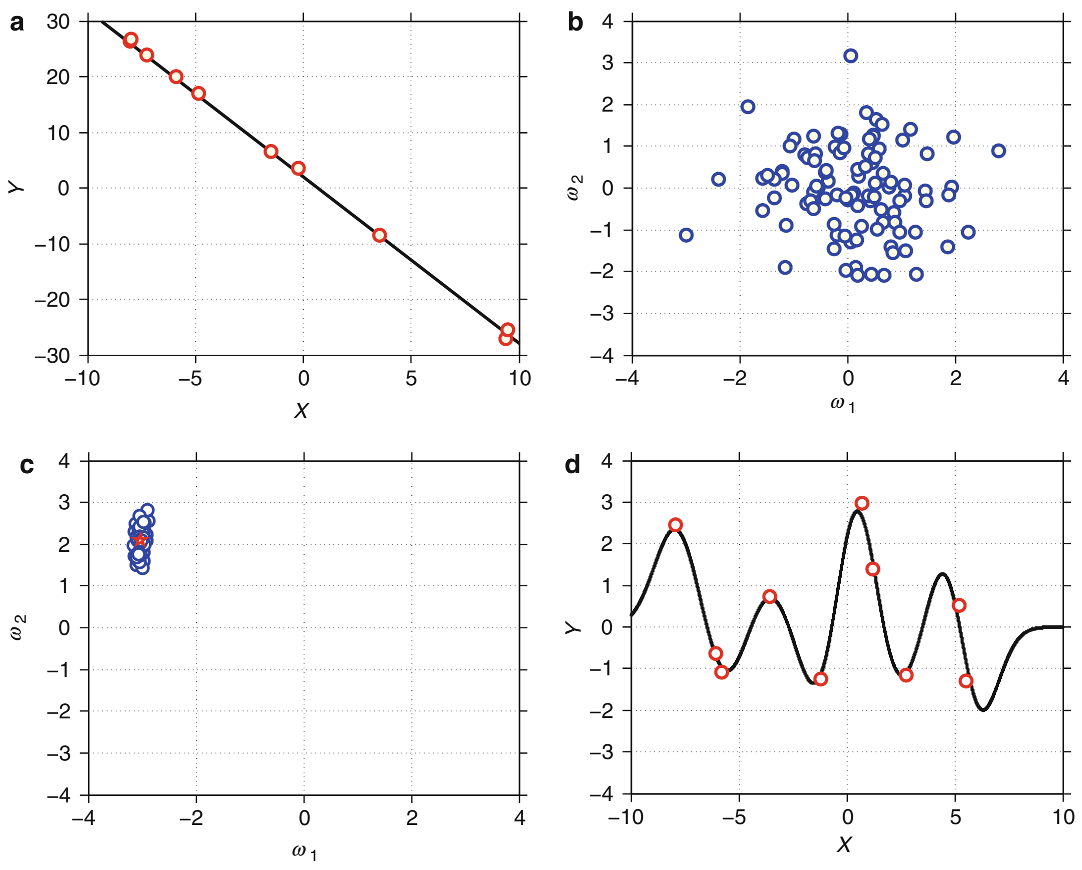
Expectation propagation; Kernels; Laplace estimate; Nonparametric Bayesian

Definition

Gaussian processes generalize multivariate Gaussian distributions over finite-dimensional vectors to infinite dimensionality. Specifically, a Gaussian process is a stochastic process that has Gaussian-distributed finite-dimensional marginal distributions, hence the name. In doing so, it defines a distribution over functions, i.e., each draw from a Gaussian process is a function. Gaussian processes provide a principled, practical, and probabilistic approach to inference and learning in kernel machines.

Motivation and Background

Bayesian probabilistic approaches have many virtues, including their ability to incorporate prior knowledge and their ability to link related



Gaussian Process, Fig. 1 (**a**) Ten observations (one-dimensional input x and output y variables) generated from a ▶ Linear Regression model $y = -3x + 2 + \epsilon$ with Gaussian noise ϵ. The task is to learn the functional relationship between x and y. Assuming the parametric model $y = \omega_1 x + \omega_2 + \epsilon$, i.e., $\omega = (\omega_1, \omega_2)$, is the vector of parameters, and the prior distribution over ω is a two-dimensional Gaussian as shown in (**b**), the posterior distribution over ω can be estimated as shown in (**c**). Its mean $(-2.9716, 1.9981)$ is close to the true parameters $(-3, 2)$. The inference, however, was performed in an ideal situation where in the relationship between x and y was indeed linear. If the true relationship is not known in advances and/or cannot easily be described using a finite set of parameters, this approach may fail. For example, in (**d**), infinite number of parameters might be required to recover the functional relationship

sources of information. Typically, we are given a set of data points sampled from an underlying but unknown distribution, each of which includes input x and output y, such as the ones shown in Fig. 1a. The task is to learn a functional relationship between x and y. Traditionally, in a parametric approach, an assumption on the mathematical form of the relationship such as linear, polynomial, exponential, or a combination of them needs to be chosen a priori. Subsequently, weights (or parameters) are placed on each of the chosen forms, and a prior distribution is then defined over parameters. Thus, the learning task is now reduced to the Bayesian estimation over the parameters, cf. Fig. 1a–c. This approach, however, may not always be practical, as illustrated in Fig. 1d. To discover the latent input–output relationship in Fig. 1d, we might need infinitely many functional forms, and this translates to infinite number of parameters. Instead of working over a parameter space, Gaussian processes place a prior directly on the space of functions without parameterizing the function, hence nonparametric. As will be shown, the computational complexity of inference now scales as the number of data points instead of the number of parameters.

Several nonparametric Bayesian models have been developed for different tasks such as density estimation, regression, classification, survival time analysis, topic modeling, etc. Among the most popular ones are ▶ Dirichlet Processes and *Gaussian processes*. Just as the Gaussian process, a Dirichlet process has Dirichlet-distributed finite-dimensional marginal distributions, hence the name.

Gaussian processes were first formalized for machine-learning tasks by Williams and Rasmussen (1996) and Neal (1996).

Theory

Formally, a Gaussian process is a stochastic process (i.e., a collection of random variables) in which all the finite-dimensional distributions are multivariate Gaussian distributions for any finite choice of variables. In general, Gaussian processes are used to define a probability distribution over functions $f : \mathcal{X} \rightarrow \mathbb{R}$ such that the set of values of f evaluated at an arbitrary set of points $\{x_i\}_{i=1}^{N} \in \mathcal{X}$ will have an N-variate Gaussian distribution. Note that, for $x_i \in \mathbb{R}^2$, this may also be known as a Gaussian random field.

Gaussian Process

A Gaussian distribution is completely specified by its mean and covariance matrix. Similarly, a Gaussian process is characterized by its mean function $m(x) := \mathbf{E}[f(x)]$ and covariance function

$$C(x, x') := \mathbf{E}[(f(x) - m(x))(f(x') - m(x'))] \, .$$

We say a real process $f(x)$ is a Gaussian process distributed with a mean function $m(x)$ and a covariance function $C(x, x')$, written as $f \sim \mathcal{GP}(m(x), C(x, x'))$.

The mean function can be arbitrarily chosen (for convenience, it is often taken to be a zero function since we can always center our observed outputs to have a zero mean), but the covariance function must be a positive-definite function to ensure the existence of all finite-dimensional distributions. That is, the positive definiteness of $C(.,.)$ ensures the positive (semi-)definiteness of all covariance matrices, Σ, appearing in the exponent of the finite-dimensional multivariate Gaussian distribution.

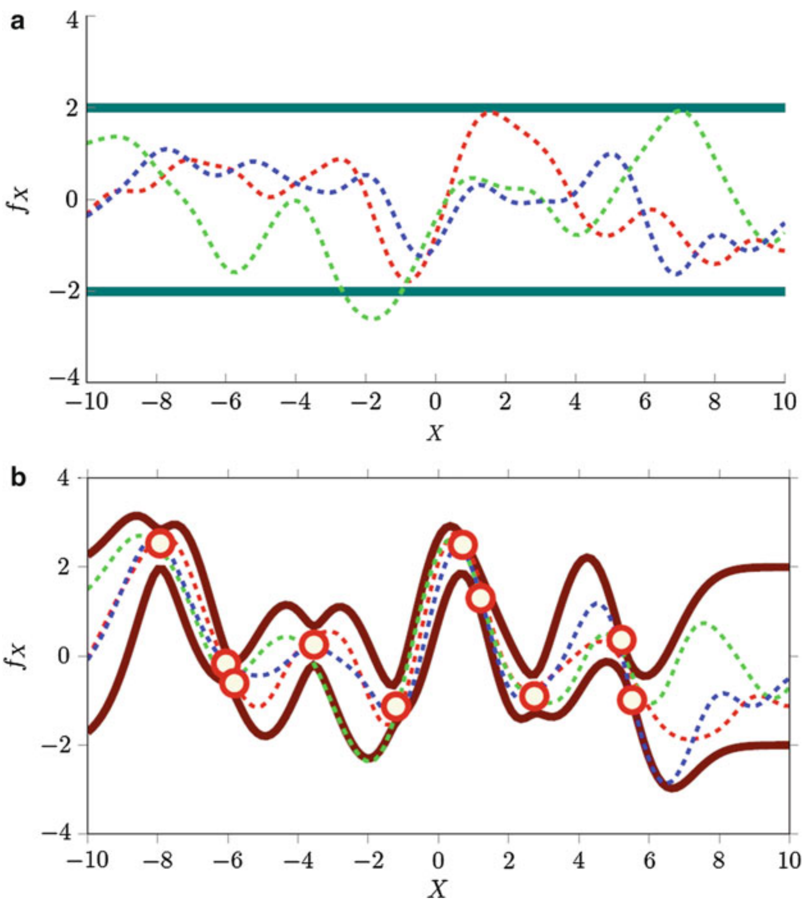
The attractiveness of Gaussian processes is that they admit the marginalization property (▶ Gaussian Distribution), i.e., if the Gaussian process specifies $(f(x_1), f(x_2)) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify $f(x_1) \sim \mathcal{N}(\mu_1, \Sigma_{11})$, where Σ_{11} is the relevant submatrix of Σ. This means addition of novel points will not influence the distribution of existing points. The marginalization property allows us to concentrate on distribution of only observed data points with the rest of unobserved points considered to be marginalized out; thus, a finite amount of computation for inference can be achieved.

Covariance Functions

A covariance function bears an essential role in a Gaussian process model as its continuity properties determine the continuity properties of samples (functions) from the Gaussian process. In the literature, covariance functions are also known as positive (semi-)definite kernels or Mercer's kernels.

There are generally two types of covariance functions: *stationary* and *nonstationary*. A stationary covariance function is a function that is translation invariant, i.e., $C(x, x') = D(x - x')$ for some function D. The typical examples include squared exponential, Matérn class, γ-exponential, exponential, and rational quadratic, while examples of nonstationary covariance functions are dot product and polynomial.

Squared exponential (SE) is a popular form of stationary covariance function, and it corresponds to the class of sums of infinitely many Gaussian-shaped basis functions placed everywhere, $f(x) := \lim_{n\rightarrow\infty} \frac{s}{n} \sum_i^n \gamma_i \exp\left(-((x - x_i)/2\ell)^2\right)$ with $\gamma_i \sim \mathcal{N}(0, 1)\ \forall i$. This covariance function is in the form of



Gaussian Process, Fig. 2 (**a**) Three functions drawn at random from a Gaussian process prior. (**b**) Three random functions drawn from the posterior, i.e., the distribution learned with the prior from Fig. 2a and the ten observations from Fig. 1d. In both plots, the *shaded area* within two *solid lines* shows the pointwise mean plus and minus two times the standard deviation for each input value, i.e., the 95 % confidence region. Animations, if they are visible, are generated using the method described in Hennig (2013)

$$C(x, x') = \mathbf{E}[f(x)f(x')]$$
$$= s^2 \exp\left(-\frac{1}{2\ell^2}\left\|x - x'\right\|_2^2\right).$$

Typical functions sampled from this covariance function can be seen in Fig. 2a. This covariance function has the characteristic length scale ℓ and the signal variance s^2 as free parameters (hyperparameters). The longer the characteristic length scale, the more slowly varying the typical sample function is. The signal variance defines the vertical scale of variations of a sample function. Figure 3 illustrates prediction with SE covariance function with varying characteristic length scale. Several other covariance functions are listed in Table 1.

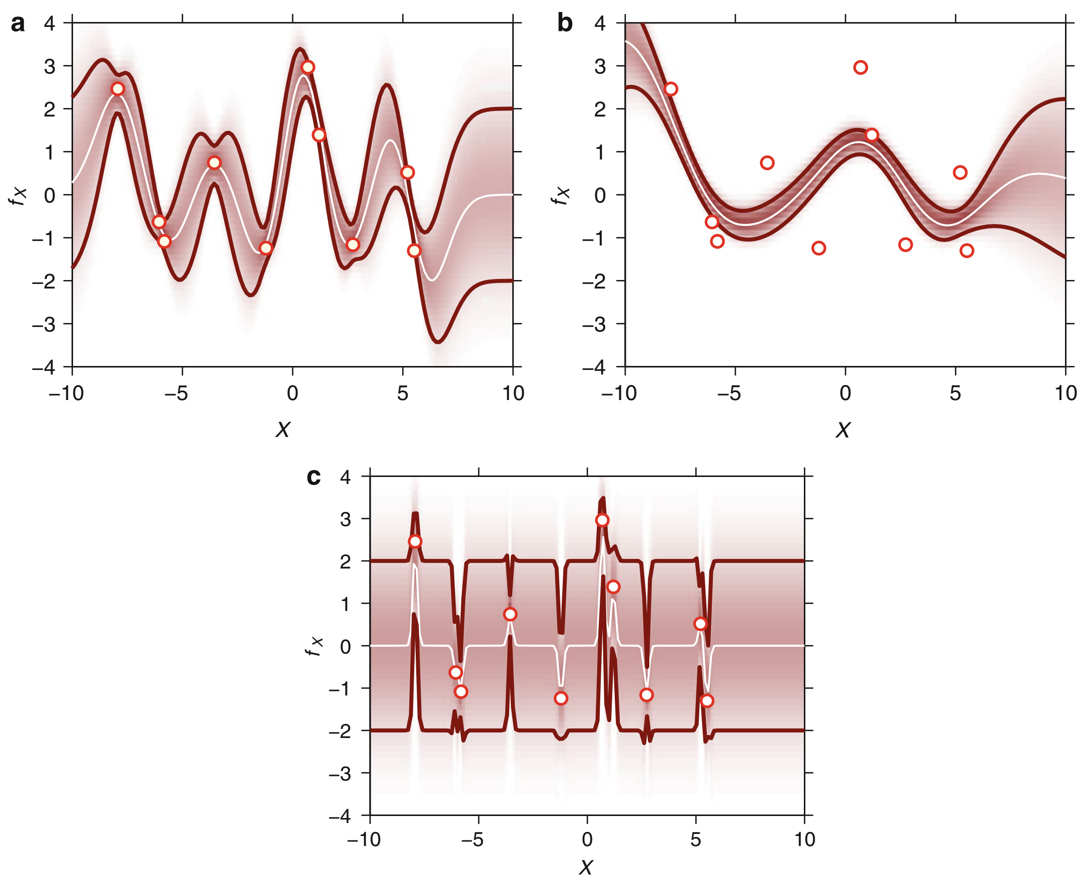
For a comprehensive review on the field of covariance functions, we refer interested readers to Abrahamsen (1992).

Applications

For Gaussian processes, there are two main classes of applications: regression and classification. We will discuss each of them in turn.

Regression

In a ▸ Regression problem, we are interested to recover 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 $x_i \in \mathbb{R}^d$. Traditionally, in the Bayesian ▸ Linear Regression model, this regression problem is tackled by requiring us to parameterize the latent function f by a parameter $w \in \mathbb{R}^H$, $f(x) := \langle \phi(x), w \rangle$ for H fixed basis functions $\{\phi_h(x)\}_{h=1}^H$. A prior distribution is then defined over parameter w. The idea of the Gaussian process regression (in the



Gaussian Process, Fig. 3 The Gaussian process prediction with the SE kernel. (**a**) Mean of the prediction distribution with length scale 1.0 and signal variance 1.0 (the hyperparameters of the original process used to generate the data in Fig. 1). The other two plots show the prediction setting of the length scale: **(b)** longer (3.0) and **(c)** shorter (0.1). In all plots, the 95 % confidence region is shown

Gaussian Process, Table 1 Examples of covariance functions. θ_{cov} denotes the set of hyperparameters

Name	$C(x, x')$	θ_{cov}	Remark
Squared exp. (SE)	$s^2 \exp\left(-\frac{1}{2\ell^2}\,\Vert x - x'\Vert_2^2\right)$	$\{s, \ell\}$	Strong smoothness assumption
Matérn class	$\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu\vert x-x'\vert}}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu r}}{\ell}\right)$	$\{\nu, \ell\}$	Less smooth than SE
γ-exponential	$\exp(-(\vert x - x'\vert/\ell)^{\gamma})$, with $0 < \gamma <= 2$	$\{\ell\}$	Includes both Exp. and SE
Exponential	$\exp\left(\frac{-\vert x-x'\vert}{\ell}\right)$	$\{\ell\}$	$\nu = 1/2$ in the Matérn class
Rational quadratic	$\left(1 + \frac{\Vert x-x'\Vert_2^2}{2\alpha\ell^2}\right)^{-\alpha}$	$\{\alpha, \ell\}$	An infinite sum of SE
Dot product	$\sigma_w^2\,\langle x, x'\rangle + \sigma_c^2$	$\{\sigma_w, \sigma_c\}$	
Polynomial	$(\langle x, x'\rangle + \sigma_c^2)^p$	$\{\sigma_c\}$	Effective for high-dimensional classification with binary or grayscale input

geostatistical literature, this is also called *kriging*; see, e.g., Krige 1951; Matheron 1963) is to place a prior directly on the space of functions without parameterizing the function (vide Motivation and Background).

Likelihood Function and Posterior Distribution

Assuming independent and normally distributed noise terms, $\epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2_{\text{noise}})$, the likelihood model on an output vector $Y \in \mathbb{R}^N$ and an input matrix $X \in \mathbb{R}^{N\times d}$ will be

$$\begin{aligned} p(Y \mid f_X) &= \prod_{i=1}^{N} p(y_i|x_i, f) \\ &= \mathcal{N}(Y \mid f_X, \sigma^2_{\text{noise}} I), \end{aligned}$$

with $f_X = (f(x_1), \ldots, f(x_N))^\top$ be an N-dimensional vector of function values at N input locations x_i. That is, the data likelihood is distributed according to a Gaussian distribution with the function values evaluated at training input locations as its mean and the variance of the noise terms as its variance.

Placing a (zero mean) Gaussian process prior over functions

$$f \sim \mathcal{GP}(m(x) \equiv 0, k(x, x')), \qquad (1)$$

will lead us to a Gaussian process posterior (this form of posterior process is described in the next section):

$$\begin{aligned} f|X, Y &\sim \mathcal{GP}(m_{\text{post}}(x) \\ &= k(x, X)[K + \sigma^2_{\text{noise}} I]^{-1} Y, k_{\text{post}}(x, x') \\ &= k(x, x') - k(x, X)[K + \sigma^2_{\text{noise}} I]^{-1} k(x', X)). \end{aligned} \qquad (2)$$

In the above equations, $K \in \mathbb{R}^{N\times N}$ denotes the Gram matrix with elements $K_{ij} = k(x_i, x_j)$, and $k(x, x')$ is the kernel function. The term $k(x, X)$ denotes a kernel function with one of the inputs fixed at training points.

Predictive Distribution

The final goal in regression is to make an output prediction for a novel input x_*, given a set of input–output training points. By the marginalization property, instead of working with a prior over infinite-dimensional function spaces as in (1), we can concentrate on the marginal distribution over the training inputs:

$$f_X \sim \mathcal{N}(0, K). \qquad (3)$$

Subsequently, the marginal distribution over training outputs (conditioned on inputs) can be computed via

$$\begin{aligned} p(Y|X) &= \int p(Y \mid f_X) p(f_X) \mathrm{d} f_X \\ &= \mathcal{N}(0, K + \sigma^2_{\text{noise}} I). \end{aligned} \qquad (4)$$

The above integration is computed by using the standard result for the convolution of two Gaussian distributions (▶ Gaussian Distribution).

Therefore, given inputs X, the joint distribution over outputs Y and the latent function f_X is given by

$$p(Y, f_X|X) = \mathcal{N}(0, C), \qquad (5)$$

where $C \in \mathbb{R}^{(2N)\times(2N)}$ is the joint covariance matrix. We can partition this joint covariance matrix as follows:

$$C = \begin{bmatrix} K + \sigma^2_{\text{noise}} I & K \\ K & K \end{bmatrix},$$

The noise variance appears only at the diagonal elements of the covariance matrix C; this is due to the independence assumption about the noise. Using a standard Gaussian property on computing conditional distribution from a joint Gaussian distribution (▶ Gaussian Distribution), the Gaussian posterior distribution can be seen to admit the following form: $p(f_X|X, Y) = \mathcal{N}(\mu_{f_X}, \Sigma_{f_X})$ with the mean $\mu_{f_X} = K(K + \sigma^2_{\text{noise}} I)^{-1} Y$ and the covariance $\Sigma_{f_X} = K - K(K + \sigma^2_{\text{noise}} I)^{-1} K$.

From the posterior distribution, we can compute the predictive distribution on the new output

y^* at an input location x^*, as follows:

$$p(y_*|x_*, X, Y) = \int p(y_*|f_*, x_*) \times \int p(f_*|f_X)p(f_X|X, Y)\mathrm{d}f_X \mathrm{d}f_*. \quad (6)$$

The $p(f_*|f_X)$ is a conditional multivariate Gaussian with mean $\mu_{f_*|f_X} = k_{X,x_*}^\top K^{-1} f_X$ and variance $\sigma^2_{f_*|f_X} = k(x_*, x_*) - k_{X,x_*}^\top K^{-1} k_{X,x_*}$ due to the GP marginalization property, where the vector $k_{X,x_*} \in \mathbb{R}^N$ has elements $k(x_i, x_*)$ for $i = 1, \ldots, N$ and $\top$ denotes a transpose operation. Equation (6) is a general equation for computing the predictive distribution in a Gaussian process framework and is applicable, among others, for both regression and classification settings. For regression, since all the terms are Gaussians, the inner integration $\int p(f_*|f_X)p(f_X|X, Y)\mathrm{d}f_X$ is a Gaussian with mean $\mu_{f_*} = k_{X,x_*}^\top K^{-1} \mu_{f_X}$ and variance $\sigma^2_{f_*} = k(x_*, x_*) - k_{X,x_*}^\top K^{-1} k_{X,x_*} + k_{X,x_*}^\top K^{-1} \Sigma_{f_X} K^{-1} k_{X,x_*}$. Subsequently, the outer integration $\int \mathcal{N}(f_*, \sigma^2_{\text{noise}})\mathcal{N}(\mu_{f_*}, \sigma^2_{f_*})\mathrm{d}f_*$ is also a Gaussian, and therefore the above $p(y_*|x_*, X, Y)$ is a Gaussian distribution, and it is in the form of

$$p(y_*|x_*, X, Y) = \mathcal{N}(\mu_*, \sigma_*^2), \quad (7)$$

with

$$\mu_* = k_{X,x_*}^\top (K + \sigma^2_{\text{noise}} I)^{-1} Y, \quad (8)$$

$$\sigma_*^2 = k(x_*, x_*) - k_{X,x_*}^\top (K + \sigma^2_{\text{noise}} I)^{-1} k_{X,x_*} + \sigma^2_{\text{noise}}. \quad (9)$$

Note that (8) and (9) are the mean function and the covariance function of the posterior process in (2) for any novel inputs. The only difference is the additional term σ^2_{noise}, since there exists observation noise ϵ_* such that $y_* = f_* + \epsilon_*$.

Point Prediction

The previous section has shown how to compute a predictive distribution for outputs y_* associated with the novel test inputs x_*. To convert this predictive distribution into a point prediction, we need the notion of a loss function, $\mathcal{L}(y_{\text{true}}, y_{\text{prediction}})$. This function specifies the loss incurred for predicting the value $y_{\text{prediction}}$, while the true value is y_{true}. Thus, the optimal point prediction can be computed by minimizing the expected loss as follows:

$$y_{\text{optimal}}|x_* = \underset{y_{\text{prediction}} \in \mathbb{R}}{\operatorname{argmin}} \int \mathcal{L}(y_*, y_{\text{prediction}}) \times p(y_*|x_*, X, Y)\mathrm{d}y_*. \quad (10)$$

For a squared loss function (or any other symmetric loss functions) and predictive distribution (7), the solution to the above equation is the mean of the predictive distribution, i.e.,

$$y_{\text{optimal}}|x_* = \mathbf{E}_{y_* \sim p(y_*|x_*, X, Y)}[y_*] = \mu_*.$$

The above Gaussian process regression description is known as a function space view in the literature (Rasmussen and Williams 2006). Equivalently, a Gaussian process regression can also be viewed from the traditional Bayesian linear regression with a possibly infinite number of basis functions $\phi(x)$ and with a zero mean and arbitrary positive-definite covariance matrix Gaussian prior on the parameter w; see, e.g., Rasmussen and Williams (2006).

Classification

Gaussian process models can also be applied to classification problems. In a probabilistic approach to classification, the goal is to model posterior probabilities of an input point x_i belonging to one of Ω classes, $y_i \in \{1, \ldots, \Omega\}$. These probabilities must lie in the interval [0, 1]; however, a Gaussian process model draws functions that lie on $(-\infty, \infty)$. For the binary classification ($\Omega = 2$), we can turn the output of a Gaussian process into a class probability using an appropriate nonlinear activation function. In the following, we will show this for the case of binary classification. For the more general cases, see, e.g., Rasmussen and Williams (2006).

Likelihood Function and Posterior Distribution

In a regression problem, a Gaussian likelihood is chosen and combined with a Gaussian process prior, which leads to a Gaussian posterior process over functions where in all required integrations remain analytically tractable. For classification, however, Gaussian likelihood is not the most suitable owing to the discreteness nature of the class labels. The most commonly used likelihood functions are

$$
\begin{aligned}
p(y_i \mid f, x_i) &= \frac{1}{1+\exp(-y_i f_{x_i})} \quad \text{or} \\
p(y_i \mid f, x_i) &= \int_{-\infty}^{y_i f_{x_i}} \mathcal{N}(0,1)\mathrm{d}t \\
&= \Phi_{0,1}(y_i f_{x_i}), \qquad (11)
\end{aligned}
$$

known as logistic and cumulative Gaussian likelihood functions, respectively. Assuming that the class labels (conditioned on f) are generated independent and identically distributed, the joint likelihood over N data points can be expressed as $p(Y \mid f_X) = \prod_{i=1}^{N} p(y_i \mid f, x_i)$. By Bayes' rule, the posterior distribution over latent functions is given by $p(f_X|X,Y) = \frac{p(Y \mid f_X)p(f_X)}{\int p(Y \mid f_X)p(f_X)\mathrm{d}f_X}$. This posterior is no longer analytically tractable (due to intractable integration in the denominator), and an approximation is needed.

There are several approximation methods to handle intractability of the inference stage in the Gaussian process classification such as Laplace approximation, expectation propagation, variational bounding, and MCMC, among others (see Nickisch and Rasmussen (2008) for a comprehensive overview of approximate inference in binary Gaussian process classification). Most of the methods (if not all) approximate the non-Gaussian posterior with a tractable Gaussian distribution. We describe in detail the straightforward Laplace approximation method, but note that the more complicated expectation propagation (▶ Expectation Propagation) is almost always the method of choice unless the computational budget is very tight (Nickisch and Rasmussen 2008).

Laplace method approximates the non-Gaussian posterior with a Gaussian one by performing a second-order Taylor expansion of the log posterior, $\log p(f_X|X,Y)$ at the maximum point of the posterior

$$
p(f_X|X,Y) \approx \hat{p}(f_X|X,Y) = \mathcal{N}(\hat{f}_X, H^{-1}), \qquad (12)
$$

where $\hat{f}_X = \operatorname{argmax}_{f_X} \log p(f_X|X,Y)$ and $H := -\nabla\nabla \log p\ (f_X|X,Y)|_{f_X=\hat{f}_X}$ is the Hessian of the negative log posterior at the maxima. Since the denominator of the Bayes' theorem is independent of the latent function, the mode of the posterior can be computed instead from the log un-normalized posterior:

$$
\Psi(f_X) := \log p(Y \mid f_X) + \log p(f_X), \qquad (13)
$$

with the expression for $p(f_X)$ given in (3). Computation of the mode requires us to evaluate the gradient of $\Psi(f_X)$ which is given as

$$
\nabla\Psi(f_X) = \nabla \log p(Y \mid f_X) - K^{-1} f_X. \qquad (14)
$$

To find the stationary point, however, we cannot simply set this gradient to zero as $\nabla \log p(Y \mid f)$ depends nonlinearly on f_X. We need to resort to an iterative scheme based on the Newton–Raphson's method with the updated equation given by

$$
f_X^{\text{new}} \leftarrow f_X^{\text{old}} - (\nabla\nabla\Psi(f_X))^{-1}\nabla\Psi(f_X), \qquad (15)
$$

and the Hessian given by

$$
\nabla\nabla\Psi(f_X) = -W - K^{-1}, \qquad (16)
$$

and $W := -\nabla\nabla \log p(Y \mid f_X)$ is a diagonal matrix. It is important to note that if the likelihood function $p(Y|f_X)$ is log-concave, the diagonal elements of W are nonnegative, and the Hessian in (16) is negative definite (since $-K$ and its inverse are negative definite by construction and the sum of two negative-definite matrices is also negative definite). Thus, $\Psi(f_X)$ is concave and has a unique maxima point.

Predictive Distribution

The latent function f_X plays the role of a nuisance function, i.e., we do not observe values of f_X itself, and more importantly, we are not particularly interested in the values of f_X. What we are interested in is a class conditional posterior probability, $p(y_* = +1|x_*, X, Y)$, for a novel input x_*. We note that a class conditional probability of a class label of not 1 is $p(y_* = -1|x_*, X, Y) = 1 - p(y_* = +1|x_*, X, Y)$.

We use Equation (6) to compute the predictive distribution on the new output y^* at an input location x^*, restated here for the sake of readability:

$$p(y_*|x_*, X, Y) = \int p(y_*|f_*, x_*) \times \int p(f_*|f_X) p(f_X|X, Y) \mathrm{d}f_X \mathrm{d}f_*.$$

As in regression, the term $p(f_*|f_X)$ is a conditional multivariate Gaussian with the assumption that the underlying Gaussian process model is a noise-free process. Another approach would be assuming an independent Gaussian noise in combination with a step function likelihood function. However, this is equivalent to the noise-free latent process with a cumulative Gaussian likelihood function (Rasmussen and Williams 2006). With Laplace approximation of posterior distribution $p(f_X|X, Y) \approx \mathcal{N}(\hat{f}_X, (K^{-1} + W)^{-1})$, we can now compute the inner integration of the predictive distribution, $\int \mathcal{N}(\mu_{f_*|f_X}, \sigma^2_{f^*|f_X})\mathcal{N}(\hat{f}_X, (K^{-1} + W)^{-1})\mathrm{d}f_X$, by using the standard result for the convolution of two Gaussian distributions. It is again a Gaussian with mean $\mu_{f_*} = k^\top_{X,x_*} K^{-1} \hat{f}_X$ and variance $\sigma^2_{f_*} = k(x_*, x_*) - k^\top_{X,x_*}(K + W^{-1})^{-1} k_{X,x_*}$.

The predictive distribution can now be computed as follows:

$$\pi_* := p(y_* = +1|x_*, X, Y) = \int p(y_* = +1|f_*, x_*)\mathcal{N}(\mu_{f_*}, \sigma^2_{f_*})\mathrm{d}f_*.$$

The above integral can be solved analytically for a cumulative Gaussian likelihood function,

$$\pi_* = \int_{-\infty}^{\frac{\mu_{f_*}}{(y_*^{-2}+\sigma^2_{f_*})^{1/2}}} \mathcal{N}(t|0, 1)\mathrm{d}t = \Phi_{0,1}\left(\frac{\mu_{f_*}}{(y_*^{-2} + \sigma^2_{f_*})^{1/2}}\right),$$

and can be approximated for a logistic likelihood function (MacKay 1992),

$$\pi_* = \frac{1}{1 + \exp(-\mu_{f_*}\kappa(\sigma^2_{f_*}))},$$

with $\kappa(c) = (1 + c\pi/8)^{-1/2}$.

Point Prediction

Similar to the regression case, we might need to make a point prediction from the predictive distribution described in the section above. For a zero-one loss function, i.e., a loss of one unit is suffered for a wrong classification and 0 for not making a classification mistake, the optimal point prediction (in the sense of expected loss) is

$$y_{\text{optimal}}|x^* = \underset{y_* \in \{1,\ldots,\Omega\}}{\operatorname{argmax}} \; p(y_*|x_*, X, Y). \quad (17)$$

It is worth noting that the probabilistic approach to classification allows the same inference stage to be reused with different loss functions. In some situations, a cost-sensitive loss function, i.e., different classification mistakes incur different losses, is more desirable. The optimal point prediction is now taken by minimizing expected cost-sensitive loss with respect to the same $p(y_*|x_*, X, Y)$.

Extension of binary classification to multiclass Gaussian process classification ($\Omega > 2$) can be achieved via the softmax activation function, i.e., a generalization of logistic activation function (refer to Williams and Barber (1998) for the Laplace approximation of the posterior distribution). Recently, Bratières, Quadrianto, and Ghahramani (to appear) propose a Gaussian process classification approach to structured output problems ($\Omega \gg 2$) using a generalization of softmax function called a structured softmax function. Examples of structured outputs are a tree, a grid, or a sequence, where the output consists of interdependent categorical atoms.

Practical Issues

We have seen how to do regression and classification using Gaussian processes. Like other kernel-based methods such as support vector machines, they are very flexible in that all operations are kernelized, i.e., the operations are performed in the (possibly infinite dimensional) feature space. However, this feature space is only defined implicitly via positive-definite kernels (covariance functions), which only require computation in the (lower dimensional) input space. Compared to other non-Bayesian kernel approaches, Gaussian processes provide an explicit probabilistic formulation of the model. This directly provides us with confidence intervals (for regression) or posterior class probabilities (for classification).

So far, however, we have assumed a covariance function with the known functional form and hyperparameters. In many practical applications, it may not be easy to specify all aspects of the covariance function by hand. Furthermore, inverting the corresponding $N \times N$ Gram matrix is the main computational cost, and it may be prohibitive as it scales as $\mathcal{O}(N^3)$. We will now discuss approaches to overcome both limitations in turn.

Model Selection

In many practical applications, the functional form of the covariance function needs to be chosen, and any values of hyperparameters associated with the chosen covariance function and possible free parameters of the likelihood function need to be optimally determined. This is called model selection.

Ideally, we would like to define a prior distribution over the hyperparameters θ, and predictions are made by integrating over different possible choices of hyperparameters. More formally,

$$p(y_*|x_*, X, Y) = \int p(y_*|x_*, X, Y, \theta)p(\theta|X, Y)\mathrm{d}\theta. \quad (18)$$

The evaluation of the above integral, however, may be difficult, and an approximation is needed either by using the most likely value of hyperparameters, $p(y_*|x_*, X, Y) \approx p(y_*|x_*, X, Y, \theta_{\mathrm{ML}})$, or by performing the integration numerically via Monte Carlo methods. We will focus here on the approximation approach and show how to use it for regression and classification problems.

Marginal Likelihood for Regression

The posterior probability of the hyperparameters θ in (18) is

$$p(\theta|X, Y) \propto p(Y|X, \theta)p(\theta), \quad (19)$$

where the first term is known as marginal likelihood or evidence for the hyperparameters and it is in the form of (as in (4) but with an explicit conditioning on θ)

$$\begin{aligned} p(Y|X, \theta) &= \int p(Y|\, f_X, \theta)p(f_X|\theta)\mathrm{d}f_X \\ &= \mathcal{N}(0, K + \sigma^2_{\mathrm{noise}} I), \end{aligned}$$

where the set of free parameters θ includes both parameters of the kernel function and the noise term $\sigma^2_{\mathrm{noise}}$. We can then set the hyperparameters by maximizing the logarithm of this marginal likelihood, and its partial derivative with respect to hyperparameters is

$$\begin{aligned} &\frac{\partial}{\partial\theta_j} \log p(Y|X, \theta) \\ &= \frac{1}{2} Y^\top \bar{K}^{-1} \frac{\partial \bar{K}}{\partial\theta_j} \bar{K}^{-1} Y - \frac{1}{2}\mathrm{tr}\left(\bar{K}^{-1}\frac{\partial \bar{K}}{\partial\theta_j}\right), \end{aligned}$$

with $\bar{K} := K + \sigma^2_{\mathrm{noise}} I$. This is known as a type II maximum likelihood approximation, ML-II. We can also maximize the un-normalized posterior instead, assuming finding the derivatives of the priors is straightforward.

Marginal Likelihood for Classification

The Laplace approximation of the marginal likelihood, $p(Y|X, \theta) \approx \hat{p}(Y|X, \theta)$

$$
\begin{aligned}
&= \int \exp(\Psi(f_X))\mathrm{d}f_X \\
&= \exp(\Psi(\hat{f}_X)) \int \exp(-\frac{1}{2}(f_X - \hat{f}_X)^\top \\
&\quad H(f_X - \hat{f}_X))\mathrm{d}f_X,
\end{aligned}
$$

which is achieved via a Taylor expansion of (13) locally around $\hat{f}_X$ to obtain $\Psi(f_X) \approx \Psi(\hat{f}_X) - \frac{1}{2}(f_X - \hat{f}_X)^\top H(f_X - \hat{f}_X)$. Computing the integral analytically gives us the approximate marginal likelihood

$$
\log \hat{p}(Y|X,\theta) \propto -\frac{1}{2}\hat{f}_X K^{-1}\hat{f}_X - \frac{1}{2}\log|K| + \log p(Y|\hat{f}_X) - \frac{1}{2}\log|K^{-1} + W|.
$$

Subsequently, the partial derivatives with respect to hyperparameters is given by

$$
\begin{aligned}
\frac{\partial}{\partial\theta_j}\log \hat{p}(Y|X,\theta) &= \frac{1}{2}\hat{f}_X^\top K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\hat{f}_X \\
&\quad - \frac{1}{2}\mathrm{tr}\left((K + W^{-1})^{-1}\frac{\partial K}{\partial\theta_j}\right) \\
&\quad + \sum_{i=1}^{N}\frac{\partial \log \hat{p}(Y|X,\theta)}{\partial \hat{f}_{x_i}}\frac{\partial \hat{f}_{x_i}}{\partial\theta_j}.
\end{aligned}
$$

The familiar multiple local optima problem is also present in the marginal likelihood maximization. However, practical experiences suggest that local optima are not a devastating problem especially with simple functional forms of covariance function.

Sparse Approximation

A significant problem with the Gaussian process model is associated with the computation cost of inverting the $N \times N$ Gram matrix. A number of sparse approximation methods have been proposed to overcome this high computational demand. Common to all these methods is that only a subset of the latent function values of size $M < N$ are treated exactly, and the remaining latent values are approximated with cheaper computational demand. Quiñonero-Candela and Rasmussen (2005) describe a unifying view of sparse approximation. Several existing sparse methods are shown to be an instance of it. The framework is described for regression problems; however, it should also be applicable for classification learning settings, albeit with complicacy associated with the non-Gaussian likelihood.

In this unifying treatment, an additional set of M latent variables $f_U \in \mathbb{R}^M$, called inducing variables, is introduced. These latent variables are latent function values corresponding to a set of input locations $X_U \in \mathbb{R}^{M\times d}$, called inducing inputs. The choice of inducing inputs is not restricted to only form the training or test inputs. Due to the marginalization property, introducing more latent variables will not change the distribution of the original variables. Consider (5) but as a joint distribution over latent training and test function values, $p(f_X, f_*|X, x_*)$

$$
\begin{aligned}
&= \int p(f_X, f_*, f_U|X, X_U, x_*)\mathrm{d}f_U \\
&= \int p(f_X, f_*|X, x_*, f_U)p(f_U)\mathrm{d}f_U, \qquad (20)
\end{aligned}
$$

with $p(f_U) = \mathcal{N}(0, K_{X_U,X_U})$. So far, no approximations have been introduced. Introducing the key assumption which is f_X is conditionally independent of f_* given f_U, $f_* \perp\!\!\!\perp f_X \mid f_U$, allow us to approximate (20) as

$$
p(f_X, f_*|X, x_*) \approx \int p(f_*|x_*, f_U)p(f_X|X, f_U)p(f_U)\mathrm{d}f_U, \qquad (21)
$$

where $p(f_*|x_*, f_U)$ and $p(f_X|X, f_U)$ are again conditional multivariate Gaussians due to the GP marginalization property. Different computationally efficient algorithms in the literature correspond to different assumptions made on those two conditional distributions. Table 2 shows various sparse approximation methods with their corresponding approximated conditional distributions. For all sparse approximation methods, the com-

Gaussian Process, Table 2 Sparse approximation methods

Method	$\hat{p}(f_X \mid X, f_U)$	$\hat{p}(f_* \mid x_*, f_U)$	Ref.
SR	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	$\mathcal{N}(K_{x_*,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	Silverman (1985)
PP	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, 0)$	$p(f_* \mid x_*, f_U)$	Seeger et al. (2003)
SPGPs	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, \Delta_1)$ $\Delta_1 = \mathrm{diag}[K_{X,X} - K_{X,X_U} K^{-1}_{X_U,X_U} K_{X_U,X}]$	$p(f_* \mid x_*, f_U)$	Snelson and Ghahramani (2006)
BCM	$\mathcal{N}(K_{X,X_U} K^{-1}_{X_U,X_U} f_U, \Delta_2)$	$p(f_* \mid x_*, f_U)$	Tresp (2000a)
	$\Delta_2 = \mathrm{blockdiag}[K_{X,X} - K_{X,X_U} K^{-1}_{X_U,X_U} K_{X_U,X}]$		

SR subset of regressors, *PP* projected process, *SPGPs* sparse pseudo-input Gaussian processes, *BCM* Bayesian committee machine

putational complexity is reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$.

Current and Future Directions

Gaussian processes are an active area of research both within the machine learning and the Bayesian statistics community. First, there is the issue of efficient inference and learning as already discussed in the text above. Some of the recent approaches include converting Gaussian processes with stationary covariance functions to state-space models (Särkkä et al. 2013) and using stochastic variational inference (Hensman et al. 2013). Second, there is interest in adapting Gaussian processes to other learning settings. They have been used for ordinal regression (Chu and Ghahramani 2005a; Yu et al. 2006b), preference learning (Chu and Ghahramani 2005b), ranking (Guiver and Snelson 2008), mixtures of experts (Tresp 2000b), transductive learning (Schwaighofer and Tresp 2003), multitask learning (Yu et al. 2005), dimensionality reduction (Lawrence 2005), matrix factorization (Lawrence and Urtasun 2009), and reinforcement learning (Engel et al. 2005; Deisenroth and Rasmussen 2009), among other settings. They have also been extended to handle relational data (Chu et al. 2006; Yu et al. 2006a; Silva et al. 2007; Xu et al. 2009; Kersting and Xu 2009). Standard Gaussian processes only exploit the available information about attributes of instances and typically ignore any relations among the instances. Intuitively, however, we would like to use our information about one instance to help us reach conclusions about other related instances.

There is the issue of relaxing the assumption of the standard Gaussian process model that the noise on the output is uniform throughout the domain. If we assume that the noise is a smooth function of the inputs, the noise variance can be modeled using a second Gaussian process, in addition to the process governing the noise-free output values. The resulting heteroscedastic, i.e., input-dependent noise regression model, has been shown to outperform state-of-the-art methods for mobile robot localization (Plagemann et al. 2007). Heteroscedastic classification has also been explored by Hernández-Lobato et al. (2014) in the context of ▶ Learning Using Privileged Information.

Finally, Gaussian processes are also of great interest for practical applications because they naturally deal with noisy measurements, unevenly distributed observations, and fill small gaps in the data with high confidence while assigning higher predictive uncertainty in sparsely sampled areas. Two recent applications areas are Bayesian optimization for automatically tuning hyperparameters of machine-learning models (see, e.g., Snoek et al. 2012) and an automated Bayesian statistician for automating the process of statistical modeling (see, e.g., Lloyd et al. 2014).

In addition to the references embedded in the text above, we also recommend http://www.gaussian-process.org/. A highly recommended textbook is Rasmussen and Williams (2006).

Cross-References

▶ Dirichlet Process

Recommended Reading

Abrahamsen P (1992) A review of Gaussian random fields and correlation functions. Rapport 917, Norwegian Computing Center, Oslo. www.nr.no/publications/917_Rapport.ps

Bratières S, Quadrianto N, Ghahramani Z (to appear) GPstruct: Bayesian structured prediction using Gaussian processes. IEEE Trans Pattern Anal Mach Intell

Chu W, Ghahramani Z (2005a) Gaussian processes for ordinal regression. J Mach Learn Res 6:1019–1041

Chu W, Ghahramani Z (2005b) Preference learning with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Bonn

Chu W, Sindhwani V, Ghahramani Z, Keerthi S (2006) Relational learning with Gaussian processes. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Deisenroth MP, Rasmussen CE, Peters J (2009) Gaussian process dynamic programming. Neurocomputing 72(7–9):1508–1524

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Bonn

Guiver J, Snelson E (2008) Learning to rank with softrank and Gaussian processes. In: Proceedings of special interest group on information retrieval (SIGIR), Singapore

Hensman J, Fusi N, Lawrence N (2013) Gaussian processes for big data. In: Proceedings of uncertainty in artificial intelligence (UAI), Bellvue

Kersting K, Xu Z (2009) Learning preferences with hidden common cause relations. In: Proceedings of European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD), Bled

Krige DG (1951) A statistical approach to some basic mine valuation problems on the witwatersrand. J Chem Metall Mining Soc S Afr 52(6):119–139

Lawrence N (2005) Probabilistic non-linear principal component analysis with Gaussian process latent variable models. J Mach Learn Res 6:1783–1816

Lawrence N, Urtasun R (2009) Non-linear matrix factorization with Gaussian processes. In: Proceedings of international conference on machine learning (ICML), Montreal

Lloyd JR, Duvenaud D, Grosse R, Tenenbaum JB, Ghahramani Z (2014) Automatic construction and natural-language description of nonparametric regression models. In: Proceedings of association for the advancement of artificial intelligence (AAAI), Québec City

Hennig P (2013) Animating samples from Gaussian distributions. Technical report, 8. Max Planck Institute for Intelligent Systems

Hernández-Lobato D, Sharmanska V, Kersting K, Lampert C, Quadrianto N (2014) Mind the Nuisance: Gaussian process classification using privileged noise. In: Proceedings of advances in neural information processing systems (NIPS), Montreal

MacKay DJC (1992) The evidence framework applied to classification networks. Neural Comput 4(5):720–736

Matheron G (1963) Principles of geostatistics. Econ Geol 58:1246–1266

Neal R (1996) Bayesian learning in neural networks. Springer, New York

Nickisch H, Rasmussen CE (2008) Approximations for binary Gaussian process classification. J Mach Learn Res 9:2035–2078

Plagemann C, Kersting K, Pfaff P, Burgard W (2007) Gaussian beam processes: a nonparametric Bayesian measurement model for range finders. In: Proceedings of robotics: science and systems (RSS), Atlanta

Quiñonero-Candela J, Rasmussen CE (2005) A unifying view of sparse approximate Gaussian process regression. J Mach Learn Res 6:1939–1959

Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. MIT Press, Cambridge

Särkkä S, Solin A, Hartikainen J (2013) Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing. IEEE Signal Process Mag 30:51–61

Schwaighofer A, Tresp V (2003) Transductive and inductive methods for approximate guassian process regression. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Seeger M, Williams CKI, Lawrence N (2003) Fast forward selection to speed up sparse Gaussian process regression. In: Proceedings of artificial intelligence and statistics (AISTATS), Key West

Silva R, Chu W, Ghahramani Z (2007) Hidden common cause relations in relational learning. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Silverman BW (1985) Some aspects of the spline smoothing approach to non-parametric regression curve fitting. J R Stat Soc B 47(1):1–52

Snelson E, Ghahramani Z (2006) Sparse Gaussian processes using pseudo-inputs. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver

Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Proceedings of advances in neural information processing systems (NIPS), Lake Tahoe

Tresp V (2000a) A Bayesian committee machine. Neural Comput 12(11):2719–2741
Tresp V (2000b) Mixtures of Gaussian processes. In: Proceedings of advances in neural information processing systems (NIPS), Denver
Williams C, Barber D (1998) Bayesian classification with Gaussian processes. IEEE Trans Pattern Anal Mach Intell PAMI 20(12):1342–1351
Williams C, Rasmussen C (1996) Gaussian processes for regression. In: Proceedings of advances in neural information processing systems (NIPS), Denver
Xu Z, Kersting K, Tresp V (2009) Multi-relational learning with Gaussian processes. In: Proceedings of the international joint conference on artificial intelligence (IJCAI), Pasadena
Yu K, Tresp V, Schwaighofer A (2005) Learning Gaussian processes from multiple tasks. In: Proceedings of international conference on machine learning (ICML), Bonn
Yu K, Chu W, Yu S, Tresp V, Xu Z (2006a) Stochastic relational models for discriminative link prediction. In: Proceedings of advances in neural information processing systems (NIPS), Vancouver
Yu S, Yu K, Tresp V, Kriegel HP (2006b) Collaborative ordinal regression. In: Proceedings of international conference on machine learning (ICML), Pittsburgh

Gaussian Process Reinforcement Learning

Yaakov Engel
University of Alberta, Edmonton, AB, Canada

Definition

Gaussian process reinforcement learning generically refers to a class of ▸ reinforcement learning (RL) algorithms that use Gaussian processes (GPs) to model and learn some aspect of the problem.

Such methods may be divided roughly into two groups:

1. *Model-based methods*: Here, GPs are used to learn the transition and reward model of the ▸ Markov decision process (MDP) underlying the RL problem. The estimated MDP model is then used to compute an approximate solution to the true MDP.
2. *Model-free methods*: Here, no explicit representation of the MDP is maintained. Rather, GPs are used to learn either the MDP's value function, state–action value function, or some other quantity that may be used to solve the MDP.

This entry is concerned with the latter class of methods, as these constitute the majority of published research in this area.

Motivation and Background

▸ Reinforcement learning is a class of learning problems concerned with achieving long-term goals in unfamiliar, uncertain, and dynamic environments. Such tasks are conventionally formulated by modeling the environment as ▸ MDPs (or more generally as partially observable MDPs (▸ POMDPs)) and modeling the agent as an adaptive controller implementing an action-selection policy.

Markov Decision Processes

Let us denote by $\mathcal{P}(\mathcal{S})$ the set of probability distributions over (Borel) subsets of a set $\mathcal{S}$. A discrete time MDP is a tuple $(\mathcal{X}, \mathcal{U}, p_0, p, q, \gamma)$, where $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively; $p_0(\cdot) \in \mathcal{P}(\mathcal{X})$ is a probability density over initial states; $p(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathcal{X})$ is a probability density over successor states, conditioned on the current state $\mathbf{x}$ and action $\mathbf{u}$; $q(\cdot|\mathbf{x}, \mathbf{u}) \in \mathcal{P}(\mathbb{R})$ is a probability distribution over immediate single-step rewards, conditioned on the current state and action. We denote by $R(\mathbf{x}, \mathbf{u})$ the random variable distributed according to $q(\cdot|\mathbf{x}, \mathbf{u})$. Finally, $\gamma \in [0, 1]$ is a discount factor. We assume that both p and q are stationary, that is, they do not depend explicitly on time. To maintain generality, we use $\mathbf{z}$ to denote either a state $\mathbf{x}$ or a state–action pair $(\mathbf{x}, \mathbf{u})$. This overloaded notation will allow us to present models and algorithms in a concise and unified form.

In the context of control, it is useful to make several additional definitions. A *stationary policy* $\mu(\cdot|\mathbf{x}) \in \mathcal{P}(\mathcal{U})$ is a time-independent mapping from states to action-selection probabilities. A stationary policy μ induces a Markov reward process (MRP) (Puterman 1994) via *policy-dependent* state-transition probability density, defined as (Here and in the sequel, whenever integration is performed over a finite or discrete space, the integral should be understood as a summation.)

$$p^{\mu}_{\mathbf{x}}(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} d\mathbf{u}\, \mu(\mathbf{u}|\mathbf{x}) p(\mathbf{x}'|\mathbf{u},\mathbf{x}).$$

Similarly, the policy μ may also be used to define a state–action transition probability density, defined as

$$p^{\mu}_{\mathbf{x},\mathbf{u}}(\mathbf{x}',\mathbf{u}'|\mathbf{x},\mathbf{u}) = p(\mathbf{x}'|\mathbf{u},\mathbf{x})\mu(\mathbf{u}'|\mathbf{x}').$$

Using our overloaded notational convention, we refer to either of these as $p^{\mu}_{\mathbf{z}}$. Let us denote by $\boldsymbol{\xi}(\mathbf{z})$ a path that starts at $\mathbf{z}$. Hence, for a fixed policy μ and a fixed initial state or state–action pair $\mathbf{z}_0$, the probability (density) of observing the path $\boldsymbol{\xi}(\mathbf{z}_0) = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ of length t is (we take $\mathbf{z}_0$ as given) $\mathbb{P}(\boldsymbol{\xi}(\mathbf{z}_0)) = \prod_{i=1}^{t} p^{\mu}_{\mathbf{z}}(\mathbf{z}_i|\mathbf{z}_{i-1})$. The *discounted return* $D^{\mu}(\boldsymbol{\xi}(\mathbf{z}))$ of a path $\boldsymbol{\xi}(\mathbf{z})$ is a random process, defined (with some abuse of notation) as

$$D^{\mu}(\mathbf{z}) = D^{\mu}(\boldsymbol{\xi}(\mathbf{z})) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{z}_i)|(\mathbf{z}_0 = \mathbf{z}), \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor. (When $\gamma = 1$, the policy must be proper; see Bertsekas and Tsitsiklis (1996).) The randomness in $D^{\mu}(\mathbf{z})$, for any given $\mathbf{z}$, is due both to $\boldsymbol{\xi}(\mathbf{z})$ being a random process and to the randomness, or noise, in the rewards $R(\mathbf{z}_0), R(\mathbf{z}_1), \ldots$, etc., both of which jointly constitute the *intrinsic* randomness of the MDP. Equation (1) together with the stationarity of the MDP yields the recursive formula

$$D^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma D^{\mu}(\mathbf{z}') \quad \text{where } \mathbf{z}' \sim p^{\mu}_{\mathbf{z}}(\cdot|\mathbf{z}). \tag{2}$$

Let us define the expectation operator $\mathbf{E}_{\xi}$ as the expectation over all possible trajectories and all possible rewards collected in them. This allows us to define the *value function* $V^{\mu}(\mathbf{z})$ as the result of applying this expectation operator to the discounted return $D^{\mu}(\mathbf{z})$, i.e.,

$$V^{\mu}(\mathbf{z}) = \mathbf{E}_{\xi} D^{\mu}(\mathbf{z}). \tag{3}$$

Applying the law of total expectation to this equation results in the MRP (fixed policy) version of the Bellman equation:

$$V^{\mu}(\mathbf{z}) = R(\mathbf{z}) + \gamma \mathbf{E}_{\mathbf{z}'|\mathbf{z}}[V^{\mu}(\mathbf{z}')]. \tag{4}$$

A policy that maximizes the expected discounted return from each state is called an optimal policy and is denoted by μ^*. In the case of stationary MDPs, there exists a *deterministic* optimal policy (this is no longer the case for POMDPs and Markov games; see Kaelbling et al. (1998) and Littman (1994)). The value function corresponding to an optimal policy is called the optimal value and is denoted by $V^* = V^{\mu^*}$. While there may exist more than one optimal policy, the optimal value function is unique (Bertsekas 1995).

Reinforcement Learning

Many of the algorithms developed for solving RL problems may be traced back to the ▶ dynamic programming *value iteration* and *policy iteration* algorithms (Bellman 1957; Bertsekas 1995; Bertsekas and Tsitsiklis 1996; Howard 1960). However, there are two major features distinguishing RL from the traditional planning framework. First, while in planning it is assumed that the environment is fully known, in RL no such assumption is made. Second, the learning process in RL is usually assumed to take place *online*, namely, concurrently with the acquirement of data by the learning agent as it interacts with its environment. These two features make solving RL problems a significantly more challenging undertaking.

An important algorithmic component of policy iteration-based RL algorithms is the estimation of either state or state–action values of a fixed policy controlling an MDP, a task known as *policy evaluation*. Sutton's TD(λ) algorithm (Sutton 1984) is an early RL algorithm that performs policy evaluation based on observed sample trajectories from the MDP, while it is being controlled by the policy being evaluated (see ▶ Temporal Difference Learning). In its original formulation, TD(λ) as well as many other algorithms (e.g., Watkins' ▶ Q-Learning 1989) employs a lookup table to store values corresponding to the MDP's states or state–action pairs. This approach clearly becomes infeasible when the size of the MDP's joint state–action space exceeds the memory capacity of modern workstations. One solution to this problem is to represent the value function using a parametric function approximation architecture and allow these algorithms to estimate the parameters of approximate value functions. Unfortunately, with few exceptions, this seemingly benign modification turns out to have ruinous consequences to the convergence properties of these algorithms. One notable exception is TD(λ), when it is used in conjunction with a function approximator $\hat{V}(\mathbf{z}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{z})$, which is linear in its tunable parameters $\mathbf{w} = (w_1, \ldots, w_N)^\top$. Under certain technical conditions, it has been shown that in this case, TD(λ) converges almost surely, and the limit of convergence is "close" (in a well-defined manner) to a projection ΠV^μ of the true value function V^μ onto the finite-dimensional space $\mathcal{H}_\phi$ of functions spanned by $\{\phi_i | i = 1, \ldots, N\}$ (Tsitsiklis and Van Roy 1996). Note that this projection is the best one may hope for, as long as one is restricted to a fixed function approximation architecture. In fact, when $\lambda = 1$, the bound of Tsitsiklis and Van Roy (1996) implies that TD(1) converges to ΠV^μ (assuming it is unique). However, as λ is reduced toward 0, the quality of TD(λ)'s solution may deteriorate significantly. If V^μ happens to belong to $\mathcal{H}_\phi$, then $V^\mu = \Pi V^\mu$ and TD(λ) converges almost surely to V^μ, for any $\lambda \in [0, 1]$.

As noted in Bertsekas and Tsitsiklis (1996), TD(λ) is a stochastic approximation algorithm (Kushner and Yin 1997). As such, to ensure convergence to a meaningful result, it relies on making small and diminishing updates to its value function estimates. Moreover, in the typical online mode of operation of TD(λ), a sample is observed, is acted upon (by updating the parameters of $\hat{V}$), and is then discarded, never to be seen again. A negative consequence of these two properties is that online TD(λ) is inherently wasteful in its use of the observed data. The least-squares TD(λ), or LSTD(λ) algorithm (Boyan 1999; Bradtke and Barto 1996), was put forward as an alternative to TD(λ) that makes better use of data, by directly solving a set of equations characterizing the fixed point of the TD(λ) updates. LSTD(λ) is amenable to a recursive implementation, at a time and memory cost of $O(N^2)$ per sample. A more fundamental shortcoming, shared by both TD(λ) and LSTD(λ), is that they do not supply the user with a measure of the accuracy of their value predictions.

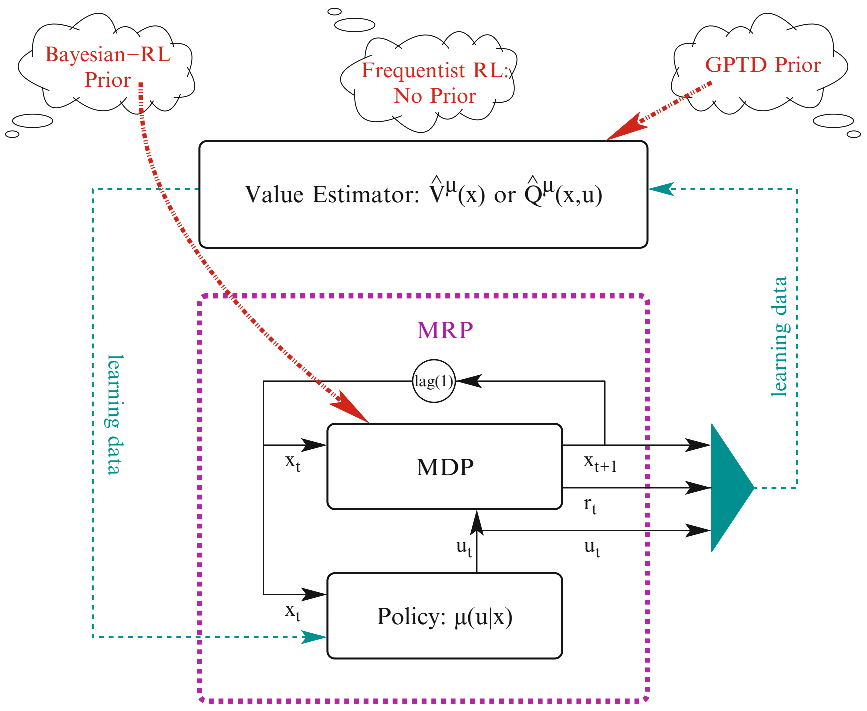
The discussion above motivates the search for:

1. Nonparametric estimators for V^μ, since these are not generally restricted to searching in any finite-dimensional hypothesis space, such as $\mathcal{H}_\phi$.
2. Estimators that make efficient use of the data.
3. Estimators that, in addition to value predictions, deliver a measure of the uncertainty in their predictions.

Structure of Learning System

We first describe the structure and operation of the basic GP temporal difference (GPTD) algorithm for policy evaluation. We then build on this algorithm to describe policy-improving algorithms, in the spirit of Howard's policy iteration (Howard 1960).

In the preceding section, we showed that the value V is the result of taking the expectation of the discounted return D with respect to the randomness in the trajectories and in the rewards collected therein. In the classic or frequentist approach, V is no longer random, since it is the true, albeit unknown value function induced by the policy μ. Adopting the Bayesian approach, we may still view the value V as a random entity by assigning it additional randomness, that is due to our *subjective uncertainty* regarding the MDP's transition model (p, q). We do not know what the true distributions p and q are,



Gaussian Process Reinforcement Learning, Fig. 1 An illustration of the frequentist as well as the two different Bayesian approaches to value function-based reinforcement learning. In the traditional Bayesian RL approach, a prior is placed on the MDP's model, whereas in our GPTD approach, the prior is placed directly on the value function. x, $\mathbf{u}$, and r denote state, action, and reward, respectively. The data required to learn value estimators typically consists of a temporal stream of state–action–reward triplets. Another stream of data is used to update the policy based on the current estimate of the value function. An MDP and a stationary policy controlling it jointly constitute an MRP. lag(1) denotes the 1-step time-lag operator

which means that we are also uncertain about the true value function. Previous attempts to apply Bayesian reasoning to RL modeled this uncertainty by placing priors over the MDP's transition and reward model (p, q) and applying Bayes' rule to update a posterior based on observed transitions. This line of work may be traced back to the pioneering works of Bellman (1956) and Howard (1960) followed by more recent contributions in the machine learning literature (Dearden et al. 1999, 1998; Duff 2002; Mannor et al. 2004; Poupart et al. 2006; Strens 2000; Wang et al. 2005). A fundamental shortcoming of this approach is that the resulting algorithms are limited to solving MDPs with finite (and typically rather small) state and action spaces, due to the need to maintain a probability distribution over the MDP's transition model. In this work, we pursue a different path – we choose to model our uncertainty about the MDP by placing a prior (and updating a posterior) directly on V. We achieve this by modeling V as a random process or, more specifically, as a Gaussian process. This mirrors the traditional classification of classical RL algorithms to either model-based or model-free (direct) methods; see Chapter 9 in Sutton and Barto (1998). Figure 1 illustrates these different approaches.

Gaussian Process Temporal Difference Learning

GPTD should be viewed as a family of *statistical generative models* (see ▸ Generative Learning) for value functions, rather than as a family of algorithms. As such, GPTD models specify the statistical relation between the unobserved value function and the observable quantities, namely, the observed trajectories and the rewards collected in them. The set of equations prescribing the GPTD model for a path $\boldsymbol{\xi} = (\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_t)$ is (Here and in the sequel, to simplify notation, we omit the superscript μ, with the understanding that quantities such as D, V, or $\boldsymbol{\xi}$ generally depend on the policy μ being evaluated.)

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1})$$
$$\text{for } i = 0, 1, \ldots, t-1.$$

$N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is a zero-mean noise term that must account for the statistics of $R(\mathbf{z}_i) + \gamma V(\mathbf{z}_{i+1}) - V(\mathbf{z}_i)$. If V is a priori distributed according to a GP prior and the noise term $N(\mathbf{z}_i, \mathbf{z}_{i+1})$ is also normally distributed, then $R(\mathbf{z}_i)$ is also normally distributed and so is the posterior distribution of V conditioned on the observed rewards. To fully specify the GPTD model, we need to specify the GP prior over V in terms of prior mean and covariance as well as the covariance of the noise process N. In Engel et al. (2003), it was shown that modeling N as a white noise process is a suitable choice for MRPs with deterministic transition dynamics. In Engel et al. (2005a), a different, correlated noise model was shown to be useful for general MRPs. Let us define $R_t = (R(\mathbf{z}_0), \ldots, R(\mathbf{z}_t))$, $V_t = (V(\mathbf{z}_0), \ldots, V(\mathbf{z}_t))$, and $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))$ and also define the $t \times (t+1)$ matrix

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & -\gamma \end{bmatrix}.$$

In the white noise and correlated noise GPTD models, the noise covariance matrices $\boldsymbol{\Sigma}_t = \text{Cov}[N_t]$ are given, respectively, by

$$\begin{bmatrix} \sigma_R^2(\mathbf{z}_0) & 0 & \ldots & 0 \\ 0 & \sigma_R^2(\mathbf{z}_1) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_R^2(\mathbf{z}_{t-1}) \end{bmatrix}$$

$$\text{and} \quad \mathbf{H}_t \begin{bmatrix} \sigma_0^2 & 0 & \ldots & 0 \\ 0 & \sigma_1^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \sigma_t^2 \end{bmatrix} \mathbf{H}_t^\top.$$

The final component of the GPTD model remaining to be specified is the prior distribution of the GP V. This distribution is specified by prior mean and covariance functions $v_0(\mathbf{z})$ and $k(\mathbf{z}, \mathbf{z}')$, respectively.

Let us define $\boldsymbol{v}_t = (v_0(\mathbf{z}_0), \ldots, v_0(\mathbf{z}_t))^\top$. Employing ▸ Bayes' rule, the posterior distribution of $V(\mathbf{z})$ – the value function at some arbitrary query point $\mathbf{z}$ – is now given by

$$(V(\mathbf{z}) | R_{t-1} = \mathbf{r}_{t-1}) \sim \mathcal{N}\{\hat{V}_t(\mathbf{z}), P_t(\mathbf{z}, \mathbf{z}),$$

where

$$\begin{aligned} \hat{V}_t(\mathbf{z}) &= v_0(\mathbf{z}) + \mathbf{k}_t(\mathbf{z})^\top \boldsymbol{\alpha}_t, \quad P_t(\mathbf{z}, \mathbf{z}') \\ &= k(\mathbf{z}, \mathbf{z}') - \mathbf{k}_t(\mathbf{z})^\top \mathbf{C}_t \mathbf{k}_t(\mathbf{z}'), \\ \boldsymbol{\alpha}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} (\mathbf{r}_{t-1} - \mathbf{H}_t \boldsymbol{v}_t), \\ \mathbf{C}_t &= \mathbf{H}_t^\top (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^\top + \boldsymbol{\Sigma}_t)^{-1} \mathbf{H}_t. \end{aligned}$$

It is seen here that in order to compute the posterior distribution of V for arbitrary sets of query points, one only needs the vector $\boldsymbol{\alpha}_t$ and the matrix $\mathbf{C}_t$. Consequently, $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ are sufficient statistics for the posterior of V.

Algorithms 1 and 2 provide pseudocode for recursive computations of these sufficient statistics, in the deterministic transitions and general MDP models, respectively.

It can be seen that after observing t sample transitions, both the algorithms require storage quadratic in t (due to the matrix $\mathbf{C}_t$). The updates also require time quadratic in t due to matrix-vector products involving $\mathbf{C}_t$. These properties are unsatisfying from a practical point of view, since realistic RL problems typically require large amounts of data to learn. There are two general approaches for reducing the memory and time footprints of GPTD. One approach is to define parametric counterparts of the two GPTD models described earlier and derive the corre-

Algorithm 1 Recursive nonparametric GPTD for deterministic MDPs

```
Initialize $\boldsymbol{\alpha}_0 = 0$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$
for $t = 1, 2, \ldots$
    observe $\mathbf{z}_{t-1}, r_{t-1}, \mathbf{z}_t$
    $\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$
    $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{z}_t)$
    $\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$
    $\mathbf{c}_t = \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
    $d_t = r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}$
    $s_t = \sigma_{t-1}^2 + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t$
    $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
    $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t \mathbf{c}_t^\top$
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$
```

Algorithm 2 Recursive nonparametric GPTD for general MDPs

```
Initialize $\boldsymbol{\alpha}_0 = \mathbf{0}$, $\mathbf{C}_0 = 0$, $\mathcal{D}_0 = \{\mathbf{z}_0\}$, $\mathbf{c}_0 = \mathbf{0}$, $d_0 = 0$, $1/s_0 = 0$
for $t = 1, 2, \ldots$
    observe $\mathbf{z}_{t-1}, r_{t-1}, \mathbf{z}_t$
    $\mathbf{h}_t = (0, \ldots, 1, -\gamma)^\top$
    $\boldsymbol{\Delta}\mathbf{k}_t = \mathbf{k}_{t-1}(\mathbf{z}_{t-1}) - \gamma \mathbf{k}_{t-1}(\mathbf{z}_t)$
    $\Delta k_{tt} = k(\mathbf{z}_{t-1}, \mathbf{z}_{t-1}) - 2\gamma k(\mathbf{z}_{t-1}, \mathbf{z}_t) + \gamma^2 k(\mathbf{z}_t, \mathbf{z}_t)$
    $\mathbf{c}_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} \begin{pmatrix} \mathbf{c}_{t-1} \\ 0 \end{pmatrix} + \mathbf{h}_t - \begin{pmatrix} \mathbf{C}_{t-1}\boldsymbol{\Delta}\mathbf{k}_t \\ 0 \end{pmatrix}$
    $d_t = \frac{\gamma \sigma_{t-1}^2}{s_{t-1}} d_{t-1} + r_{t-1} - \boldsymbol{\Delta}\mathbf{k}_t^\top \boldsymbol{\alpha}_{t-1}$
    $s_t = \sigma_{t-1}^2 + \gamma^2 \sigma_t^2 - \frac{\gamma^2 \sigma_{t-1}^4}{s_{t-1}} + \Delta k_{tt} - \boldsymbol{\Delta}\mathbf{k}_t^\top \mathbf{C}_{t-1} \boldsymbol{\Delta}\mathbf{k}_t + \frac{2\gamma \sigma_{t-1}^2}{s_{t-1}} \mathbf{c}_{t-1}^\top \boldsymbol{\Delta}\mathbf{k}_t$
    $\boldsymbol{\alpha}_t = \begin{pmatrix} \boldsymbol{\alpha}_{t-1} \\ 0 \end{pmatrix} + \frac{\mathbf{c}_t}{s_t} d_t$
    $\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \frac{1}{s_t}\mathbf{c}_t \mathbf{c}_t^\top$
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{z}_t\}$
end for
return $\boldsymbol{\alpha}_t$, $\mathbf{C}_t$, $\mathcal{D}_t$
```

sponding recursive algorithms. If the number of independent parameters (i.e., the dimensionality of the hypothesis space $\mathcal{H}_\phi$) used to represent the value function is m, the memory and time costs of the algorithms become quadratic in m, rather than t. Another approach, which is based on an efficient sequential kernel sparsification method, allows us to selectively exclude terms from $\mathcal{D}_t$, while controlling the error incurred as a result. Here again (bounds on m in this case may be derived using arguments based on the finiteness of packing numbers of the hypothesis space; see Engel (2005) for details), if the size of $\mathcal{D}_t$ saturates at m, the memory and time costs of the resulting algorithms are quadratic in m. For the complete derivations, as well as detailed pseudocode of the corresponding algorithms, we refer the reader to Engel (2005).

Theory

In this section, we derive the two GPTD models mentioned above, explicitly stating the assumptions underlying each model.

MRPs with Deterministic Transitions

In the deterministic case, the Bellman equation (4) degenerates into

$$\bar{R}(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}'), \qquad (5)$$

where $\mathbf{z}'$ is the state or state–action pair succeeding $\mathbf{z}$, under the deterministic policy μ. We also assume that the noise in the rewards is independent and Gaussian, but not necessarily identically distributed. We denote the reward variance by $\sigma_R^2(\mathbf{z}) = \mathrm{Var}[R(\mathbf{z})]$. Formally, this means that the reward $R(\mathbf{z})$, at some $\mathbf{z}$, satisfies $R(\mathbf{z}) = \bar{R}(\mathbf{z}) + N(\mathbf{z})$ where $\bar{R}(\mathbf{z})$ is the mean reward for that state. Assume we have a sequence of rewards sampled along a sampled path $\boldsymbol{\xi}$. Then, at the ith time step, we have $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$. Using the random vectors R_t, V_t, and N_t defined earlier, we have $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$, where

$$\boldsymbol{\Sigma}_t = \mathrm{diag}(\sigma_R^2(\mathbf{z}_0), \ldots, \sigma_R^2(\mathbf{z}_{t-1})), \qquad (6)$$

and diag$(\cdot)$ denotes a diagonal matrix whose diagonal elements are the components of the argument vector. Writing the Bellman equations (5) for the points belonging to the sample path and substituting $R(\mathbf{z}_i) = \bar{R}(\mathbf{z}_i) + N(\mathbf{z}_i)$, we obtain the following set of t equations

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i),$$
$$i = 0, 1, \ldots, t-1.$$

This set of linear equations may be concisely written as

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{7}$$

General MRPs

Let us consider a decomposition of the discounted return D into its mean V and a zero-mean residual ΔV:

$$D(\mathbf{z}) = \mathbf{E}_\xi D(\mathbf{z}) + (D(\mathbf{z}) - \mathbf{E}_\xi D(\mathbf{z})) \overset{\text{def}}{=} V(\mathbf{z}) + \Delta V(\mathbf{z}). \tag{8}$$

This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return process D: For a known MDP model, V is a (deterministic) function, and the randomness in D is fully attributed to the intrinsic randomness in the trajectories generated by the MDP and policy pair, modeled by ΔV. On the other hand, in an MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV is deterministic (identically zero), and the randomness in D is due solely to the extrinsic Bayesian uncertainty, modeled by the random process V.

Substituting (8) into (2) and Rearranging, we get

$$R(\mathbf{z}) = V(\mathbf{z}) - \gamma V(\mathbf{z}') + N(\mathbf{z}, \mathbf{z}'),$$

where $\mathbf{z}' \sim p^\mu(\cdot|\mathbf{z})$ and

$$N(\mathbf{z}, \mathbf{z}') \overset{\text{def}}{=} \Delta V(\mathbf{z}) - \gamma \Delta V(\mathbf{z}'). \tag{9}$$

As before, we are provided with a sample path $\boldsymbol{\xi}$, and we may write the model Eqs. (9) for these samples, resulting in the following set of t equations:

$$R(\mathbf{z}_i) = V(\mathbf{z}_i) - \gamma V(\mathbf{z}_{i+1}) + N(\mathbf{z}_i, \mathbf{z}_{i+1})$$
$$\text{for } i = 0, \ldots, t-1.$$

Using our standard definitions for R_t, V_t, $\mathbf{H}_t$, and with $N_t = (N(\mathbf{z}_0, \mathbf{z}_1), \ldots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))^\top$, we again have

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{10}$$

In order to fully define a complete probabilistic generative model, we also need to specify the distribution of the noise process N_t. We model the residuals $\Delta V_t = (\Delta V(\mathbf{z}_0), \ldots, \Delta V(\mathbf{z}_t))^\top$ as random Gaussian noise. (This may not be a correct assumption in general; however, in the absence of any prior information concerning the distribution of the residuals, it is the *simplest* assumption we can make, since the Gaussian distribution possesses the highest entropy among all distributions with the same covariance. It is also possible to relax the Gaussianity requirement on both the prior and the noise. The resulting estimator may then be shown to be the *linear minimum mean-squared error* estimator for the value.) In particular, this means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{z}_i)$ is independently distributed. Denoting $\sigma_i^2 = \text{Var}[D(\mathbf{z}_i)]$, the distribution of ΔV_t is given by

$$\Delta V_t \sim \mathcal{N}(\mathbf{0}, \text{diag}(\boldsymbol{\sigma}_t)),$$

where $\boldsymbol{\sigma}_t = (\sigma_0^2, \sigma_1^2, \ldots, \sigma_t^2)^\top$. Since $N_t = \mathbf{H}_t \Delta V_t$, we have $N_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with

$$\boldsymbol{\Sigma}_t = \mathbf{H}_t \text{diag}(\boldsymbol{\sigma}_t) \mathbf{H}_t^\top$$
$$= \begin{bmatrix} \sigma_0^2 + \gamma^2\sigma_1^2 & -\gamma\sigma_1^2 & 0 & \ldots & 0 & 0 \\ -\gamma\sigma_1^2 & \sigma_1^2 + \gamma^2\sigma_2^2 & -\gamma\sigma_2^2 & 0 & \ldots & 0 \\ 0 & -\gamma\sigma_2^2 & \sigma_2^2 + \gamma^2\sigma_3^2 & \ddots & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \vdots & & \ddots & \ddots & -\gamma\sigma_{t-1}^2 \\ 0 & 0 & \ldots & 0 & -\gamma\sigma_{t-1}^2 & \sigma_{t-1}^2 + \gamma^2\sigma_t^2 \end{bmatrix}. \tag{11}$$

Applications

Any RL algorithm that requires policy evaluation as an algorithmic component can potentially use a GPTD algorithm for this task. In particular, this is true of algorithms based on Howard's policy iteration. In Engel et al. (2005a) and Engel (2005), it is shown how GPTD may be used to construct a SARSA-type algorithm (Rummery and Niranjan 1994; Sutton and Barto 1998), called GPSARSA. In Engel et al. (2005b), GPSARSA was used to learn control policies for a simulated Octopus arm. In Ghavamzadeh and Engel (2007), GPTD was used within a Bayesian actor–critic learning algorithm.

Future Directions

By virtue of the posterior covariance, GPTD algorithms compute a confidence measure (or, more precisely, Bayesian *credible intervals*) for their value estimates. So far, little use has been made of this additional information. Several potential uses of the posterior covariance may be envisaged:

1. It may be used to construct stopping rules for value estimation.
2. It may be used to guide exploration.
3. In the context of Bayesian actor–critic algorithms (Ghavamzadeh and Engel 2007), it may be used to control the size and direction of policy updates.

Further Reading

Yaakov Engel's doctoral thesis (Engel 2005) is currently the most complete reference to GPTD methods. Two conference papers (Engel et al. 2003, 2005a) provide a more concise view. The first of these introduces the GPTD model for deterministic MRPs, while the second introduces the general MDP model, as well as the GPSARSA algorithm. A forthcoming journal article will subsume these two papers and include some additional results, concerning the connection between GPTD and the popular TD(λ) and LSTD(λ) algorithms.

Recommended Reading

Bellman RE (1956) A problem in the sequential design of experiments. Sankhya 16:221–229
Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton
Bertsekas DP (1995) Dynamic programming and optimal control. Athena Scientific, Belmont
Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont
Boyan JA (1999) Least-squares temporal difference learning.
In: Proceedings of the 16th international conference on machine learning, Bled. Morgan Kaufmann, San Francisco, pp 49–56
Bradtke SJ, Barto AG (1996) Linear least-squares algorithms for temporal difference learning. Mach Learn 22:33–57
Dearden R, Friedman N, Andre D (1999) Model based Bayesian exploration. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence, Stockholm. Morgan Kaufmann, San Francisco, pp 150–159
Dearden R, Friedman N, Russell S (1998) Bayesian Q-learning. In: Proceedings of the fifteenth national conference on artificial intelligence, Madison. AAAI, Menlo Park, pp 761–768
Duff M (2002) Optimal learning: computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts, Amherst
Engel Y (2005) Algorithms and representations for reinforcement learning. PhD thesis, The Hebrew University of Jerusalem
Engel Y, Mannor S, Meir R (2003) Bayes meets Bellman: the Gaussian process approach to temporal difference learning. In: Proceedings of the 20th international conference on machine learning, Washington, DC. Morgan Kaufmann, San Francisco
Engel Y, Mannor S, Meir R (2005) Reinforcement learning with Gaussian processes. In: Proceedings of the 22nd international conference on machine learning, Bonn
Engel Y, Szabo P, Volkinshtein D (2005) Learning to control an Octopus arm with Gaussian process temporal difference methods. Technical report, Technion Institute of Technology. www.cs.ualberta.ca/~yaki/reports/octopus.pdf
Ghavamzadeh M, Engel Y (2007) Bayesian actor-critic algorithms. In: Ghahramani Z (ed) 24th international conference on machine learning, Corvalis. Omnipress, Corvallis
Howard R (1960) Dynamic programming and Markov processes. MIT, Cambridge

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

Kushner HJ, Yin CJ (1997) Stochastic approximation algorithms and applications. Springer, Berlin

Littman ML (1994) Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the 11th international conference on machine learning (ICML-94), New Brunswick. Morgan Kaufmann, New Brunswick, pp 157–163

Mannor S, Simester D, Sun P, Tsitsiklis JN (2004) Bias and variance in value function estimation. In: Proceedings of the 21st international conference on machine learning, Banff

Poupart P, Vlassis NA, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning. In: Proceedings of the twenty-third international conference on machine learning, Pittsburgh, pp 697–704

Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York

Rummery G, Niranjan M (1994) On-line Q-learning using connectionist systems. Technical report CUED/F-INFENG/TR 166, Cambridge University Engineering Department

Strens M (2000) A Bayesian framework for reinforcement learning. In: Proceedings of the 17th international conference on machine learning, Stanford. Morgan Kaufmann, San Francisco, pp 943–950

Sutton RS (1984) Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts, Amherst

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT, Cambridge

Tsitsiklis JN, Van Roy B (1996) An analysis of temporal-difference learning with function approximation. Technical report LIDS-P-2322. MIT, Cambridge

Wang T, Lizotte D, Bowling M, Schuurmans D (2005) Bayesian sparse sampling for on-line reward optimization. In: Proceedings of the 22nd international conference on machine learning, Bonn. ACM, New York, pp 956–963

Watkins CJCH (1989) Learning from delayed rewards. PhD thesis, King's College, Cambridge

Gaussian Processes

▶ Bayesian Nonparametric Models

Generality and Logic

▶ Logic of Generality

Generalization

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

A hypothesis, h, is a predicate that maps an instance to *true* or *false*. That is, if $h(x)$ is true, then x is hypothesized to belong to the concept being learned, the *target*. Hypothesis, h_1, is more general than or equal to h_2, if h_1 covers at least as many examples as h_2 (Mitchell, 1997). That is, $h_1 \geq h_2$ if and only if

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

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

Note that the *more general than* ordering is strongly related to *subsumption*.

Cross-References

▶ Classification
▶ Induction
▶ Learning as Search
▶ Logic of Generality
▶ Specialization
▶ Subsumption

Recommended Reading

Mitchell TM (1997) Machine learning. McGraw-Hill, New York

Generalization Bounds

Mark Reid
The Australian National University, Canberra, ACT, Australia

Synonyms

Inequalities; Sample complexity

Definition

In the theory of statistical machine learning, a generalization bound – or, more precisely, a generalization error bound – is a statement about the predictive performance of a learning algorithm or class of algorithms. Here, a learning algorithm is viewed as a procedure that takes some finite training sample of labeled instances as input and returns a hypothesis regarding the labels of all instances, including those which may not have appeared in the training sample. Assuming labeled instances are drawn from some fixed distribution, the quality of a hypothesis can be measured in terms of its risk – its incompatibility with the distribution. The performance of a learning algorithm can then be expressed in terms of the expected risk of its hypotheses given randomly generated training samples.

Under these assumptions, a generalization bound is a theorem, which holds for any distribution and states that, with high probability, applying the learning algorithm to a randomly drawn sample will result in a hypothesis with risk no greater than some value. This bounding value typically depends on the size of the training sample, an empirical assessment of the risk of the hypothesis on the training sample as well as the "richness" or "capacity" of the class of predictors that can be output by the learning algorithm.

Motivation and Background

Suppose we have built an e-mail classifier and then collected a random sample of e-mail labeled as "spam" or "not spam" to test it on. We notice that the classifier incorrectly labels 5 % of the sample. What can be said about the accuracy of this classifier when it is applied to new, previously unseen e-mail? If we make the reasonable assumption that the mistakes made on future e-mails are independent of mistakes made on the sample, basic results from statistics tell us that the classifier's true error rate will also be around 5 %.

Now suppose that instead of building a classifier by hand we use a learning algorithm to *infer* one from the sample. What can be said about the future error rate of the inferred classifier if it also misclassifies 5 % of the training sample? In general, the answer is "nothing" since we can no longer assume future mistakes are independent of those made on the training sample. As an extreme case, consider a learning algorithm that outputs a classifier that just "memorizes" the training sample – predicts labels for e-mail in the sample according to what appears in the sample – and predicts randomly otherwise. Such a classifier will have a 0 % error rate on the sample, however, if most future e-mail does not appear in the training sample the classifier will have a true error rate around 50 %.

To avoid the problem of memorizing or overfitting the training data it is necessary to restrict the "flexibility" of the hypotheses a learning algorithm can output. Doing so forces predictions made off the training set to be related to those made on the training set so that some form of generalization takes place. However, doing this can limit the ability of the learning algorithm to output a hypothesis with small risk. Thus, there is a classic and trade-off: the bias being the limits placed on how flexible the hypotheses can be versus the variance between the training and the true error rates (see bias variance decomposition).

By quantifying the notion of hypothesis flexibility in various ways, generalization bounds provide inequalities that show how the flexibility and empirical error rate can be traded off to control the true error rate. Importantly, these statements are typically probabilistic but distribution-independent – they hold for nearly all sets of training data drawn from a fixed but unknown distribution. When such a bound holds for a learning algorithm it means that, unless the choice of training sample was very unlucky, we can be confident that some form of generalization will take place. The first results of this kind were established by Vapnik and Chervonenkis (1971) about 40 years ago and the measure of hypothesis flexibility they introduced – the ▸ VC dimension (see below) – now bears their initials. A similar style of results were obtained independently by Valiant in 1984 in the Probably Approximately Correct, or ▸ PAC learning framework (Valiant 1984). These two lines of work were drawn together by Blumer et al. (1989) and now form the basis of what is known today as statistical learning theory.

Details

For simplicity, we restrict our attention to generalization bounds for binary ▶ classification problems such as the spam classification example above. In this setting *instances* (e.g., e-mail) from a set $\mathcal{X}$ are associated with *labels* from a set $\mathcal{Y} = \{-1, 1\}$ (e.g., indicating not spam/spam) and an *example* $z = (x, y)$ is a labeled instance from $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$. The association of instances to labels is assumed to be governed by some unknown distribution P over $\mathcal{Z}$.

A *hypothesis* h is a function that assigns labels $h(x) \in \mathcal{Y}$ to instances. The quality of a hypothesis is assessed via a *loss* function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty$, which assigns penalty $\ell(y, y')$ when h predicts the label $y' = h(x)$ for the example (x, y). For convenience, we will often combine the loss and hypothesis evaluation on an example $z = (x, y)$ by defining $\ell_h(z) = \ell(y, h(x))$. When examples are sampled from P the expected penalty, or *risk*

$$L_P(h) := \mathbb{E}_P[\ell_h(z)]$$

can be interpreted as a measure of how well h models the distribution P. A loss that is prevalent in classification is the *0–1 loss* $\ell^{0-1}(y, y') = [y \neq y']$ where $[p]$ is the indicator function for the predicate p. This loss simply assigns a penalty of 1 for an incorrect prediction and 0 otherwise. The associated 0–1 risk for h is the probability the prediction $h(x)$ disagrees with a randomly drawn sample (x, y) from P. Unless stated otherwise, the bounds discussed below are for the 0–1 loss only but, with care, can usually be made to hold with more general losses also.

Once a loss is specified, the goal of a learning algorithm is to produce a low-risk hypothesis based on a finite number of examples. Formally, a *learning algorithm* $\mathcal{A}$ is a procedure that takes a *training sample* $\mathbf{z} = (z_1, \ldots, z_n) \in \mathcal{Z}^n$ as input and returns a hypothesis $h = \mathcal{A}(\mathbf{z})$ with an associated *empirical risk*

$$\hat{L}_{\mathbf{z}}(h) := \frac{1}{n}\sum_{i=1}^{n} \ell_h(z_i).$$

In order to relate the empirical and true risks, a common assumption made in statistical learning theory is that the examples are drawn independently from P. In this case, a sample $\mathbf{z} = (z_1, \ldots, z_n)$ is a random variable from the product distribution P^n over $\mathcal{Z}^n$. Since the sample can be of arbitrary but finite size a learning algorithm can be viewed as a function $\mathcal{A} : \bigcup_{n=1}^{\infty} \mathcal{Z}^n \to \mathcal{H}$ where $\mathcal{H}$ is the algorithm's ▶ hypothesis space.

A generalization bound typically comprises several quantities: an empirical estimate of a hypothesis's performance $\hat{L}_{\mathbf{z}}(h)$; the actual (and unknown) risk of the hypothesis $L_P(h)$; a confidence term $\delta \in [0, 1]$; and some measure of the flexibility or *complexity* C of the hypotheses that can be output by learning algorithm. The majority of the bounds found in the literature fit the following template.

▶ *A generic generalization bound*: Let $\mathcal{A}$ be a learning algorithm, P some unknown distribution over $\mathcal{X} \times \mathcal{Y}$, and $\delta > 0$. Then, with probability at least $1 - \delta$ over randomly drawn samples $\mathbf{z}$ from P^n, the hypothesis $h = \mathcal{A}(\mathbf{z})$ has risk $L_P(h)$ no greater than $\hat{L}_{\mathbf{z}}(h) + \epsilon(\delta, C)$.

Of course, there are many variations, refinements, and improvements of the bounds presented below and not all fit this template. The bounds discussed below are only intended to provide a survey of some of the key ideas and main results.

Basic bounds: The penalties $\ell_h(z_i) := \ell(y_i, h(x_i))$ made by a fixed hypothesis h on a sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn from P^n are independent random variables. The law of large numbers guarantees (under some mild conditions) that their mean $\hat{L}_{\mathbf{z}}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell_h(z_i)$ converges to the true risk $L_P(h) = \mathbb{E}_P[\ell_h(z)]$ for h as the sample size increases and several inequalities from probability theory can be used to quantify this convergence. A key result is ▶ McDiarmid's inequality, which can be used to bound the deviation of a function of independent random variables from its mean. Since the 0–1 loss takes values in [0, 1], applying this result to the random variables $\ell_h(Z_i)$ gives

$$P^n(L_P(h) > \hat{L}_{\mathbf{z}}(h) + \varepsilon) \le \exp(-2n\varepsilon^2). \quad (1)$$

We can invert this and obtain an upper bound for the true risk that will hold on a given proportion of samples. That is, if we want $L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon$ to hold on at least $1 - \delta$ of the time on randomly drawn samples we can solve $\delta = \exp(-2n\varepsilon^2)$ for ε and obtain $\varepsilon = \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}$ so that

$$P^n\left(L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}\right) \ge 1 - \delta. \quad (2)$$

This simple bound lays the foundation for many of the subsequent bounds discussed below and is the reason for the ubiquity of the $\sqrt{\frac{\ln\frac{1}{\delta}}{n}}$-like terms.

A crucial observation to make about the above bound is that while it holds for any hypothesis h it does *not* hold for all $h \in \mathcal{H}$ *simultaneously*. That is, the samples for which the bounds hold for h_1 may be completely different to those which make the bound hold for h_2. Since a generalization bound must hold for all possible hypotheses output by a learning algorithm we need to extend the above analysis by exploiting additional properties of the hypothesis space $\mathcal{H}$.

In the simple case when there are only finitely many hypothesis, we use the *union bound*. This states that for any distribution P and any finite or countably infinite sequence of events $A_1, A_2 \ldots$ we have $P(\bigcup_i A_i) \le \sum_i P(A_i)$. For $\mathcal{H} = \{h_1, \ldots, h_m\}$ we consider the events $Z_h = \{\mathbf{z} \in \mathcal{Z}^n : L_P(h) > \hat{L}_{\mathbf{z}}(h) + \epsilon\}$ when samples of size n give empirical risks for h that are least ε smaller than its true risk. Using the union bound and (1) on these events gives

$$P^n\left(\bigcup_{h\in\mathcal{H}} Z_h(n,\varepsilon)\right) \le \sum_{i=1}^{m} P^n(Z_h(n,\varepsilon)) \\ = m \cdot \exp(-2n\varepsilon^2).$$

This is a bound on the probability of drawing a training sample from P^n such that *every* hypothesis has a true risk that is ε larger than its empirical risk. Inverting this inequality by setting $\delta = m\exp(-2n\varepsilon^2)$ yields the following bound.

Finite class bound: Suppose $\mathcal{A}$ has finite hypothesis class $\mathcal{H} = \{h_1, \ldots, h_m\}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{2n}}. \quad (3)$$

It is instructive to compare this to the single hypothesis bound in (2) and note the bound is weakened by the additional term $|\mathcal{H}|$.

Since the union bound also holds for countable sets of events this style of bound can be extended from finite hypothesis classes to countable ones. To do this requires a slight modification of the above argument and the introduction of a distribution π over a countable hypothesis space $\mathcal{H} = \{h_1, h_2, \ldots\}$, which is chosen before any samples are seen. This distribution can be interpreted as a prior belief or preference over the hypotheses in $\mathcal{H}$. Letting $\delta(h) = \delta \cdot \pi(h)$ in the bound (2) implies that for each $\mathcal{H}$ we have

$$P^n\left(L_P(h) < \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\delta\cdot\pi(h)}}{2n}}\right) < \delta\cdot\pi(h).$$

Thus, applying the countable union bound to the union of these events over all of $\mathcal{H}$, and noting that $\sum_{h\in\mathcal{H}} \delta.\ \pi(h) = \delta$ since π is a distribution over $\mathcal{H}$, gives use the following bound:

Countable class bound: Suppose μ is a probability distribution over a finite or countably infinite hypothesis space $\mathcal{H}$. Then with probability at least $1-\delta$ over draws of $\mathbf{z}$ from P^n the following bound holds for all $h \in \mathcal{H}$

$$L_P(h) \le \hat{L}_{\mathbf{z}}(h) + \sqrt{\frac{\ln\frac{1}{\pi(h)} + \ln\frac{1}{\delta}}{2n}}. \quad (4)$$

Although the finite and countable class bounds are proved using very similar techniques (indeed, the former can be derived from the latter by choosing $\pi(h) = \frac{1}{|\mathcal{H}|}$), they differ in the type of penalty they introduce for simultaneously bounding all the hypotheses in $\mathcal{H}$. In (3), the penalty $\ln|\mathcal{H}|$ is purely a function of the size of the class whereas in (4) the penalty $\ln\frac{1}{\pi(h)}$ varies with h. These two different styles of bound can be seen as templates for the two main classes of bounds discussed below: the hypothesis-independent bounds of the next section and the hypothesis-dependent bounds in the section on PAC-Bayesian bounds. The main conceptual leap from here is the extension of the arguments above to non-countable hypothesis classes.

Class complexity bounds: A key result in extending the notion of size or complexity in the above bounds to more general classes of hypotheses is the *symmetrization lemma*. Intuitively, it is based on the observation that if the empirical risks for different samples are frequently near the true risk then they will also be near each other. Formally, it states that for any $\varepsilon > 0$ such that $n\varepsilon^2 \geq 2$ we have

$$P^n\left(\sup_{h\in\mathcal{H}} |L_P(h) - \hat{L}_{\mathbf{z}}(h)| > \varepsilon\right)$$

$$\leq 2P^{2n}\left(\sup_{h\in\mathcal{H}} |\hat{L}_{\mathbf{z}'}(h) - \hat{L}_{\mathbf{z}}(h)| > \frac{\varepsilon}{2}\right). \quad (5)$$

Thus, to obtain a bound on the difference between empirical and true risk it suffices to bound the difference in empirical risks on two independent samples $\mathbf{z}$ and $\mathbf{z}'$, both drawn from P^n. This is useful since the maximum difference $\sup_{h\in\mathcal{H}} |\hat{L}_{\mathbf{z}'}(h) - \hat{L}_{\mathbf{z}}(h)|$ is much easier to handle than the difference involving $L_P(h)$ as the former term only evaluates losses on the points in $\mathbf{z}$ and $\mathbf{z}'$ while the latter takes into account the entire space $\mathcal{Z}$.

To study these restricted evaluations, we define the restriction of a function class $\mathcal{F}$ to the sample $\mathbf{z}$ by $\mathcal{F}_{\mathbf{z}} = \{(f(z_1), \ldots, f(z_n)) : f \in \mathcal{F}\}$. Since the empirical risk $\hat{L}_{\mathbf{z}}(h) = \frac{1}{n}\Sigma_{i=1}^n \ell_h(z_i)$ only depends on the values of the loss functions ℓ_h on samples from $\mathbf{z}$ we define the *loss class* $\mathbb{L} = \ell_{\mathcal{H}} = \{\ell_h : h \in \mathcal{H}\}$ and consider its restriction $\mathbb{L}_{\mathbf{z}}$ as well as the restriction $\mathcal{H}_{\mathbf{z}}$ of the hypothesis class it is built upon. As we will see, the measures of complexity of these two classes are closely related.

One such complexity measure is arrived at by examining the size of a restricted function class $\mathcal{F}_{\mathbf{z}}$ as the size of the sample $\mathbf{z}$ increases. The *growth function* or ▶ shattering coefficient for the function class $\mathcal{F}$ is defined as the maximum number of distinct values the vectors in $\mathcal{F}_{\mathbf{z}}$ can take given a sample of size n : $S_n(\mathcal{F}) = \sup_{\mathbf{z}\in\mathcal{Z}^n} |\mathcal{F}_{\mathbf{z}}|$. In the case of binary classification with a 0–1 loss, it is not hard to see that the growth functions for both $\mathbb{L}$ and $\mathcal{F}$ are equal, that is, $S_n(\mathrm{L}) = S_n(\mathcal{H})$, and so they can be used interchangeably. Applying a union bound argument to (1) as in the previous bounds guarantees that $P^n(\sup_{h\in\mathcal{H}} |L_P(h) - \hat{L}_{\mathbf{z}}(h)| > \varepsilon) \leq 2S_n(\mathcal{H})\exp(-n\varepsilon^2/8)$ and by inversion we obtain the following generalization bound for arbitrary hypothesis classes $\mathcal{H}$:

Growth function bound: For all $\delta > 0$, a draw of $\mathbf{z}$ from P^n will, with probability at least $1 - \delta$, satisfy for all $h \in \mathcal{H}$

$$L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + 2\sqrt{\frac{2\ln S_n(\mathcal{H}) + 2\ln\frac{2}{\delta}}{n}}. \quad (6)$$

One conclusion that can be immediately drawn from this bound is that the shattering coefficient must grow sub-exponentially for the bound to provide any meaningful guarantee. If the class $\mathcal{H}$ is so rich that hypotheses from it can fit all 2^n possible label combinations – if $S_n(\mathcal{H}) = 2^n$ for all n – then the term $\sqrt{2\ln S_n(\mathcal{H})/n} > 1$ and so (6) just states $L_P(h) \leq 1$. Therefore, to get nontrivial bounds from (6) there needs to exist some value d for which $S_n(\mathcal{H}) < 2^n$ whenever $n > d$.

VC dimension: This desired property of the growth function is exactly what is captured by the ▶ VC dimension $VC(\mathcal{H})$ of a hypothesis class $\mathcal{H}$. Formally, it is defined as $VC(\mathcal{H}) = \max\{n \in \mathbb{N} : S_n(\mathcal{H}) = 2^n\}$ and is infinite if no finite maximum exists. Whether or not the VC dimension is finite plays a central role in the consistency of empirical risk minimization techniques. Indeed, it is possible to show that using ERM on a hypothesis class $\mathcal{H}$ is consistent if and only if

$VC(\mathcal{H}) < \infty$. This is partly due to *Sauer's lemma*, which shows that when a hypothesis class $\mathcal{H}$ has finite VC dimension $VC(\mathcal{H}) = d_{\mathcal{H}} < \infty$ its growth function is eventually polynomial in the sample size. Specifically, for all $n \geq d_{\mathcal{H}}$ the growth function satisfies $S_n(\mathcal{H}) \leq \left(\frac{en}{d_{\mathcal{H}}}\right)^{d_{\mathcal{H}}}$. By substituting this result into the Growth Function Bound (6) we obtain the following bound, which shows how the VC dimension plays a role that is analogous to the size a hypothesis class in the finite case.

► *VC dimension bound*: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$ with finite VC dimension $d_{\mathcal{H}}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{z})$ satisfies

$$L_P(h) \leq \hat{L}_z(h) + 2\sqrt{\frac{2d_{\mathcal{H}} \ln\left(\frac{2en}{d_{\mathcal{H}}}\right) + 2\ln\frac{2}{\delta}}{n}}. \tag{7}$$

There are many other bounds in the literature that are based on the VC dimension. See the Recommended Reading for pointers to these.

Rademacher averages: Rademacher averages are a second kind of measure of complexity for uncountable function classes and can be used to derive more refined bounds than those above. These averages arise naturally by treating as a random variable the sample-dependent quantity $M_{\mathcal{F}}(\mathbf{z}) = \sup_{f\in\mathcal{F}}[\mathbb{E}_P[f] - \mathbb{E}_{\mathbf{z}}[f]]$. This is just the largest difference taken over all $f \in \mathcal{F}$ between its true mean $\mathbb{E}_P[f]$ and its empirical mean $\mathbb{E}_{\mathbf{z}}[f] := \frac{1}{|\mathbf{z}|}\sum_{i=1}^{|\mathbf{z}|} f(z_i)$. For a loss class $\mathbb{L} = \ell_{\mathcal{H}}$ a bound on this maximum difference – $M_{\mathbb{L}}(\mathbf{z}) \leq B$ – immediately gives a generalization bound of the form $L_P(h) \leq \hat{L}_{\mathbf{z}}(h) + B$. Since $M_{\mathcal{F}}(\mathbf{z})$ is a random variable, McDiarmid's inequality can be used to bound its value in terms of its expected value plus the usual $\sqrt{\frac{\ln\frac{1}{\delta}}{2n}}$ term. Applying symmetrization it can then be shown that this expected value satisfies

$$\mathbb{E}_{P^n}[M_{\mathcal{F}}(\mathbf{z})] \leq \mathbb{E}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i(f(z_i') - f(z_i))\right] \leq 2R_n(\mathcal{F})$$

where the right-hand expectation is taken over two independent samples $\mathbf{z}, \mathbf{z}' \sim P^n$ and the *Rademacher variables* $\rho_1, \ldots, \rho_n$. These are independent random variables, each with equal probability of taking the values -1 or 1, that give their name to the *Rademacher average*

$$R_n(\mathcal{F}) = \mathrm{E}\left[\sup_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \rho_i f(z_i)\right].$$

Intuitively, this quantity measures how well the functions in $\mathcal{F}$ can be chosen to align with randomly chosen labels ρ_i. The Rademacher averages for the loss class $\mathbb{L}$ and the hypothesis class $\mathcal{H}$ are closely related. For 0–1 loss, it can be shown they satisfy $R_n(\mathbb{L}) = \frac{1}{2}R_n(\mathcal{H})$.

Putting all the above steps together gives the following bounds.

Rademacher bound: Suppose $\mathcal{A}$ has hypothesis class $\mathcal{H}$. Then with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n the hypothesis $h = \mathcal{A}(\mathbf{Z})$ satisfies

$$L_P(h) \leq \hat{L}_z(h) + R_n(\mathcal{H}) + \sqrt{\frac{\ln\frac{1}{\delta}}{2n}}. \tag{8}$$

This bound is qualitatively different to the Growth Function and VC bounds above as the Rademacher average term is *distribution-dependent* whereas the other complexity terms are purely a function of the hypothesis space. Indeed, it is possible to bound the Rademacher average in terms of the VC dimension and obtain the VC bound (7) from (8). Furthermore, the Rademacher average is closely related to the minimum empirical risk via $R_n(\mathcal{H}) = 1 - 2\mathbb{E}[\inf_{h\in\mathcal{H}} \hat{L}_{\mathbf{x},\rho}(h)]$ where $\hat{L}_{\mathbf{x},\rho}(h)$ is the empirical risk of h for the randomly labeled sample $\mathbf{z} = ((x_1, \rho_1), \ldots, (x_n, \rho_n))$. Thus, in principle, $R_n(\mathcal{H})$ could be estimated for a given learning problem using standard ERM methods.

The Rademacher bound can be further refined so that the complexity term is *data-dependent* rather than distribution-dependent. This is done by noting that the Rademacher average $R_n\mathcal{F} = \mathbb{E}[\hat{R}_{\mathbf{z}}(\mathcal{F})]$ where $\hat{R}_{\mathbf{z}}(\mathcal{F})$ is the *empirical Rademacher average* for $\mathcal{F}$ conditioned on the sample $\mathbf{z}$. Applying McDiarmid's inequality to the difference between $\hat{R}_{\mathbf{z}}(\mathcal{F})$ and its mean gives a sample-dependent bound:

Empirical Rademacher bound: Under the same conditions as the Rademacher bound, the following holds with probability $1 - \delta$:

$$L_P(h) \le \hat{L}_z(h) + \hat{R}_z(\mathcal{H}) + 3\sqrt{\frac{\ln \frac{2}{\delta}}{2n}}. \quad (9)$$

PAC-Bayesian bounds: All the bounds in the previous section provide bounds on deterministic hypotheses, which include complexity terms that are functions of the entire hypothesis space. PAC-Bayesian bounds differ from these in two ways: they provide bounds on nondeterministic hypotheses – labels may be predicted for instances stochastically; and their complexity terms are *hypothesis-dependent*. The term "Bayesian" given to these bounds refers to the use of a distribution over hypotheses that is used to define the complexity term. This distribution can be interpreted as a prior belief over the efficacy of each hypothesis before any observations are made.

Nondeterministic hypotheses are modeled by assuming that a distribution μ over $\mathcal{H}$ is used to randomly draw a deterministic hypothesis $h \in \mathcal{H}$ to predict $h(x)$ each time a new instance x is seen. Such a strategy is called a *Gibbs hypothesis* for μ. Since its behavior is defined by the distribution μ, we will abuse our notation slightly and define its loss on the example z to be $\ell_\mu(z) := \mathbb{E}_{h\sim\mu}[\ell_h(z)]$. Similarly, the true risk and empirical risk for a Gibbs hypothesis are, respectively, defined to be $L_P(\mu) := \mathbb{E}_{h\sim\mu}[L_P(h)]$ and $\hat{L}_{\mathbf{z}}(\mu) := \mathbb{E}_{h\sim\mu}[\hat{L}_{\mathbf{z}}(h)]$. As with the earlier generalization bounds, the aim is to provide guarantees about the difference between $L_P(\mu)$ and $\hat{L}_{\mathbf{z}}(\mu)$. In the case of 0–1 loss, $p := L_P(\mu) \in [0, 1]$ is just the probability of the Gibbs hypothesis for μ misclassifying an example and $q := \hat{L}_{\mathbf{z}}(\mu) \in [0, 1]$ can be thought of as an estimate of p. However, unlike the earlier bounds on the difference between the true and estimated risk, PAC-Bayesian bounds are expressed in terms the *Kullback–Leibler (KL) divergence*. For the values $p, q \in [0, 1]$ this is defined as $kl(q\|p) := q \ln \frac{q}{p} + (1-q)\ln\frac{1-q}{1-p}$ and for distributions μ and π over the hypothesis space $\mathcal{H}$ we write $KL(\mu \parallel \pi) := \int_{\mathcal{H}} \ln \frac{d\mu}{d\pi} d\mu$. Using these definitions, the most common PAC-Bayesian bound states the following.

Theorem (PAC-Bayesian bound): For all choices of the distribution π over $\mathcal{H}$ made prior to seeing any examples, the Gibbs hypothesis defined by μ satisfies

$$kl(L_P(\mu), \hat{L}_{\mathbf{z}}(\mu)) \le \frac{KL(\mu \parallel \pi) + \ln \frac{n+1}{\delta}}{n} \quad (10)$$

with probability at least $1 - \delta$ over draws of $\mathbf{z}$ from P^n.

This says that the difference (as measured by *kl*) between the true and empirical risk for the Gibbs hypothesis based on μ is controlled by two terms: a *complexity* term $\frac{KL(\mu\|\pi)}{n}$ and a *sampling* term $\frac{\ln\frac{n+1}{\delta}}{n}$, both of which converge to zero as n increases. To make connections with the previous bounds more apparent, we can weaken (10) using the inequality $kl(q \parallel p) \ge 2(p-q)^2$ to get the following bound that holds under the same assumptions:

$$L_P(\mu) \le \hat{L}_{\mathbf{z}}(\mu) + \sqrt{\frac{KL(\mu \parallel \pi) + \ln\frac{n+1}{\delta}}{2n}}.$$

The sampling term is similar to the ubiquitous estimation penalty in the earlier bounds but with an additional $\ln(n+1)/n$. The complexity term is a measure of the complexity of the Gibbs hypothesis for μ *relative* to the distribution π. Intuitively, $KL(\cdot \| \pi)$ can be thought of as a parametrized family of complexity measures where hypotheses from a region where π is large are "cheap" and those where π is small are "expensive". Information theoretically, it is the expected number of extra bits required to code hypotheses drawn from μ using a code based on π instead of a code based on μ. It is for these reasons the PAC-Bayes bound is said to demonstrate the importance of choosing a good prior. If the Gibbs hypothesis μ, which minimizes $\hat{L}_z(\mu)$ is also "close" to π then the bound will be tight.

Unlike the other bounds discussed above, PAC-Bayesian bounds are in terms of the complexity of single meta-classifiers rather than the complexity of classes. Furthermore, for specific base hypothesis classes such as margin classifiers used by SVMs it is possible to get hypothesis-specific bounds via the PAC-Bayesian bounds. These are typically much tighter than the VC or Rademacher bounds.

Other bounds: While the above bounds are landmarks in statistical learning theory there is obviously much more territory that has not been covered here. For starters, the VC bounds for classification can be refined by using more sophisticated results from empirical process theory such as the Bernstein and Variance-based bounds. These are discussed in Sect. 5 of Boucheron et al. (2005). There are also other distribution- and sample-dependent complexity measures that are motivated differently to Rademacher averages. For example, the *VC entropy* (see Sect. 4.5 of Bousquet et al. 2004) is a distribution-dependent measure obtained by averaging $|\mathcal{F}_{\mathbf{z}}|$ with respect to the sample distribution rather than taking supremum in the definition of the shattering coefficient.

Moving beyond classification, bounds for regression problems have been studied in depth and have similar properties to those for classification. These bounds are obtained by essentially discretizing the function spaces. The growth function is replaced by what is known as a *covering number* but the essence of the bounds remain the same. The reader is referred to Herbrich and Williamson (2002) for a brief discussion and Anthony and Bartlett (1999) for more detail.

There are a variety of bounds that, unlike those above, are algorithm-specific. For example, the regularized empirical risk minimization performed by SVMs has been analyzed within an *algorithmic stability* framework. As discussed in Boucheron et al. (2005) and Herbrich and Williamson (2002), hypotheses are considered stable if their predictions are not varied too much when a single training example is perturbed. Two other algorithm-dependent frameworks include the *luckiness* and *compression* frameworks, both summarized in Herbrich and Williamson (2002). The former gives bounds in terms of an a priori measure of luckiness – how well a training sample aligns with biases encoded in an algorithm – while the latter considers algorithms, like SVMs, which base hypotheses on key examples within a training sample.

Recently, there has been work on a type of algorithm-dependent, relative bound called *reductions* (see Beygelzimer et al. 2008 for an overview). By transforming inputs and outputs for one type of problem (e.g., probability estimation) into a different type of problem (e.g., classification), bounds for the former can be given in terms of bounds for the latter while making very few assumptions. This opens up a variety of avenues for applying existing results to new learning tasks.

Cross-References

- ▶ Classification
- ▶ Empirical Risk Minimization
- ▶ Hypothesis Space
- ▶ Loss
- ▶ PAC Learning
- ▶ Regression
- ▶ Regularization
- ▶ Structural Risk Minimization
- ▶ VC Dimension

Recommended Reading

As mentioned above, the uniform convergence bounds by Vapnik and Chervonenkis (1971) and the PAC framework of Valiant (1984) were the first generalization bounds for statistical learning. Ideas from both were synthesized and extended by Blumer et al. (1989). The book by Kearns and Vazirani (1994) provides a good overview of the early PAC-style bounds while Vapnik's comprehensive book (Vapnik 1998), and Anthony and Bartlett's book (1999) cover classification and regression bounds involving the VC dimension. Rademacher averages were first considered as an alternative to VC dimension in the context of learning theory by Koltchinskii and

Panchenko (2001) and were refined and extended by Bartlett and Mendelson (2003) who provide a readable overview. Early PAC-Bayesian bounds were established by McAllester (1999) based on an earlier PAC analysis of Bayesian estimators by Shawe-Taylor and Williamson (1997). Applications of the PAC-Bayesian bound to SVMs are discussed in Langford's tutorial on prediction theory (Langford 2005) and recent paper by Banerjee (2006) provides an information theoretic motivation, a simple proof of the bound in (11), as well as connections with similar bounds in online learning.

There are several well-written surveys of generalization bounds and learning theory in general. Herbrich and Williamson (2002) present a unified view of VC, compression, luckiness, PAC-Bayesian, and stability bounds. In a very readable introduction to statistical learning theory, Bousquet et al. (2004) provide good intuition and concise proofs for all but the PAC-Bayesian bounds presented above. That introduction is a good companion for the excellent but more technical survey by Boucheron et al. (2005) based on tools from the theory of empirical processes. The latter paper also provides a wealth of further references and a concise history of the development of main techniques in statistical learning theory.

Anthony M, Bartlett PL (1999) Neural network learning: theoretical foundations. Cambridge University Press, Cambridge

Banerjee A (2006) On Bayesian bounds. In: ICML'06: proceedings of the 23rd international conference on machine learning, Pittsburgh, pp 81–88

Bartlett PL, Mendelson S (2003) Rademacher and Gaussian complexities: risk bounds and structural results. J Mach Learn Res 3:463–482

Beygelzimer A, Langford J, Zadrozny B (2008) Machine learning techniques – reductions between prediction quality metrics. In: Zhen L, Cathy HX (eds) Performance modeling and engineering. Springer, New York, pp 3–28

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

Boucheron S, Bousquet O, Lugosi G (2005) Theory of classification: a survey of some recent advances. ESAIM Probab Stat 9:323–375

Bousquet O, Boucheron S, Lugosi G (2004) Introduction to statistical learning theory. Volume 3176 of lecture notes in artificial intelligence. Springer, Berlin, pp 169–207

Herbrich R, Williamson RC (2002) Learning and generalization: theory and bounds. In: Arbib M (ed) Handbook of brain theory and neural networks, 2nd ed. MIT Press, Cambridge

Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press, Cambridge

Koltchinskii V (2001) Rademacher penalties and structural risk minimization. IEEE Trans Inf Theory 47(5):1902–1914

Langford J (2005) Tutorial on practical prediction theory for classification. J Mach Learn Res 6(1):273–306

McAllester DA (1999) Some PAC-Bayesian theorems. Mach Learn 37(3):355–363

Shawe-Taylor J, Williamson RC (1997) A PAC analysis of a Bayesian estimator. In: Proceedings of the tenth annual conference on computational learning theory. ACM, New York, p 7

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1142

Vapnik VN (1998) Statistical learning theory. Wiley, New York

Vapnik VN, Chervonenkis AY (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab Appl 16(2):264–280

Generalization Performance

The *generalization performance* of a learning algorithm refers to the performance on ▸ out-of-sample data of the models learned by the algorithm.

Cross-References

▸ Algorithm Evaluation

Generalized Delta Rule

▸ Backpropagation

General-to-Specific Search

When searching a hypothesis space, a general-to-specific search starts from the most general hypothesis and expands the search by specialization. See ▸ Learning as Search.

Generative and Discriminative Learning

Bin Liu[1] and Geoffrey I. Webb[2]
[1]Monash University, Clayton, VIC, Australia
[2]Faculty of Information Technology, Monash University, Victoria, Australia

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability P(y,**x**) or to any classification learning process that classifies by using estimates of the ▶ prior probability P(y) and the conditional probability P(**x**|y) (Jaakkola and Haussler 1999; Jaakkola et al. 1999; Ng and Jordan 2002; Lasserre et al. 2006; Bishop 2007), where y is a class and **x** is a description of an object to be classified. Given such models or estimates, it is possible to generate synthetic objects from the joint distribution. Generative learning contrasts to *discriminative learning* in which a model or estimate of P(y|**x**) is formed without reference to an explicit estimate of any of P(y, **x**), P(**x**), or P(**x**|y).

It is also common to categorize as discriminative approaches based on a decision function that directly map from input **x** onto the output y (such as ▶ support vector machines, ▶ neural networks, and ▶ decision trees), where the decision risk is minimized without estimation of P(y, **x**), P(**x**|y), or P(y|**x**) (Jaakkola and Haussler 1999).

The standard exemplar of generative learning is ▶ naïve Bayes and that of discriminative learning is ▶ logistic regression Another important contrasting pair is the generative ▶ hidden Markov model and discriminative ▶ conditional random field.

It is widely accepted that generative learning works well when samples are rare, while discriminative learning has better asymptotic error performance (Ng and Jordan 2002).

Motivation and Background

Efron (1975) provides an early examination of the generative/discriminative distinction. Efron performs an empirical comparison of the efficiency of the generative ▶ linear discriminant analysis (LDA) and discriminative ▶ logistic regression. His results show that logistic regression has 30 % less efficiency than LDA, which means the discriminative approach is 30 % slower to reach its asymptotic error than the generative approach.

Ng and Jordan (2002) give a theoretical discussion of the efficiency of generative ▶ naïve Bayes and discriminative ▶ logistic regression. This is an interesting pair because they both form linear models of forms that are directly equivalent to one another, the only substantive difference being the manner in which they parameterize those models. Their result shows that logistic regression converges toward its asymptotic error in order n samples, while naïve Bayes converges in order log$n$ samples. While logistic regression converges much slower than naïve Bayes, it has lower asymptotic error than naïve Bayes. These results suggest that it is desirable to use a generative approach when training data is scarce and to use a discriminative approach when there is enough training data. However, it is worth noting that the generative/discriminative distinction is not the only difference in how these two algorithms parameterize their models. Whereas logistic regression seeks to directly fit its model to the discriminative objective, P(y|**x**), naïve Bayes does not directly fit P(y, **x**). Instead it fits its model to P(y) and each P(x_i|y) (where x_i is an individual attribute), making the simplifying attribute independence assumption.

Recent research into the generative/discriminative learning distinction has concentrated on the area of hybrids of generative and discriminative learning as well as generative learning and discriminative learning in structured data learning or semi-supervised learning context.

In hybrid approaches, researchers seek to obtain the merits of both generative learning and discriminative learning. Some examples include the Fisher kernel for discriminative learning (Jaakkola and Haussler 1999), max-ent discriminative learning (Jaakkola et al. 1999), and principled hybrids of generative and discriminative models (Lasserre et al. 2006; Zaidi et al. 2014).

In structured data learning, the output data have dependent relationships As an example of generative learning, hidden Markov models are used in structured data problems which need sequential decisions. The discriminative analogue is conditional random field models. Another example of discriminatively structured learning is max-margin Markov networks (Taskar et al. 2004).

In ▶ semi-supervised learning, co-training and multiview learning are usually applied to generative learning (Blum and Mitchell 1998). It is less straightforward to apply semi-supervised learning in traditional discriminative learning, since $P(y|\mathbf{x})$ is estimated by ignoring $P(\mathbf{x})$. Examples of semi-supervised learning methods in discriminative learning include transductive SVM, Gaussian processes, information regularization, and graph-based methods (Chapelle et al. 2006).

Recommended Reading

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of the eleventh annual conference on computational learning theory

Chapelle O, Schölkopf B, Zien A (2006) Semi-supervised learning. The MIT Press, Cambridge

Efron B (1975) The efficiency of logistic regression compared to normal discriminant analysis. J Am Stat Assoc 70(352):892–898

Jaakkola TS, Haussler D (1999) Exploiting generative models in discriminative classifiers. Adv Neural Inf Process Syst 11:487–493

Jaakkola T, Meila M, Jebara T (1999) Maximum entropy discrimination. Adv Neural Inf Process Syst 12

Lasserre JA, Bishop CM, Minka TP (2006) Principled hybrids of generative and discriminative models. In: IEEE conference on computer vision and pattern recognition

Ng AY, Jordan MI (2002) On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. Adv Neural Inf Process Syst 2(14):841–848

Taskar B, Guestrin C, Koller D (2004) Max-margin Markov networks. Adv Neural Inf Process Syst 16

Zaidi N, Carman M, Webb GI (2014) Naive-Bayes inspired effective pre-conditioner for speeding-up logistic regression. In: Proceedings of the 14th IEEE international conference on data mining, ICDM-14, pp 1097–1102

Generative Learning

Definition

Generative learning refers alternatively to any classification learning process that classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or to any classification learning process that classifies by using estimates of the prior probability P(y) and the conditional probability $P(\mathbf{x}|y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Given such models or estimates it is possible to generate synthetic objects from the joint distribution. Generative learning contrasts to discriminative learning in which a model or estimate of $P(y|\mathbf{x})$ is formed without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$.

Cross-References

▶ Generative and Discriminative Learning

Genetic and Evolutionary Algorithms

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

Definitions

There are many variations of genetic algorithms (GA). Here, wedescribe a simple scheme to introduce some of the key terms in genetic and evolutionary algorithms. See the main entry on ▶ Evolutionary Algorithms for references to specific methods.

In genetic learning, we assume that there is a population of individuals, each of which represents a candidate problem solver for a given task. GAs can be thought of as a family of general purpose search methods that are capable of solving a broad range of problems from optimization and scheduling to robot control. Like evolution, genetic algorithms test each individual from the population and only the fittest survive to reproduce for the next generation. The algorithm creates new generations until at least one individual is found that can solve the problem adequately.

Each problem solver is a *chromosome*. A position, or set of positions in a chromosome is called a *gene*. The possible values (from a fixed set of symbols) of a gene are known as *alleles*. For example, a simple genetic algorithm may define the set of symbols to be $\{0, 1\}$, and chromosome lengths are fixed. The most critical problem in applying a genetic algorithm is in finding a suitable encoding of the examples in the problem domain to a chromosome. A good choice of representation will make the search easier by limiting the size of the search space. A poor choice will result in a large search space. Choosing the size of the population can be problematic since a small population size provides an insufficient sample over the space of solutions for a problem and large population requires extensive evaluation and will be slow.

Each iteration in a genetic algorithm is called a *generation*. Each chromosome in a population is used to solve a problem. Its performance is evaluated and the chromosome is given a rating of fitness. The population is also given an overall fitness rating based on the performance of its members. The fitness value indicates how close a chromosome or population is to the required solution.

New sets of chromosomes are produced from one generation to the next. Reproduction takes place when selected chromosomes from one generation are recombined with others to form chromosomes for the next generation. The new ones are called *offspring*. Selection of chromosomes for reproduction is based on their fitness values. The average fitness of the population may also be calculated at the end of each generation. The strategy must be modified if too few or too many chromosomes survive. For example, at least 10 % and at most 60 % must survive.

Genetic Operators

Operators that recombine the selected chromosomes are called *genetic operators*. Two common operators are *crossover* and *mutation*. Crossover exchanges portions of a pair of chromosomes at a randomly chosen point called the crossover point. Some Implementations have more than one crossover point. For example, if there are two chromosomes, X and Y:

$$X = 1001\,01011,\ Y = 1110\,10010$$

and the crossover point is after position 4, the resulting offspring are:

$$O1 = 100110010,\ O2 = 1110\,01011$$

Offspring produced by crossover cannot contain information that is not already in the population, so an additional operator, *mutation*, is required. Mutation generates an offspring by randomly changing the values of genes at one or more gene positions of a selected chromosome. For example, if the following chromosome,

$$Z = 100101011$$

is mutated at positions 2, 4, and 9, then the resulting offspring is:

$$O = 110001010$$

The number of offspring produced for each new generation depends on how members are introduced so as to maintain a fixed population size. In a *pure* replacement strategy, the whole population is replaced by a new one. In an *elitist* strategy, a proportion of the population survives to the next generation.

Cross-References

▸ Evolutionary Algorithms

Genetic Attribute Construction

▶ Evolutionary Feature Selection and Construction

Genetic Clustering

▶ Evolutionary Clustering

Genetic Feature Selection

▶ Evolutionary Feature Selection and Construction

Genetic Grouping

▶ Evolutionary Clustering

Genetic Neural Networks

▶ Neuroevolution

Genetic Programming

Moshe Sipper
Ben-Gurion University, Beer-Sheva, Israel

Abstract

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution, used to solve complex problems.

Genetic programming is a subclass of ▶ evolutionary algorithms, wherein a population of individual programs is evolved. The main mechanism behind genetic programming is that of a ▶ genetic algorithm, namely, the repeated cycling through four operations applied to the entire population: evaluate–select–crossover–mutate. Starting with an initial population of randomly generated programs, each individual is evaluated in the domain environment and assigned a fitness value representing how well the individual solves the problem at hand. Being randomly generated, the first-generation individuals usually exhibit poor performance. However, some individuals are better than others, that is, as in nature, variability exists, and through the mechanism of selection, these have a higher probability of being selected to parent the next generation. The size of the population is finite and usually constant.

See ▶ Evolutionary Games for a more detailed explanation of genetic programming.

Genetics-Based Machine Learning

▶ Classifier Systems

Gibbs Sampling

Gibbs Sampling is a heuristic inference algorithm for ▶ Bayesian networks. See ▶ Graphical Models for details.

Gini Coefficient

The Gini coefficient is an empirical measure of classification performance based on the area under an ROC curve (AUC). Attributed to the Italian statistician Corrado Gini (1884–1965), it can be calculated as $2 \cdot \{\text{AUC}\} - 1$ and thus takes values in the interval $[-1, 1]$, where 1 indicates perfect ranking performance and -1 indicates that all negatives are ranked before all positives. See ▶ ROC Analysis.

Gram Matrix

▶ Kernel Matrix

Grammar Learning

▶ Grammatical Inference

Grammatical Inference

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

Synonyms

Grammar learning

Definition

Grammatical inference is concerned with inferring grammars from positive (and possibly negative) examples (Angluin 1978; Korfiatis and Paliouras 2008; Sakakibara 2005). A context-free grammar (CFG) $\mathcal{G}$ (equivalent to a push-down finite-state automaton) is described by a four tuple $(\mathcal{Q}, \mathcal{E}, \delta, \Sigma)$:

- Σ is the alphabet of *terminal* symbols, upon which the grammar is defined.
- The pair $(\mathcal{Q}, \mathcal{E})$ defines a graph, where $\mathcal{Q}$ is the set of nodes (states), and $\mathcal{E}$ is the set of edges (production rules). $\mathcal{Q}$ includes one *starting* node q_0 and a set $\mathcal{Q}_f$ ($\mathcal{Q}_f \subset \mathcal{Q}$) of final or *accepting* nodes.
- Every edge in $\mathcal{E}$ is labeled by one or several letters in Σ, expressed through mapping $\delta : \mathcal{E} \mapsto 2^{\Sigma}$.
- Let $\mathcal{L}(\mathcal{G})$ denote the language associated to the grammar. Each string s in $\mathcal{L}(\mathcal{G})$ is generated along a random walk in the graph, starting in q_0 with an initially empty s. Upon traversing edge e, one symbol from $\delta(e)$ is concatenated to s. The walk ends upon reaching a final node ($e \in \mathcal{Q}_f$).

A CFG is determinist if all pairs of edges (q, q') and (q, q'') $(q' \neq q'')$ bear different labels $(\delta(q, q') \bigcap \delta(q, q'') = \emptyset)$.

One generalizes a given CFG by applying one or several operators, among the following: (1) introducing additional nodes and edges, (2) turning a node into an accepting one, and (3) *merging* two nodes q and q'. In the latter case, some non-determinism can be introduced (if some edges (q, r) and (q', r') have label(s) in common); enforcing a deterministic generalization is done using the *recursive determinization operator* (e.g., merging nodes r and r').

In general, grammatical inference proceeds as follows (Lang et al. 1998; Oncina and Garcia 1992). Let S be the set of positive examples, strings on alphabet Σ. The *prefix tree acceptor* (PTA), a most specific generalization of S, is constructed by associating to each character of every string a distinct node and applying the determinization operator. This PTA is thereafter iteratively generalized by merging a pair of nodes. Well-known grammar learners are RPNI (Oncina and Garcia 1992) and BLUE-FRINGE (Lang et al. 1998). RPNI uses a depth first search strategy and merges the pair of nodes which are closest to the start node, such that their deterministic generalization does not cover any negative example. BLUE-FRINGE uses a beam search from a candidate list, selecting the pair of nodes to be merged after the *evidence-driven* state merging (EDSM) criterion, i.e., such that their generalization involves a minimal number of final states.

Recommended Reading

Angluin D (1978) On the complexity of minimum inference of regular sets. Inf Control 39:337–350
Korfiatis G, Paliouras G (2008) Modeling web navigation using grammatical inference. Appl Artif Intell 22(1–2):116–138

Lang KJ, Pearlmutter BA, Price RA (1998) Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: ICGI'98: proceedings of the 4th international colloquium on grammatical inference. Springer, Berlin, pp 1–12

Oncina J, Garcia P (1992) Inferring regular languages in polynomial update time. In: Pattern recognition and image analysis, vol 1. World Scientific, Singapore/New Jersey, pp 49–61

Sakakibara Y (2005) Grammatical inference in bioinformatics. IEEE Trans Pattern Anal Mach Intell 27(7):1051–1062

Grammatical Tagging

▶ POS Tagging

Graph Clustering

Charu C. Aggarwal
IBM T. J. Watson Research Center, Hawthorne, NY, USA

Synonyms

Minimum cuts; Network clustering; Spectral clustering; Structured data clustering

Definition

Graph clustering refers to ▶ clustering of data in the form of graphs. Two distinct forms of clustering can be performed on graph data. Vertex clustering seeks to cluster the nodes of the graph into groups of densely connected regions based on either edge weights or edge distances. The second form of graph clustering treats the graphs as the objects to be clustered and clusters these objects on the basis of similarity. The second approach is often encountered in the context of structured or XML data.

Motivation and Background

Graph clustering is a form of ▶ graph mining that is useful in a number of practical applications including marketing, customer segmentation, congestion detection, facility location, and XML data integration (Lee et al. 2002). The graph clustering problems are typically defined into two categories:

- *Node clustering algorithms*: Node clustering algorithms are generalizations of multidimensional clustering algorithms in which we use functions of the multidimensional data points in order to define the distances. In the case of graph clustering algorithms, we associate numerical values with the edges. These numerical values need not satisfy traditional properties of distance functions such as the triangle inequality. We use these distance values in order to create clusters of nodes. We note that the numerical value associated with a given node may either be a distance value or a similarity value. Correspondingly, the objective function associated with the partitioning may either be minimized or maximized. We note that the problem of minimizing the intercluster similarity for a fixed number of clusters essentially reduces to the problem of *graph partitioning* or the *minimum multiway cut problem*. This is also referred to the problem of mining dense graphs and pseudocliques. Recently, the problem has also been studied in the database literature as that of *quasi-clique determination*. In this problem, we determine groups of nodes which are "almost cliques." In other words, an edge exists between any pair of nodes in the set with a high probability. A closely related problem is that of determining *shingles* (Gibson et al. 2005). Shingles are defined as those subgraphs which have a large number of common links. This is particularly useful for massive graphs which contain a large number of nodes. In such cases, a min-hash approach (Gibson et al. 2005) can be used in order to summarize the structural behavior of the underlying graph.

- *Graph clustering algorithms*: In this case, we have a (possibly large) number of graphs which need to be clustered based on their underlying structural behavior. This problem is challenging because of the need to match the structures of the underlying graphs and use these structures for clustering purposes. Such algorithms are discussed both in the context of classical graph data sets as well as semistructured data. In the case of semistructured data, the problem arises in the context of a large number of documents which need to be clustered on the basis of the underlying structure and attributes. It has been shown by Aggarwal et al. (2007) that the use of the underlying document structure leads to significantly more effective algorithms.

This chapter will discuss the different kinds of clustering algorithms and their applications. Each section will discuss a particular class of clustering algorithms and the different approaches which are commonly used for this class.

Graph Clustering as Minimum Cut

The graph clustering problem can be related to the minimum-cut and graph partitioning problems. In this case, it is assumed that the underlying graphs have weights on the edges. It is desired to partition the graphs in such a way so as to minimize the weights of the edges across the partitions. In general, we would like to partition the graphs into k groups of nodes. However, since the special case $k = 2$ is efficiently solvable, we would like to first provide a special discussion for this case. This version is polynomially solvable, since it is the mathematical dual of the maximum-flow problem. This problem is also referred to as the *minimum-cut problem*.

The minimum-cut problem is defined as follows. Consider a graph $G = (N, A)$ with node set N and edge set A. The node set N contains the source s and sink t. Each edge $(i, j) \in A$ has a weight associated with it which is denoted by u_{ij}. We note that the edges may be either undirected or directed, though the undirected case is often much more relevant for connectivity applications. We would like to partition the node set N into two groups S and $N - S$. The set of edges such that one end lies in S and the other lies in $N - S$ is denoted by $C(S, N - S)$. We would like to partition the node set N into two sets S and $N - S$, such that the sum of the weights in $C(S, N - S)$ is minimized. In other words, we would like to minimize $\sum_{(i,j)} \in C_{(S,N-S)} u_{ij}$. This is the unrestricted version of the minimum-cut problem. We will examine two variations of the minimum-cut problem:

- We wish to determine the global minimum $s - t$ cut with no restrictions on the membership of nodes to different partitions.
- We wish to determine the minimum $s - t$ cut, in which one partition contains the source node s and the other partition contains the sink node t.

It is easy to see that the former problem can be solved by using repeated applications of the latter algorithm. By fixing s and choosing different values of the sink t, it can be shown that the global minimum cut may be effectively determined.

It turns out that the maximum-flow problem is the mathematical dual of the minimum-cut problem. In the maximum-flow problem, we assume that the weight u_{ij} is a capacity of the edge (i, j). Each edge is allowed to have a *flow*x_{ij} which is at most equal to the capacity u_{ij}. Each node other than the source s and sink t is assumed to satisfy the *flow conservation property*. In other words, for each node $i \in N$ we have

$$\sum_{j:(i,j)\in A} x_{ij} = \sum_{j:(j,i)\in A} x_{ji}.$$

We would like to maximize the total flow originating from the source and reaching the sink t, subject to the above constraints. The maximum-flow problem is solved with the use of a variety of *augmenting path* and *preflow push algorithms*. Details of different kinds of algorithms may be found in the work by Ahuja et al. (1992).

A closely related problem to the minimum $s-t$ cut problem is that of determining a *global minimum cut* in an undirected graph. This particular case is more efficient than that of finding the $s-t$ minimum cut. One way of determining a minimum cut is by using a contraction-based edge-sampling approach. While the previous technique is applicable to both the directed and undirected versions of the problem, the contraction-based approach is applicable only to the undirected version of the problem. Furthermore, the contraction-based approach is applicable only for the case in which the weight of each edge is $u_{ij} = 1$. While the method can easily be extended to the weighted version by varying the edge-sampling probability, the polynomial running time bounds discussed by Tsay et al. (1999) do not apply to this case. The contraction approach is a probabilistic technique in which we successively sample the edges in order to collapse nodes into larger sets of nodes. By successively sampling different sequences of edges and picking the optimum value (Tsay et al. 1999), it is possible to determine a global minimum cut. The broad idea of the contraction-based approach is as follows. We pick an edge randomly in the graph and contract its two end points into a single node. We remove all the self-loops which are created as a result of the contraction. We may also create some parallel edges, which are allowed to remain, since they influence the sampling probability (Alternatively, we may replace parallel edges by a single edge of weight which is equal to the number of parallel edges. We use this weight in order to bias the sampling process.) of contractions. The process of contraction is repeated until we are left with two nodes. We note that each of this pair of "super-nodes" corresponds to a set of nodes in the original data. These two sets of nodes provide us with the final minimum cut. We note that the minimum cut will survive in this approach, if none of the edges in the minimum cut are sampled during the contraction. It has been shown by Tsay et al. that by using repeated contraction of the graph to a size of $\sqrt{n}$ nodes, it is possible to obtain a correct solution with high probability in $O(n^2)$ time.

Graph Clustering as Multiway Graph Partitioning

The *multiway graph partitioning problem* is significantly more difficult, and is NP-hard (Kernighan and Lin 1970). In this case, we wish to partition a graph into $k > 2$ components, so that the total weight of the edges whose ends lie in different partitions is minimized. A well-known technique for graph partitioning is the Kerninghan-Lin algorithm (Kernighan and Lin 1970). This classical algorithm is based on hill climbing (or more generally neighborhood-search technique) for determining the optimal graph partitioning. Initially, we start off with a random cut of the graph. In each iteration, we exchange a pair of vertices in two partitions to see if the overall cut value is reduced. In the event that the cut value is reduced, then the interchange is performed. Otherwise, we pick another pair of vertices in order to perform the interchange. This process is repeated until we converge to a optimal solution. We note that this optimum may not be a global optimum, but may only be a local optimum of the underlying data. The main variation in different versions of the Kerninghan-Lin algorithm is the policy which is used for performing the interchanges on the vertices. Some examples of strategies which may be used in order to perform the interchange are as follows:

- We randomly pick a pair of vertices and perform the interchange, if it improves the underlying solution quality.
- We test all possible vertex-pair interchanges (or a sample of possible interchanges), and pick the interchange which improves the solution by the greatest amount.
- A k-interchange is one in which a sequence of k interchanges are performed at one time. We can test any k-interchange and perform it, if it improves the underlying solution quality.
- We can pick the optimal k-interchange from a sample of possibilities.

We note that the use of more sophisticated strategies allows a better improvement in the objective function for each interchange, but also requires more time for each interchange. For example, the determination of an optimal k-interchange requires much more time than a straightforward interchange. This is a natural trade-off which may work out differently depending upon the nature of the application at hand. Furthermore, the choice of the policy also affects the likelihood of getting stuck at a local optimum. For example, the use of k-interchange techniques are far less likely to result in local optimum for larger values of k. In fact, by choosing the best interchange across all possible values of k it is possible to ensure that a global optimum is always reached. On the other hand, it is increasingly difficult to implement the algorithm efficiently with increasing value of k. This is because the time complexity of the interchange increases exponentially with the value of k.

Graph Clustering with k-Means

Two well-known (and related) techniques for clustering in the context of multidimensional data (Jain and Dubes 1998) are the k-medoid and k-means algorithms. In the k-medoid algorithm (for multidimensional data), we sample a small number of points from the original data as *seeds* and assign every other data point from the clusters to the closest of these seeds. The closeness may be defined based on a user-defined objective function. The objective function for the clustering is defined as the sum of the corresponding distances of data points to the corresponding seeds. In the next iteration, the algorithm interchanges one of the seeds for another randomly selected seed from the data, and checks if the quality of the objective function improves upon performing the interchange. If this is indeed the case, then the interchange is accepted. Otherwise, we do not accept the interchange and try another sample interchange. This process is repeated, until the objective function does not improve over a predefined number of interchanges. A closely related method is the k-means method. The main difference with the k-medoid method is that we do not use representative points from the original data after the first iteration of picking the original seeds. In subsequent iterations, we use the centroid of each cluster as the seed set for the next iteration. This process is repeated until the cluster membership stabilizes.

A method has been proposed by Rattigan et al. (2007), which uses the characteristics of both the k-means and k-medoids algorithms. As in the case of the conventional partitioning algorithms, it picks k graph nodes as seeds. The main differences from the conventional algorithms are in terms of computation of distances (for assignment purposes), and in determination of subsequent seeds. A natural distance function for graphs is the *geodesic distance*, or the smallest number of hops between a pair of nodes. In order to determine the seed set for the next iteration, we compute the *local closeness centrality* for each cluster, and use the corresponding node as the sample seed. Thus, while this algorithm continues to use seeds from the original data set (as in the k-medoids algorithm), it uses intuitive ideas from the k-means algorithms in order to determine the identity of these seeds.

Graph Clustering with the Spectral Method

Eigenvector techniques are often used in multidimensional data in order to determine the underlying correlation structure in the data. It is natural to question as to whether such techniques can also be used for the more general case of graph data. It turns out that this is indeed possible with the use of a method called *spectral clustering*.

In the spectral clustering method, we make use of the node-node adjacency matrix of the graph. For a graph containing n nodes, let us assume that we have an $n \times n$ adjacency matrix, in which the entry (i, j) correspond to the weight of the edge between the nodes i and j. This essentially corresponds to the similarity between nodes i and j. This entry is denoted by w_{ij}, and the corresponding matrix is denoted by W. This matrix is assumed to be symmetric, since we are working with undirected graphs. Therefore, we assume that $w_{ij} = w_{ji}$ for any pair (i, j). All diagonal entries of the matrix W are assumed to be 0. As discussed earlier, the aim of any node partitioning algorithm is to minimize (a function of) the weights across the partitions. The spectral clustering method constructs this minimization function in terms of the matrix structure of the adjacency matrix and another matrix which is referred to as the *degree matrix*.

The *degree matrix D* is simply a diagonal matrix in which all entries are zero except for the diagonal values. The diagonal entry d_{ii} is equal to the sum of the weights of the incident edges. In other words, the entry d_{ij} is defined as follows:

$$d_{ij} = \sum_{j=1}^{n} w_{ij}, \quad i = j,$$
$$0, \quad i \neq j.$$

We formally define the *Laplacian matrix* as follows: (*Laplacian matrix*): The Laplacian matrix L is defined by subtracting the weighted adjacency matrix from the degree matrix. In other words, we have

$$L = D - W.$$

This matrix encodes the structural behavior of the graph effectively and its eigenvector behavior can be used in order to determine the important clusters in the underlying graph structure. It can be shown that the Laplacian matrix L is positive semidefinite i.e., for any n-dimensional row vector $f = [f_1 \dots fn]$ we have $f \cdot L \cdot f^T \geq 0$. This can be easily shown by expressing L in terms of its constituent entries which are a function of the corresponding weights w_{ij}. Upon expansion, it can be shown that

$$f \cdot L \cdot f^{\mathrm{T}} = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_J)^2.$$

The Laplacian matrix L is positive semidefinite. Specifically, for any n-dimensional row vector $f = [f_1 \dots f_n]$, we have

$$f \cdot L \cdot f^T = (1/2) \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (f_i - f_J)^2.$$

At this point, let us examine some *interpretations* of the vector f in terms of the underlying graph partitioning. Let us consider the case in which each f_i is drawn from the set $\{0, 1\}$, and this determines a two-way partition by labeling each node either 0 or 1. The particular partition to which the node i belongs is defined by the corresponding label. Note that the expansion of the expression $f \cdot L \cdot f^T$ from the above relationship simply represents the sum of the weights of the edges across the partition defined by f. Thus, the determination of an appropriate value of f for which the function $f \cdot L \cdot f^T$ is minimized also provides us with a good node partitioning. Unfortunately, it is not easy to determine the *discrete values* of f which determine this optimum partitioning. Nevertheless, we will see later in this section that even when we restrict f to real values, this provides us with the intuition necessary to create an effective partitioning.

An immediate observation is that the indicator vector $f = [1 \dots 1]$ is an eigenvector with a corresponding eigenvalue of 0. We note that $f = [1 \dots 1]$ must be an eigenvector, since L is positive semidefinite and $f \cdot L \cdot f^T$ can be 0 only for eigenvectors with 0 eigenvalues. This observation can be generalized further in order to determine the number of connected components in the graph. We make the following observation.

The number of (*linearly independent*) *eigenvectors with zero eigenvalues for the Laplacian matrix L is equal to the number of connected components in the underlying graph.*

We observe that connected components are the most obvious examples of clusters in the graph. Therefore, the determination of eigenvectors corresponding to zero eigenvalues provides us the information about (relatively rudimentary set of) clusters. Broadly speaking, it may not be possible to glean such clean membership behavior from the other eigenvectors. One of the problems is that other than this particular rudimentary set of eigenvectors (which correspond to the connected components), the vector components of the other eigenvectors are drawn from the real domain rather than the discrete {0, 1} domain. Nevertheless, because of the nature of the natural interpretation of $f \cdot L \cdot f^T$ in terms of the weights of the edges across nodes with very differing values of f_i, it is natural to cluster together the nodes for which the values of f_i are as similar as possible across any particular eigenvector on an average. This provides us with the intuition necessary to define an effective spectral clustering algorithm, which partitions the data set into k clusters for any arbitrary value of k. The algorithm is as follows:

- Determine the k eigenvectors with the smallest eigenvalues. Note that each eigenvector has as many components as the number of nodes. Let the component of the jth eigenvector for the ith node be denoted by p_{ij}.
- Create a new data set with as many records as the number of nodes. The ith record in this data set corresponds to the ith node and has k components. The record for this node is simply the eigenvector components for that node, which are denoted by $pi_1 \ldots p_{ik}$.
- Since we would like to cluster nodes with similar eigenvector components, we use any conventional clustering algorithm (e.g., k-means) in order to create k clusters from this data set. Note that the main focus of the approach was to create a *transformation* of a structural clustering algorithm into a more conventional multidimensional clustering algorithm, which is easy to solve. The particular choice of the multidimensional clustering algorithm is orthogonal to the broad spectral approach.

The above algorithm provides a broad framework for the spectral clustering algorithm. The input parameter for the above algorithm is the number of clusters k. In practice, a number of variations are possible in order to tune the quality of the clusters which are found. More details on the different methods which can be used for effective spectral graph clustering may be found in Chung (1997).

Graph Clustering as Quasi-clique Detection

A different way of determining massive graphs in the underlying data is that of determining *quasi-cliques*. This technique is different from many other partitioning algorithms, in that it focuses on definitions which maximize the edge densities *within a partition*, rather than minimizing the edge densities across partitions. A clique is a graph in which every pair of nodes are connected by an edge. A quasi-clique is a relaxation on this concept, and is defined by imposing a lower bound on the degree of each vertex in the given set of nodes. Specifically, we define a γ-quasi-clique is as follows:

A k-graph ($k \geq 1$) G is a γ-quasi-clique if the degree of each node in the corresponding subgraph of vertices is at least $\gamma \cdot k$.

The value of γ always lies in the range (0, 1]. We note that by choosing $\gamma = 1$, this definition reverts to that of standard cliques. Choosing lower values of γ allows for the relaxations which are more true in the case of real applications. This is because we rarely encounter complete cliques in real applications, and at least some edges within a dense subgraph would always be missing. A vertex is said to be critical if its degree in the corresponding subgraph is the smallest integer which is at least equal to $\gamma \cdot k$.

The earliest piece of work on this problem is from Abello et al. (2002). The work of Abello et al. (2002) uses a greedy randomized adaptive search algorithm, GRASP, to find a quasi-clique with the maximum size. A closely related problem is that of finding *frequently occurring cliques* in *multiple data sets*. In other words, when multiple graphs are obtained from different data sets, some dense subgraphs occur frequently together in the different data sets. Such graphs help in determining *important dense patterns of behavior in different data sources*. Such techniques find applicability in mining important patterns in graphical representations of customers. The techniques are also helpful in mining cross-graph quasi-cliques in gene expression data. An efficient algorithm for determining cross graph quasi-cliques was proposed by Pei et al. (2005).

The main restriction of the work proposed by Pei et al. (2005) is that the support threshold for the algorithms is assumed to be 100 %. This restriction has been relaxed in subsequent work (Zeng et al. 2007). The work by Zeng et al. (2007) examines the problem of mining frequent, closed quasi-cliques from a graph database with arbitrary support thresholds.

Graph Clustering as Dense Subgraph Determination

A closely related problem is that of dense subgraph determination in massive graphs. This problem is frequently encountered in large graph data sets. For example, the problem of determining large subgraphs of web graphs was studied by Gibson et al. (2005). The broad idea in the min-hash approach is to represent the outlinks of a particular node as sets. Two nodes are considered similar if they share many outlinks. Thus, consider a node A with an outlink set S_A, and a node B with outlink set S_B. Then the similarity between the two nodes is defined by the *Jaccard coefficient*, which is defined as $\frac{S_A \cap S_B}{S_A \cup S_B}$. We note that explicit enumeration of all the edges in order to compute this can be computationally inefficient. Rather, a *min-hash approach* is used in order to perform the estimation. This *min-hash approach* is as follows. We sort the universe of nodes in a random order. For any set of nodes in random sorted order, we determine the first node $First(A)$ for which an outlink exists from A to $First(A)$. We also determine the first node $First(B)$ for which an outlink exists from B to $First(B)$. It can be shown that the Jaccard coefficient is an unbiased estimate of the probability that $First(A)$ and $First(B)$ are the same nodes. By repeating this process over different permutations over the universe of nodes, it is possible to accurately estimate the Jaccard coefficient. This is done by using a constant number of permutations c of the node order. The actual permutations are implemented by associated c different randomized hash values with each node. This creates c sets of hash values of size n. The sort-order for any particular set of hash-values defines the corresponding permutation order. For each such permutation, we store the minimum node index of the outlink set. Thus, for each node, there are c such minimum indices. This means that, for each node, a fingerprint of size c can be constructed. By comparing the fingerprints of two nodes, the Jaccard coefficient can be estimated. This approach can be further generalized with the use of every s element set contained entirely with S_A and S_B. Thus, the above description is the special case when s is set to 1. By using different values of s and c, it is possible to design an algorithm which distinguishes between two sets that are above or below a certain threshold of similarity.

The overall technique by Gibson et al. (2005) first generates a set of c shingles of size s for each node. The process of generating the c shingles is extremely straightforward. Each node is processed independently. We use the min-wise hash function approach in order to generate subsets of size s from the outlinks at each node. This results in c subsets for each node. Thus, for each node, we have a set of c shingles. Thus, if the graph contains a total of n nodes, the total size of this shingle fingerprint is $n \times c \times sp$, where sp is the space required for each shingle. Typically, sp will be $O(s)$, since each shingle contains s nodes. For each distinct shingle thus created, we can create a list of nodes which contain it. In general, we would like to determine groups of shingles which contain a large number of common nodes. In order to do so, the method by Gibson et al. performs a second-order shingling in which the meta-shingles are created from the shingles. Thus, this further compresses the graph in a data structure of size $c \times c$. This is essentially a constant-size data structure. We note that this group of meta-shingles have the property that they contain a large number of common nodes. The dense subgraphs can then be extracted from these meta-shingles. More details on this approach may be found in the work by Gibson et al.

Clustering Graphs as Objects

In this section, we will discuss the problem of clustering *entire graphs* in a *multigraph database*, rather than examining the node clustering problem within a single graph. Such situations are often encountered in the context of XML data, since each XML document can be regarded as a structural record, and it may be necessary to create clusters from a large number of such objects. We note that XML data is quite similar to graph data in terms of how the data is organized structurally. The attribute values can be treated as graph labels and the corresponding semistructural relationships as the edges. In has been shown by Aggarwal et al. (2007), Dalamagas et al. (2005), Lee et al. (2002), and Lian et al. (2004) that this structural behavior can be leveraged in order to create effective clusters.

Since we are examining entire graphs in this version of the clustering problem, the problem simply boils down to that of clustering arbitrary *objects*, where the objects in this case have structural characteristics. Many of the conventional algorithms discussed by Jain and Dubes (1998) (such as k-means type partitional algorithms and hierarchical algorithms) can be extended to the case of graph data. The main changes required in order to extend these algorithms are as follows:

- Most of the underlying classical algorithms typically use some form of distance function in order to measure similarity. Therefore, we need appropriate measures in order to define similarity (or distances) between structural objects.
- Many of the classical algorithms (such as k-means) use *representative objects* such as centroids in critical intermediate steps. While this is straightforward in the case of multidimensional objects, it is much more challenging in the case of graph objects. Therefore, appropriate methods need to be designed in order to create representative objects. Furthermore, in some cases it may be difficult to create representatives in terms of single objects. We will see that it is often more robust to use *representative summaries* of the underlying objects.

There are two main classes of conventional techniques, which have been extended to the case of structural objects. These techniques are as follows:

- *Structural distance-based approach*: This approach computes structural distances between documents and uses them in order to compute clusters of documents. One of the earliest work on clustering tree structured data is the *XClust algorithm* (Lee et al. 2002), which was designed to cluster XML schemas in order for efficient integration of large numbers of document type definitions (DTDs) of XML sources. It adopts the agglomerative hierarchical clustering method which starts with clusters of single DTDs and gradually merges the two most similar clusters into one larger cluster. The similarity between two DTDs is based on their element similarity, which can be computed according to the semantics, structure, and context information of the elements in the corresponding DTDs. One of the shortcomings of the XClust algorithm is that it does not make full use of the structure information of the DTDs, which is quite important in the context of clustering tree-like structures. The method by Chawathe (1999) computes similarity measures based on the structural edit-distance between documents. This edit-distance is used in order to compute the distances between clusters of documents.

  S-GRACE is hierarchical clustering algorithm (Lian et al. 2004). In the work by Lian et al., an XML document is converted to a structure graph (or s-graph), and the distance between two XML documents is defined according to the number of the common element-subelement relationships, which can capture better structural similarity relationships than the tree edit-distance in some cases (Lian et al.).

- *Structural summary-based approach*: In many cases, it is possible to create summaries from the underlying documents. These summaries are used for creating groups of documents which are similar to these summaries. The first summary-based approach for clustering XML documents was presented by Dalamagas et al. (2005). In the work by Dalamagas et al., the XML documents are modeled as rooted, ordered labeled trees. A framework for clustering XML documents by using structural summaries of trees is presented. The aim is to improve algorithmic efficiency without compromising cluster quality.

  A second approach for clustering XML documents is presented by Aggarwal et al. (2007). This technique is a partition-based algorithm. The primary idea in this approach is to use frequent-pattern mining algorithms in order to determine the summaries of frequent structures in the data. The technique uses a k-means type approach in which each cluster center comprises a set of frequent patterns which are local to the partition for that cluster. The frequent patterns are mined using the documents assigned to a cluster center in the last iteration. The documents are then further reassigned to a cluster center based on the average similarity between the document and the newly created cluster centers from the local frequent patterns. In each iteration the document assignment and the mined frequent patterns are iteratively reassigned until the cluster centers and document partitions converge to a final state. It has been shown by Aggarwal et al. that such a structural summary-based approach is significantly superior to a similarity function-based approach, as presented by Chawathe (1999). The method is also superior to the structural approach by Dalamagas et al. (2005) because of its use of more robust representations of the underlying structural summaries.

Conclusions and Future Research

In this chapter, we presented a review of the commonly known algorithms for clustering graph data. The problem of clustering graphs has been widely studied in the literature, because of its application to a variety of data mining and data management problems. Graph clustering algorithms are of two types:

- *Node clustering algorithms*: In this case, we attempt to partition the graph into groups of clusters so that each cluster contains groups of nodes which are densely connected. These densely connected groups of nodes may often provide significant information about how the entities in the underlying graph are interconnected with one another.
- *Graph clustering algorithms*: In this case, we have complete graphs available, and we wish to determine the clusters with the use of the structural information in the underlying graphs. Such cases are often encountered in the case of XML data, which are commonly encountered in many real domains.

We provided an overview of the different clustering algorithms available and the trade-offs with the use of different methods. The major challenges that remain in the area of graph clustering are as follows:

- *Clustering massive data sets*: In some cases, the data sets containing the graphs may be so large that they may be held only on disk. For example, if we have a dense graph containing 10^7 nodes, then the number of edges may be as high as 10^{13}. In such cases, it may not even be possible to store the graph effectively on disk. In the cases in which the graph can be stored on disk, it is critical that the algorithm should be designed in order to take the disk-resident behavior of the underlying data into account. This is especially challenging in the case of graph data sets, because the structural behavior of the graph interferes with our ability to process the edges sequentially for many applications. In the cases in which the graph is too large to store on disk, it is essential to design summary structures which can effectively store the underlying structural behavior of the graph. This stored summary can then be used effectively for graph clustering algorithms.

- *Clustering graph streams*: In this case, we have large graphs which are received as edge streams. Such graphs are more challenging, since a given edge cannot be processed more than once during the computation process. In such cases, summary structures need to be designed in order to facilitate an effective clustering process. These summary structures may be utilized in order to determine effective clusters in the underlying data. This approach is similar to the case discussed above in which the size of the graph is too large to store on disk.

In addition, techniques need to be designed for interfacing clustering algorithms with traditional database management techniques. In order to achieve this goal, effective representations and query languages need to be designed for graph data. This is a new and emerging area of research, and can be leveraged upon in order to further improve the effectiveness of graph algorithms.

Cross-References

▶ Group Detection
▶ Partitional Clustering

Recommended Reading

Abello J, Resende MG, Sudarsky S (2002) Massive quasi-clique detection. In: Proceedings of the 5th Latin American symposium on theoretical informatics (LATIN). Springer, Berlin, pp 598–612
Aggarwal C, Ta N, Feng J, Wang J, Zaki MJ (2007) XProj: a framework for projected structural clustering of XML documents. In: KDD conference, San Jose, pp 46–55
Ahuja R, Orlin J, Magnanti T (1992) Network flows: theory, algorithms, and applications. Prentice-Hall, Englewood Cliffs
Chawathe SS (1999) Comparing hierachical data in external memory. In: Very large data bases conference. Morgan Kaufmann, San Francisco, pp 90–101
Chung F (1997) Spectral graph theory. Conference Board of the Mathematical Sciences, Washington, DC
Dalamagas T, Cheng T, Winkel K, Sellis T (2005) Clustering XML documents using structural summaries. In: Information systems. Elsevier, Jan 2005
Gibson D, Kumar R, Tomkins A (2005) Discovering large dense subgraphs in massive graphs. In: VLDB conference, pp 721–732. http://www.vldb2005.org/program/paper/thu/p721-gibson.pdf
Jain A, Dubes R (1998) Algorithms for clustering data. Prentice-Hall, Englewood
Kernighan BW, Lin S (1970) An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J 49:291–307
Lee M, Hsu W, Yang L, Yang X (2002) XClust: clustering XML schemas for effective integration. In: ACM conference on information and knowledge management. http://doi.acm.org/10.1145/584792.584841
Lian W, Cheung DW, Mamoulis N, Yiu S (2004) An efficient and scalable algorithm for clustering XML documents by structure. IEEE Trans Knowl Data Eng 16(1):82–96
Pei J, Jiang D, Zhang A (2005) On mining cross-graph quasi-cliques. In: ACM KDD conference, Chicago
Rattigan M, Maier M, Jensen D (2007) Graph clustering with network structure indices. In: Proceedings of the international conference on machine learning. ACM, New York, pp 783–790
Tsay AA, Lovejoy WS, Karger DR (1999) Random sampling in cut, flow, and network design problems. Math Oper Res 24(2):383–413
Zeng Z, Wang J, Zhou L, Karypis G (2007) Out-of-core coherent closed quasi-clique mining from large dense graph databases. ACM Trans Database Syst 32(2):13

Graph Kernels

Thomas Gärtner, Tamás Horváth, and Stefan Wrobel
Fraunhofer IAIS, Schloss Birlinghoven, University of Bonn, Sankt Augustin, Germany

Definition

The term *graph kernel* is used in two related but distinct contexts: On the one hand, graph kernels can be defined between graphs, that is, as a *kernel function* $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ where $\mathcal{G}$ denotes the set of all graphs un-der consideration. In the most common setting $\mathcal{G}$ is the set of all labeled undirected graphs. On the other hand, graph kernels can be defined between the vertices of a single graph, that is, as a kernel function $k : V \times V \to \mathbb{R}$ where V is the vertex set of the graph G under consideration. In the most common setting G is an undirected graph.

Motivation and Background

▸ Kernel methods are a class of machine learning algorithms that can be applied to any data set on which a valid, that is, positive definite, kernel function has been defined. Many kernel methods are theoretically well founded in statistical learning theory and have shown good predictive performance on many real–world learning problems.

Approaches for Kernels Between Graphs

One desireable property of kernels between graphs is that for non-isomorphic graphs $G, G' \in \mathcal{G}$ the functions $k(G, \cdot)$ and $k(G', \cdot)$ are not equivalent. If this property does not hold, the distance is only a pseudometric rather than a metric, that is, non-isomorphic graphs can be mapped to the same point in feature space and no kernel method can ever distinguish between the two graphs. However, it can be seen that computing graph kernels for which the property does hold is at least as hard as solving graph isomorphism (Gärtner et al. 2003).

For various classes of graphs, special purpose kernels have been defined such as for paths (▸ string kernels) and trees (Collins and Duffy 2002). These kernels are typically defined as the number of patterns that two objects have in common or as the inner product in a feature space counting the number of times a particular pattern occurs. The problem of computing a graph kernel where the patterns are all connected graphs, all cycles, or all paths and occurrence is determined by subgraph-isomorphism is, however, NP-hard (Gärtner et al. 2003).

Techniques that have been used to cope with the computational intractability of such graph kernels are (1) to restrict the considered patterns, for example, to bound the pattern size by a constant; (2) to restrict the class of graphs considered, for example, to trees or small graphs; (3) to define occurrence of a pattern differently, that is, not by subgraph-isomorphism; and (4) to approximate the graph kernel. Note that these four techniques can be combined.

While for technique (1) it is not immediately clear if the resulting graph kernel is feasible, technique (2) allows for fixed parameter tractable graph kernels. (Notice that even counting paths or cycles of length k in a graph is #W[1]-complete while the corresponding decision problem is fixed parameter tractable.) Though these will often still have prohibitive runtime requirements, it has been observed that enumerating cycles in real-world databases of small molecules is feasible (Horvath et al. 2004).

With respect to technique (3) it has been proposed to use graph kernels where the patterns are paths but the occurrences are determined by homomorphism (Gärtner et al. 2003; Kashima et al. 2003). Despite the explosion in the number of pattern occurrences (even very simple graphs can contain an infinite number of walks, that is, images of paths under homomorphism), if one downweights the influence of larger patterns appropriately, the kernel takes a finite value and closed form polynomial time computations exist. To increase the practical applicability of these graph kernels, it has been proposed to increase the number of labels by taking neighborhoods into account (Gärtner 2005) or to avoid "tottering" walks (Mahé et al. 2004).

Various approaches to approximate computation of graph kernels (4) exist. On the one hand, work on computing graph kernels based on restricting the patterns to frequent subgraphs (Deshpande et al. 2002) can be seen as approximations to the intractable all-subgraphs kernel. Computing such graph kernels is still NP-hard and no approximation guarantees are known. On the other hand, a recent graph kernel (Borgwardt et al. 2007) based on sampling small subgraphs of a graph at random is known to have a polynomial time algorithm with approximation guarantees.

The most common application scenario for such graph kernels is the prediction pharmaceutical activity of small molecules.

Approaches for Kernels on a Graph

Learning on the vertices of a graph is inherently *transductive*. Work on kernels between the vertices of a graph began with the "diffusion kernel" (Kondor and Lafferty 2002) and was later generalized in Smola and Kondor (2003) to a framework that contains the diffusion kernel as a special case. Intuitively, these kernels can be understood as comparing the neighborhoods of two vertices in the sense that the more neighbors two vertices have in common, the more similar they are. For classification, this definition is related to making the "cluster assumption", that is, assuming that the decision boundary between classes does not cross "high density" regions of the input space. To compute such graph kernels for increasing sizes of the neighborhood, one needs to compute the limit of a matrix poser series of the (normalized) graph Laplacian or its adjacency matrix. Different graph kernels arise from choosing different coefficients. In general, the limit of such matrix power series can be computed on the eigenvalues. For geometrically decaying parameters, the kernel matrix can also be computed by inverting a sparse matrix obtained by adding a small value to the diagonal of the Laplacian (in which case the kernel is called the "regularized Laplacian kernel") or the adjacency matrix.

In the case of the regularized Laplacian kernel, rather than first computing the kernel matrix and then applying an off-the-shelf implementation of a kernel method, it is often more effective to reformulate the optimization problem of the kernel method. Several possibilities for such reformulation have been proposed, including changing the variables as in Gärtner et al. (2006).

The most common application scenario for such graph kernels is the classification of entities in a social network.

Recommended Reading

Borgwardt KM, Petri T, Vishwanathan SVN, Kriegel H-P (2007) An efficient sampling scheme for comparison of large graphs. In: Mining and learning with graphs (MLG 2007), Firenze

Collins M, Duffy N (2002) Convolution kernel for natural language. In: Advances in neural information processing systems (NIPS), Vancouver, vol 16, pp 625–632

Deshpande M, Kuramochi M, Karypis G (2002) Automated approaches for classifying structures. In: Proceedings of the 2nd ACM SIGKDD workshop on data mining in bioinformatics (BIO KDD 2002), Edmonton

Gärtner T (2005) Predictive graph mining with kernel methods. In: Bandyopadhyay S, Maulik U, Holder LB, Cook DJ (eds) Advanced methods for knowledge discovery from complex data. Springer, Heidelberg, pp 95–121

Gärtner T, Flach PA, Wrobel S (2003) On graph kernels: hardness results and efficient alternatives. In: Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop (COLT 2003), vol. 2777 of LNCS. Springer, Heidelberg, pp 129–143

Gärtner T, Le QV, Burton S, Smola AJ, Vishwanathan SVN (2006) Large-scale multiclass transduction. In: Advances in neural information processing systems, vol 18. MIT, Cambride, pp 411–418

Horvath T, Gärtner T, Wrobel S (2004) Cyclic pattern kernels for predictive graph mining. In: Proceedings of the international conference on knowledge discovery and data mining (KDD 2004). ACM, New York, pp 158–167

Kashima H, Tsuda K, Inokuchi A (2003) Marginalized kernels between labeled graphs. In: Proceedings of the 20th international conference on machine learning (ICML 2003). AAAI Press, Menlo Park, pp 321–328

Kondor RI, Lafferty J (2002) Diffusion kernels on graphs and other discrete input spaces. In: Sammut C, Hoffmann A (eds) Proceedings of the nineteenth international conference on machine learning (ICML 2002), pp. 315–322, Morgan Kaufmann, San Fransisco

Mahé P, Ueda N, Akutsu T, Perret J-L, Vert J-P (2004) Extensions of marginalized graph kernels. In: Proceedings of the 21st international conference on machine learning (ICML 2004). ACM, New York, p 70

Smola AJ, Kondor R (2003) Kernels and regularization on graphs. In: Proceedings of the 16th annual conference on computational learning theory and the 7th kernel workshop (COLT 2003). Volume 2777 of LNCS. Springer, Heidelberg, pp 144–158

Graph Mining

Deepayan Chakrabarti
Yahoo! Research, Sunnyvale, CA, USA

Definition

Graph Mining is the set of tools and techniques used to (a) analyze the properties of real-world graphs, (b) predict how the structure and properties of a given graph might affect some application, and (c) develop models that can generate realistic graphs that match the patterns found in real-world graphs of interest.

Motivation and Background

A graph $G = (V, E)$ consists of a set of edges, E connec-ting pairs of nodes from the set V; extensions allow for weights and labels on both nodes and edges. Graphs edges can be used to point *from* one node *to* another, in which case the graph is called directed; in an *undirected* graph, edges must point both ways: $i \to j \Leftrightarrow j \to i$. A variant is the bipartite graph $G = (V_1, V_2, E)$ where only edges linking nodes in V_1 to nodes in V_2 are allowed.

A graph provides a representation of the binary relationships between individual entities, and thus is an extremely common data structure. Examples include the graph of hyperlinks linking HTML documents on the Web, the social network graph of friendships between people, the bipartite graphs connecting users to the movies they like, and so on. As such, mining the graph can yield useful patterns (e.g., the communities in a social network) or help in applications (e.g., recommend new movies to a user based on movies liked by other "similar" users). Graph mining can also yield patterns that are common in many real-world graphs, which can then be used to design graph "generators" (e.g., a generator that simulates the Internet topology, for use in testing next-generation Internet protocols).

Structure of Learning System

We split up this discussion into three parts: the analysis of real-world graphs, realistic graph generators, and applications on graphs. Detailed surveys can be found in Newman (2003) and Chakrabarti and Faloutsos (2006).

Analysis of Real-World Graphs

Four basic types of large-scale patterns have been detected in real-world graphs. The first is the existence of power-laws, for instance in the degree distribution and eigenvalue distribution. Most nodes have very low degree while a few have huge degree. This has implications for algorithms whose running times are bounded by the highest degree. The second set of patterns is called the "small-world phenomenon," which state that the diameter (or effective diameter) of such graphs are very small with respect to their size. Recall that the diameter of a connected graph is the maximum number of hops needed to travel between any pair of nodes; the effective diameter is a more robust version that specifies the number of hops within which a large fraction (say, 90 %) of all pairs can reach each other. Examples include a diameter of around 4 for the Internet Autonomous System graph, around 19 for the entire US power grid, around 4 for the graph of actors who worked together in movies, and so on. Third, many large graphs exhibit "community effects," where each community consists of a set of nodes that are more tightly connected to other nodes in the community compared to nodes outside. One local manifestation of this effect is the relatively high *clustering coefficient* which counts, given all pairs of edges (i, j) and (j, k), the probability of the existence of the "transitive" edge (i, k). High clustering coefficients imply tight connections in neighborhoods, which is the basis of strong community structure. Finally, many large graphs were shown to increase in density as they evolve over time, that is, the number of edges grows according to a power-law on the number of nodes. In addition, even while more nodes and edges are being added, the diameter of the graph tends to decrease.

Graph Generators

Imagine designing an application that works on the Internet graph. Collecting the entire Internet graph in one place is hard, making the testing process for such an application infeasible. In such cases, a realistic graph generator can be used to simulate a large "Internet-like" graph, which can be used in place of the real graph. This synthetic graph must match the patterns typically found in the Internet, including the patterns discussed in the previous paragraph. Apart from generating such graphs, the generators can provide insights into the *process* by which large graphs came to attain their structure.

One example of this is the *preferential attachment* model. Starting with a small initial graph, this model adds one new node every step. The new node is connected to m previous nodes, with the probability of connecting to node i being proportional to its degree. This idea, popularly known as "the rich get richer," can be shown to lead to a power-law degree distribution after a large number of nodes and edges have been added.

Many other models have also been proposed, which demonstrate graph generation as a random process, an optimization process, as a process on nodes embedded in some geographic space, and so on.

Applications

Some graph mining algorithms are meant to solve some application on any graph(s) provided as input to the algorithm. Several basic tools are commonly used in such applications, such as the ▶ Greedy Search Approach to Graph Mining the ▶ Inductive Database Search Approach to Graph Mining spectral methods, graph partitioning methods, and models based on random walks on graphs. *Tree Mining* is a special case of graph mining where the graphs are constrained to be trees. We will discuss a few such applications here.

Frequent subgraph mining: The aim is to find subgraphs that occur very frequently in the particular graph(s) in question (Kuramochi and Karypis 2001). This is quite useful in chemical datasets consisting of the graph structures of many different molecules (say, all protein molecules that have a certain chemical property); the frequent subgraphs in such molecules might represent basic structural units responsible for giving the molecules their special property. Unfortunately, the frequent subgraph problem subsumes the problem of subgraph isomorphism, and hence is NP-Hard. However, clever techniques have been devised to represent subgraphs so that checking for isomorphism can be done quickly in many cases.

Community detection: The problem is to detect tightly knit groups of nodes, where all nodes in the group have "similar" linkage structure. There are many algorithms, each optimizing for a different notion of similarity. Examples include graph partitioning methods such as spectral partitioning (Ng et al. 2002) and METIS that try to minimize the number of edges linking nodes across partitions, and co-clustering methods that aim for homogeneity in links across partitions.

Information diffusion and virus propagation: The spread of a contagious disease or a computer virus can be modeled (somewhat crudely) as a contact process on a graph, where the nodes are individuals who can get infected, and the links allow transmission of the contagion from an infected individual to an uninfected one. Similar models have been proposed to model the diffusion of information in social networks. The topology of the graph can be used to infer the most "influential" nodes in the graph, who are most capable of spreading the information quickly throughout the graph (Kempe et al. 2003).

Graph kernels: While subgraph isomorphism is a hard problem, we still need to be able to compare graphs on the basis of some similarity measure that can be computed in polynomial time. In the Kernel-Based Approach to Graph Mining graph kernels perform this task by computing similarities based on numbers of walks, paths, cyclic patterns, trees, etc.

Ranking on graphs: Given a graph (say, the Web hyperlink graph), we often need a ranking of the nodes in the graph. The ranking could be static (as in Page-Rank Brin and Page 1998) or it could depend on a user-specified query node. Such algorithms typically use some version of random walks on graphs (Lovász 1993), with the probability of the walk hitting a node being correlated with the importance of the node; such importances in turn yield a ranking of the nodes. Both static and query-dependent rankings can be useful in information retrieval settings, where a user desires information pertinent (i.e., "similar") to her query.

Cross-References

- ▸ Graphs
- ▸ Graph Kernels
- ▸ Greedy Search Approach of Graph Mining
- ▸ Inductive Database Approach to Graphmining
- ▸ Link Mining and Link Discovery
- ▸ Tree Mining

Recommended Reading

Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Comput Netw ISDN Syst 30(1–7):107–117

Chakrabarti D, Faloutsos C (2006) Graph mining: laws, generators and algorithms. ACM Comput Surv 38(1):2

Kempe D, Kleinberg J, Tardos E (2003) Maximizing the spread of influence through a social network. In: KDD, Washington, DC

Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: ICDM, San Jose, pp 313–320

Lovász L (1993) Random walks on graphs: a survey. In: Combinatorics: Paul Erdös is eighty, vol 2, pp 353–397

Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45:167–256

Ng A, Jordan M, Weiss Y (2002) On spectral clustering: analysis and an algorithm. In: NIPS, Vancouver

Graphical Models

Julian McAuley[1,2], Tibério Caetano[2], and Wray L. Buntine[2,3]
[1]Computer Science Department, University of California, San Diego, CA, USA
[2]Statistical Machine Learning Program, NICTA, Canberra, ACT, Australia
[3]Faculty of Information Technology, Monash University, Clayton, VIC, Australia

Definition

Graphical models are a means of compactly representing multivariate distributions, allowing for efficient algorithms to be developed when dealing with high-dimensional data. At their core, graphical models make use of the fact that high-dimensional distributions tend to factorize around local interactions, meaning that they can be expressed as a product of low-dimensional terms.

The notation we shall use is defined in Table 1, and some core definitions are presented in Table 2.

A few examples of the types of data that can be efficiently represented using graphical models are shown in Fig. 1. Here we have high-dimensional distributions (e.g., the probability of observing the pixels of a particular image), which we model in terms of low-dimensional interactions. In each of the examples presented in Fig. 1, we are simply asserting that

$$\underbrace{p(x_A, x_B | x_C)}_{\text{function of three variables}} = \underbrace{p(x_A|x_C)\,p(x_B|x_C)}_{\text{functions of two variables}}, \quad (1)$$

which arises by a straightforward application of the product rule (Definition 1), along with the fact that X_A and X_B are *conditionally independent*, given X_C (Definition 3). The key observation we make is that while the left-hand side of (Eq. 1) is a function of three variables, its conditional independence properties allow it to be *factored* into functions of two variables (note that the name "graphical models" arises due to the fact that such interdependencies can be represented as a graph encoding the relationships between variables).

In general, we shall have a series of conditional independence statements about X:

$$\left\{ X_{A_i} \perp X_{B_i} \;\middle|\; X_{C_i} \right\}. \quad (2)$$

It is precisely these statements that define the "structure" of our multivariate distribution, which we shall express in the form of a graphical model.

Motivation and Background

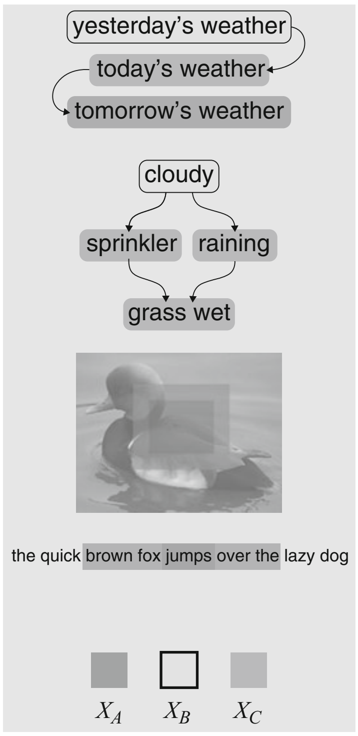
Graphical models are ubiquitous as a means to model multivariate data, since they allow us to represent high-dimensional distributions *compactly*; they do so by exploiting the *interdependencies* that typically exist in such

Graphical Models, Table 1 Notation

Notation	Description
$X = (X_1 \ldots X_N)$	A random variable (we shall also use $X = (A, B, C \ldots)$ in figures to improve readability)
$x = (x_1 \ldots x_N)$	A *realization* of the random variable X
$\mathcal{X}$	The sample space (domain) of X
X_A	X can be indexed by a *set*, where we assume $A \subseteq \{1 \ldots N\}$
$p(x)$	The probability that $X = x$
$\tilde{A}$	The *negation* of A, i.e., $\{1 \ldots N\} \setminus A$
$X_A \perp X_B$	X_A and X_B are *independent*
$X_A \perp X_B \mid X_C$	X_A and X_B are *conditionally independent*, given X_C

Graphical Models, Table 2 Definitions

Definition 1 (product rule) $p(x_A, x_B) = p(x_A \mid x_B)p(x_B)$
Definition 2 (marginalization) $p(x_A) = \sum_{x_{\tilde{A}} \in \mathcal{X}_{\tilde{A}}} p(x_A, x_{\tilde{A}})$
Definition 3 (conditional independence) X_A and X_B are said to be conditionally independent (given X_C) iff $p(x_a \mid x_b, x_c) = p(x_a \mid x_c)$, for all x_a, x_b, and x_c; the conventional definition of "independence" is obtained by setting $X_C = \varnothing$



Graphical Models, Fig. 1 Some examples of conditional independence; we say that X_A and X_B are *conditionally independent*, given X_C, or more compactly $X_A \perp X_B \mid X_C$

data. Put simply, we can take advantage of the fact that high-dimensional distributions can often be decomposed into *low-dimensional factors* to develop efficient algorithms by making use of the distributive law: $ab + ac = a(b + c)$.

Some motivating examples are presented in Fig. 1; similar examples are ubiquitous in fields ranging from computer vision and pattern recognition to economics and the social sciences. Although we are dealing with high-dimensional data, we can make certain statements about the *structure* of the variables involved, allowing us to express important properties about the distribution compactly. Some of the properties we would like to compute include the probabilities of particular outcomes and the outcomes with the highest probability.

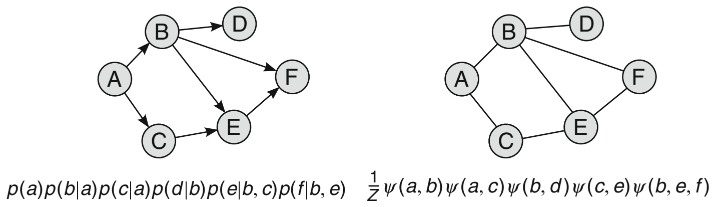
Theory

Directed Graphical Models

Due to the product rule (Definition 1), it is clear that *any* probability distribution can be written as

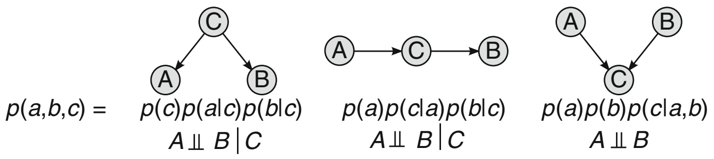
$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i} \mid x_{<\pi_i}) \qquad (3)$$

for an arbitrary permutation π of the labels, where we define $<i := \{1 \ldots i-1\}$. For example any four-dimensional distribution can be written



A
B
C
D
E
F
$p(a)p(b|a)p(c|a)p(d|b)p(e|b,c)p(f|b,e)$
$\frac{1}{Z}\psi(a,b)\psi(a,c)\psi(b,d)\psi(c,e)\psi(b,e,f)$

Graphical Models, Fig. 2 A directed model (*left*) and an undirected model (*right*). The joint distributions they represent are shown



Graphical Models, Fig. 3 Some simple Bayesian Networks and their implied independence statements. Note in particular that in the rightmost example, we *do not* have $A \perp B \mid C$

as

$$p(x_a, x_b, x_c, x_d) = p(x_c)p(x_b|x_c)p(x_d|x_c, x_b) \\ p(x_a|x_c, x_b, x_d). \qquad (4)$$

With this idea in mind, consider a model $p(x)$ for which we have the conditional independence statements:

$$\left\{p(x_{\pi_i}|x_{<\pi_i}) = p(x_{\pi_i}|x_{pa_{\pi_i}})\right\}, \qquad (5)$$

where $pa_{\pi_i} \subset < \pi_i$. We now have

$$p(x) = \prod_{i=1}^{N} p(x_{\pi_i}|x_{pa_{\pi_i}}). \qquad (6)$$

We can interpret pa_i as referring to the "parents" of the node i. Essentially, we are saying that a variable is conditionally independent on its nondescendants, given its parents.

We can represent (Eq. 6) using a directed acyclic graph (DAG) by representing each variable X_i as a node; an arrow is formed from X_j to X_i if $j \in pa_i$. An example of such a representation is given in Fig. 2. It can easily be shown that the resulting graph is always acyclic.

A *Bayesian Network* (a type of *directed* graphical model) is simply a set of probability distributions of the form $p(x) = \prod_{i=1}^{N} p(x_i|x_{pa_i})$. Every Bayesian Network can be represented as a DAG, though we often simply say that the Bayesian Network "is" the DAG. Some trivial examples and the type of independence statements they imply are shown in Fig. 3.

We finish this section with a simple lemma:

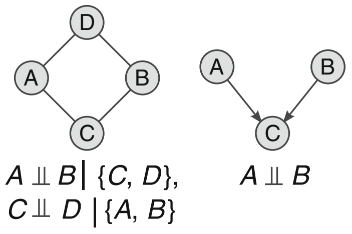
Lemma 1 (Topological Sort) *Every DAG has at least one permutation π that "sorts" the nodes such that each node has a larger index than its parents; in other words, the factorization associated to any DAG can be written in the form of (Eq. 6) for at least one π such that $\pi_i > j$ for all i, where $j \in pa_{\pi_i}$.*

Undirected Graphical Models

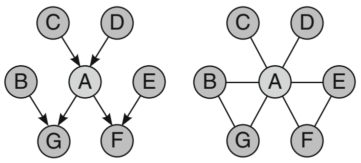
Although we have shown how conditional independence statements in the form of (Eq. 5) can be modeled using a DAG, there exist certain conditional independence statements that are not satisfied by *any* Bayesian Network, such as those in Fig. 4.

Markov random fields (or MRFs) allow for the specification of a different class of conditional independence statements, which are naturally represented by *undirected graphs* (UGs for short). The results associated with MRFs require a few additional definitions:

Definition 4 (Clique) A set of nodes X in a graph $\mathcal{G} = (V, E)$ is said to form a clique if $(X_i, X_j) \in E$ for every $X_i, X_j \in X$ (i.e., the subgraph X is fully connected).


D
A
B
C
A
B
C
$A \perp\!\!\!\perp B \mid \{C, D\}$,
$C \perp\!\!\!\perp D \mid \{A, B\}$
$A \perp\!\!\!\perp B$

Graphical Models, Fig. 4 There is no Bayesian Network that captures precisely the conditional independence properties of the Markov random field at *left*; there is no Markov random field that captures precisely the conditional independence properties of the Bayesian Network at *right*



Graphical Models, Fig. 5 The Markov blanket of the node A consists of its parents, its children, and the parents of its children (*left*). The corresponding structure for *undirected* models simply consists of the neighbors of A. Note that if we convert the directed model to an undirected one (using the procedure described in section "Conversion from Directed to Undirected Graphical Models"), then the Markov blankets of the two graphs are identical

Definition 5 (Maximal Clique) A clique X is said to be maximal if there is no clique Y such that $X \subset Y$.

A Markov random field is a probability distribution of the form $p(x) = \frac{1}{Z}\prod_{c \in C}\psi_c(x_c)$, where $\mathcal{C}$ is the set of maximal cliques of $\mathcal{G}$, ψ_c is an arbitrary nonnegative real-valued function, and Z is simply a normalization constant ensuring that $\sum_x p(x) = 1$.

Conversion from Directed to Undirected Graphical Models

It is possible to convert a directed graphical model to an undirected graphical model via the following simple procedure:

- For every node X_i with parents pa_{X_i}, add undirected edges between every $X_j, X_k \in pa_{X_i}$.
- Replace all directed edges with undirected edges.

In other words, we are replacing statements of the form $p(x_A|x_B)$ with $\psi(x_A, x_B)$, so that the nodes $\{X_i\} \cup pa_{X_i}$ now form a clique in the undirected model. This procedure of "marrying the parents" is referred to as *moralization*. Naturally, the undirected model formed by this procedure does not precisely capture the conditional independence relationships in the directed version. For example, if it is applied to the graph in Fig. 4 (*right*), then the nodes A, B, and C form a clique in the resulting model, which does not capture the fact that $A \perp B$. However, we note that every term of the form $p(x_i|x_{pa_i})$ appears in some clique of the undirected model, meaning that it can include all of the factors implied by the Bayesian Network.

Characterization of Directed and Undirected Graphical Models

We can now present some theorems that characterize both Bayesian Networks and Markov random fields:

Lemma 2 (Local Markov Property) *A node in a DAG is conditionally independent of its non-descendants, given its parents (this is referred to as the "directed" local Markov property); a node in a UG is conditionally independent of its non-neighbors, given its neighbors.*

Definition 6 (Markov Blanket) Given a node A, its "Markov blanket" is the minimal set of nodes C such that $A \perp B \mid C$ for all other nodes B in the model (in other words, the minimal set of nodes that we must know to "predict" the behavior of A).

Lemma 3 (Markov Blankets of Directed and Undirected Graphs) *In a directed network, the Markov blanket of a node A (denoted $MB(A)$) consists of its parents, its children, and its children's (other) parents. In an undirected network, it simply consists of the node's neighbors (see Fig. 5).*

Definition 7 (d-separation) The notion of a Markov blanket can be generalized to the notion of "d-separation." A set of nodes A is said to be d-separated from a set B by a set C if every (undirected) path between A and B is "blocked" when C is in the conditioning set (i.e., when C is observed). A path is said to be blocked if **either** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \leftarrow p_3$ (where arrows indicate edge directions) and neither p_2 nor any of its descendants are observed, **or** it contains (p_1, p_2, p_3) with $p_1 \rightarrow p_2 \rightarrow p_3$ and p_2 is observed, **or** it contains (p_1, p_2, p_3) with $p_1 \leftarrow p_2 \rightarrow p_3$ and p_2 is observed.

Applying (Definition 7) to the directed graphs in Fig. 1, we would say that the aqua regions (X_C) *d-separate* the red regions (X_A) from the white regions (X_B); *all conditional independence statements can simply be interpreted as d-separation in a DAG*.

The analogous notion of *graph separation* for Markov random fields is simpler than that of d-separation for Bayesian Networks. Given an undirected graph $\mathcal{G}$ and disjoint subsets of nodes A, B, C, if A is only reachable from B via C, this means that A is *separated* from B by C and these semantics encode the probabilistic fact that $A \perp B \mid C$. This is illustrated in Fig. 6.

In both the directed and undirected case, a Markov blanket of a node is simply the minimal set of nodes that d-separates/graph separates that node from all others.

A complete characterization of the class of probability distributions represented by Bayesian Networks can be obtained naturally once conditional independence statements are mapped to d-separation statements in a DAG. The following theorem settles this characterization.

Theorem 1 *Let p be a probability distribution that satisfies the conditional independence statements implied by d-separation in a DAG. Then p factors according to (Eq. 6). The converse also holds.*

For Markov random fields, an analogous characterization exists:

Theorem 2 (Hammersley-Clifford) *If a strictly positive probability distribution p satisfies the conditional independence statements implied by graph separation in an undirected graph $\mathcal{G}$, then*

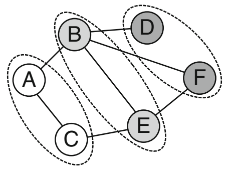
$$p(x) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(x_c). \tag{7}$$

The converse also holds, albeit in a more general sense in that p need not be strictly positive.

It can be shown that

directed local Markov property		*local Markov property*
$\Updownarrow$		$\Updownarrow$
d-separation in a DAG	and (for positive p) that	*graph separation in a UG*
$\Updownarrow$		$\Updownarrow$
factorization of p *by* (Eq. 6)		*factorization of* p *by* (Eq. 7)

Knowing that directed models can be converted to undirected models, we shall consider inference algorithms in undirected models only.



Graphical Models, Fig. 6 The nodes $\{B, E\}$ form a *clique*; the nodes $\{B, E, F\}$ form a *maximal clique*. The nodes $\{B, E\}$ *separate* the nodes $\{A, C\}$ from $\{D, F\}$

Applications

Inference Algorithms in Graphical Models

The key observation that we shall rely on in order to do inference efficiently is the *distributive law*:

$$\underbrace{ab + ac}_{\text{three operations}} = \underbrace{a(b + c)}_{\text{two operations}}. \tag{8}$$

By exploiting the factorization in a graphical model, we can use this law to perform certain queries efficiently (such as computing the marginal with respect to a certain variable).

As an example, suppose we wish to compute the marginal $p(x_1)$ in an MRF with the following factorization:

$$p(x) = \frac{1}{Z} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}). \tag{9}$$

Note that the graph representing this model is simply a *chain*. Computing the sum in the naïve way requires computing

$$p(x_1) = \frac{1}{Z} \sum_{x_{\{2 \ldots N\}}} \prod_{i=1}^{N-1} \psi(x_i, x_{i+1}), \tag{10}$$

whose complexity is $\Theta(\prod_{i=1}^{N} |\mathcal{X}_i|)$. However, due to the distributive law, the same result is simply

$$p(x_1) = \frac{1}{Z} \sum_{x_2} \Big[\psi(x_1, x_2) \sum_{x_3} \Big[\psi(x_2, x_3) \cdots \sum_{x_{N-1}} \Big[\psi(x_{N-2}, x_{N-1}) \sum_{x_N} \psi(x_{N-1}, x_N) \Big]\Big]\Big], \tag{11}$$

whose complexity is $\Theta(\sum_{i=1}^{N-1} |\mathcal{X}_i||\mathcal{X}_{i+1}|)$. As a more involved example, consider computing the marginal with respect to A in the undirected model in Fig. 2; here we wish to compute

$$p(a) = \frac{1}{Z} \sum_{b,c,d,e,f} \psi(a,b)\psi(a,c)\psi(b,d) \psi(c,e)\psi(b,e,f) \tag{12}$$

$$= \frac{1}{Z} \sum_{b} \psi(a,b) \sum_{c} \psi(a,c) \sum_{d} \psi(b,d) \sum_{e} \psi(c,e) \sum_{f} \psi(b,e,f). \tag{13}$$

Exploiting the distributive law in this way is often referred to as the *Elimination Algorithm*. It is useful for computing the marginal with respect to a single variable. However, should we wish to compute the marginal with respect to *each* variable, for example, it is not an efficient algorithm as several operations shall be repeated.

Belief Propagation

In tree-structured models, the elimination algorithm can be adapted to avoid repeated computations, using a message-passing scheme known as *belief propagation*, or the *sum-product* algorithm. This is presented in Algorithm 3. Here the "cliques" in the model are simply edges. This algorithm was invented independently by many authors and is the most efficient among many variations.

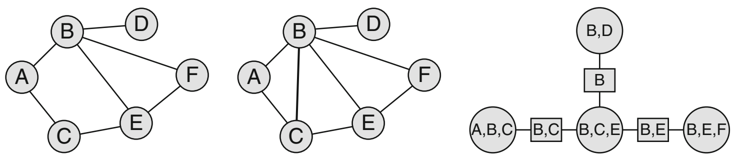
It can be easily demonstrated that the condition in Algorithm 3, Line 3, is always satisfied by some pair of edges until all messages have been passed: initially, it is satisfied by all of the "leaves" of the model; messages are then propagated inward until they reach the "root" of the tree; they are then propagated outward.

Maximum a Posteriori (MAP) Estimation

Algorithm 3 allows us to compute the *marginals* of the variables in a graphical model. There are other related properties that we may also wish to compute, such as finding which states have the

Algorithm 3 The *sum-product* algorithm

```
Input: an undirected, tree-structured graphical model $\mathcal{X}$ with cliques $\mathcal{C}$ {the cliques are simply edges in this case}
1: define $m_{A \to B}(x_{A \cap B})$ to be the "message" from an edge $A$ to an adjacent edge $B$ {for example if $A = (a,b)$ and $B = (b,c)$ then we have $m_{(a,b) \to (b,c)}(x_b)$}
2: while there exist adjacent edges $A, B \in \mathcal{C}$ for which $m_{A \to B}$ has not been computed do
3:     find some $A \in \mathcal{C}$ such that $m_{C \to A}$ has been computed for every neighbor $C \in \Gamma(A)$, except $B$ {$\Gamma(A)$ returns the edges neighboring $A$; initially the condition is satisfied by all leaf-edges}
4:     $m_{A \to B}(x_{A \cap B}) := \sum_{x_{A \setminus B}} \{\psi_A(x_A) \prod_{C \in \Gamma(A) \setminus B} m_{C \to A}(x_{A \cap C})\}$
5: end while
6: for $A \in \mathcal{C}'$ do
7:     $marginal_A(x_A) := \psi_A(x_A) \prod_{C \in \Gamma(A)} m_{C \to A}(x_{A \cap C})$
8: end for
```



Graphical Models, Fig. 7 The graph at *left* is not chordal, since the cycle (A, B, E, C) does not contain a chord; adding the edge (B, C) results in a chordal (or triangulated) graph (*center*). The graph at *right* is a junction tree for the graph at *center*; the cliques of the triangulated graph form the nodes (*circles*); their *intersection sets* are shown as squares. Note that this is not the only junction tree that we could form – the node $\{B, D\}$ could connect to any of the other three nodes

highest probability (the *maximum a posteriori*, or simply "MAP" states). To do so, we note that the operations $(+, \times)$ used in Algorithm 3 can be replaced by $(\max, \times)$. This variant is usually referred to as the *max-product* (as opposed to *sum-product*) algorithm. Indeed, different quantities can be computed by replacing $(+, \times)$ by any pair of operations that form a *semiring* (Aji and McEliece 2000).

The Junction Tree Algorithm

Algorithm 3 applies only for tree-structured graphs. We can generalize this algorithm to general graphs. We do so by working with a different type of tree-structured graph, whose *nodes* contain the *cliques* in our original graph. We begin with some definitions:

Definition 8 (Chordal Graph) A graph $\mathcal{G}$ is said to be chordal if every cycle $(c_1 \dots c_n)$ in $\mathcal{G}$ contains a chord (i.e., an edge (c_i, c_j) such that $j > (i + 1)$).

Definition 9 (Clique Graph, Clique Tree) A clique graph $\mathcal{H}$ of a graph $\mathcal{G}$ is a graph whose nodes consist of (maximal) cliques in $\mathcal{G}$ and whose edges correspond to intersecting cliques in $\mathcal{G}$. A clique tree is a clique graph without cycles.

Definition 10 (Junction Tree) A clique tree $\mathcal{H}$ of $\mathcal{G}$ is said to form a junction tree if for every pair of nodes A, B (i.e., maximal cliques in $\mathcal{G}$), the path between them $(P_1 \dots P_m)$ satisfies $(A \cap B) \subset P_i$ for all $i \in \{1 \dots m\}$.

The algorithms we shall define apply only if the graph in question is *chordal*, or "triangulated" (Definition 8); this can always be achieved by adding additional edges to the graph, as demonstrated in Fig. 7; adding additional edges means increasing the size of the maximal cliques in the graph.

Finding the "optimal" triangulation (i.e., the one that minimizes the size of the maximal cliques) is an NP-complete problem. In practice, triangulation algorithms vary from simple greedy heuristics (e.g., select a node that has as few neighbors as possible) to complex approximation algorithms working within a factor of the optimal solution (Amir 2001).

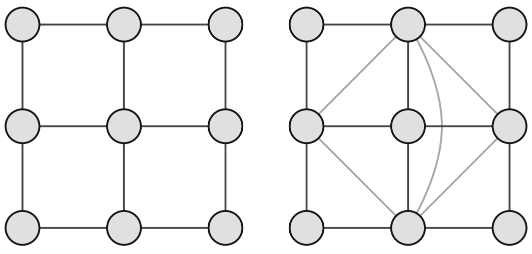
The problem of actually *generating* a junction tree from the triangulated graph is easily solved by a maximum spanning tree algorithm (where we prefer edges corresponding to pairs of cliques with large intersections).

Theorem 3 *Let $\mathcal{G}$ be a triangulated graph and $\mathcal{H}$ a corresponding clique tree. If the sum of the cardinalities of the intersection sets of $\mathcal{H}$ is maximum, then $\mathcal{H}$ is a junction tree. The converse also holds.*

If the nodes and edges in Algorithm 3 are replaced by the nodes (maximal cliques in $\mathcal{G}$) and edges (intersecting cliques in $\mathcal{G}$) of $\mathcal{H}$, then we recover the *junction tree algorithm*.

Approximate Inference

The act of triangulating the graph in the junction tree algorithm may have the effect of increasing the size of its maximal cliques, as in Fig. 8. This may be a problem, as its running time is exponential in the size of the maximal cliques in the *triangulated* graph (this size minus one is referred to as the *tree-width* of the graph, e.g., a chain has a tree-width of 1).



Graphical Models, Fig. 8 The graph above at *left* has maximal cliques of size two; in order to triangulate it, we must introduce maximal cliques of size four (*right*)

There are a variety of approximate algorithms that allow us to perform inference more efficiently:

Variational approximation. If doing inference in a graphical model $\mathcal{X}$ is intractable, we might search for a model $\mathcal{Y}$ for which inference is tractable and which is "similar" to $\mathcal{X}$ in terms of the KL-divergence between $p(x)$ and $p(y)$ (Wainwright and Jordan 2008).

Loopy belief propagation. We can build a clique graph from a graph that has not been triangulated, simply by connecting all cliques that intersect (in which case, the clique graph will contain loops). If we then propagate messages in some *random* order, we can obtain good approximations under certain conditions (Ihler et al. 2005).

Gibbs sampling. Given an estimate $x_{A \setminus B}$ of a set of variables $X_{A \setminus B}$, we can obtain an estimate of x_B by sampling from the conditional distribution $p(x_B | x_{A \setminus B})$. If we choose $B = \{X_i\}$, and repeat the procedure for random choices of $i \in \{1 \ldots N\}$, we obtain the procedure known as *Gibbs sampling* (Geman and Geman 1984).

There are several excellent books and tutorial papers on graphical models. A selection of tutorial papers includes Aji and McEliece (2000), Kschischang et al. (2001), Murphy (1998), and Wainwright and Jordan (2008); review articles include Roweis and Ghahramani (1997) and Smyth (1998), to name but a few.

Other signicant works include Koller and Friedman (2009), Jensen (2001) (introductory books), Edwards (2000) (undirected models), Pearl (1988, 2000) (directed models), Cowell et al. (2003) (exact inference), Jordan (1998) (learning and approximate inference), and Lauritzen (1996, Lauritzen and Spiegelhalter 1988) (a comprehensive mathematical theory).

There is also a variety of closely related models and extensions:

Gaussian graphical models. We have assumed throughout that our probability distributions are *discrete*; however, the only condition we require is that they are *closed under multiplication and marginalization*. This property is also satisfied for *Gaussian* random variables.

Hidden Markov models. In many applications, the variables in our model may be *hidden*. The above algorithms can be adapted to infer properties about our hidden states, given a sequence of observations.

Kalman filters. Kalman filters employ both of the above ideas, in that they include hidden state variables taking values from a *continuous* space using a Gaussian noise model. They are used to estimate the states of *linear dynamic systems* under noise.

Factor graphs. Factor graphs employ an alternate message-passing scheme, which may be preferable for computational reasons. Inference remains approximate in graphs with loops, though approximate solutions may be obtained more efficiently than by loopy belief propagation (Kschischang et al. 2001).

Relational models. Relational models allow us to explore the relationships between objects in order to predict the behavior and properties of each. Graphical models are used to predict the properties of an object based on others that relate to it (Getoor and Taskar 2007).

Learning. Often, we would like to *learn* either the parameters or the structure of the model from (possibly incomplete) data. There is an extensive variety of approaches; a collection of papers appears in Jordan (1998).

Deep learning. Deep belief networks can also be viewed as instances of graphical models, which impose a particular structure on the relationships between input variables, output variables, and hidden units. In particular, deep belief nets assume that complex relationships can be broken down into (a massive number of) purely pairwise interactions.

Cross-References

▶ Bayesian Network
▶ Deep Belief Nets
▶ Expectation Propagation
▶ Hidden Markov Models
▶ Markov Random Field

Recommended Reading

Aji SM, McEliece RJ (2000) The generalized distributive law. IEEE Trans Inf Theory 46(2):325–343
Amir E (2001) Efficient approximation for triangulation of minimum treewidth. In: Proceedings of the 17th conference on uncertainty in artificial intelligence. Morgan Kaufmann, San Francisco, pp 7–15
Cowell RG, Dawid PA, Lauritzen SL, Spiegelhalter DJ (2003) Probabilistic networks and expert systems. Springer, Berlin
Edwards D (2000) Introduction to graphical modelling. Springer, New York
Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6:721–741
Getoor L, Taskar B (eds) (2007) An introduction to statistical relational learning. MIT Press, Cambridge
Ihler AT, Fischer III JW, Willsky AS (2005) Loopy belief propagation: convergence and effects of message errors. J Mach Learn Res 6:905–936
Jensen FV (2001) Bayesian networks and decision graphs. Springer, Berlin
Jordan M (ed) (1998) Learning in graphical models. MIT Press, Cambridge
Koller D, Friedman N (2009) Probabilistic graphical models: principles and techniques. MIT Press, Cambridge
Kschischang FR, Frey BJ, Loeliger HA (2001) Factor graphs and the sum-product algorithm. IEEE Trans Inf Theory 47(2):498–519
Lauritzen SL (1996) Graphical models. Oxford University Press, Oxford
Lauritzen SL, Spiegelhalter DJ (1988) Local computations with probabilities on graphical structures and their application to expert systems. J R Stat Soc Ser B 50:157–224
Murphy K (1998) A brief introduction to graphical models and Bayesian networks. Morgan Kaufmann, San Francisco
Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Francisco
Pearl J (2000) Causality. Cambridge University Press, Cambridge
Roweis S, Ghahramani Z (1997) A unifying review of linear Gaussian models. Neural Comput 11:305–345
Smyth P (1998) Belief networks, hidden Markov models, and Markov random fields: a unifying view. Pattern Recogn Lett 18:1261–1268
Wainwright MJ, Jordan MI (2008) Graphical models, exponential families, and variational inference. Found Trends Mach Learn 1:1–305

Graphs

Tommy R. Jensen
Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria

Definition

Graph Theory is (dyadic) relations on collections specified objects. In its most common, a *graph* is a pair $G = (V, E)$ of a (finite) set of *vertices* V and a set of *edges* E (or *links*). Each edge e is a 2-element subset $\{u, v\}$ of V, usually abbreviated as $e = uv$; u and v are called the *endvertices* of e, they are mutually *adjacent* and each is *incident* to e in G. This explains the typical model of a *simple graph*.

A *directed graph* or ▶ digraph is a more general structure, in which the edges are replaced by ordered pairs of distinct elements of the vertex set V, each such pair being referred to as an *arc*. Another generalization of a graph is a *hypergraph* or "set-system" on V, in which the *hyperedges* may have any size. Various concepts in graph theory extend naturally to *multigraphs*, in which each pair of (possibly identical) vertices may be adjacent via several edges (respectively *loops*). Also studied are *infinite graphs*, for which the vertex and edge sets are not restricted to be finite.

A graph is conveniently depicted graphically by representing each vertex as a small circle, and representing each edge by a curve that joins its two endvertices. A digraph is similarly depicted by adding an arrow on the curve representing an *arc* showing the direction from its *tail* to its (possibly identical) *head*.

Motivation and Background

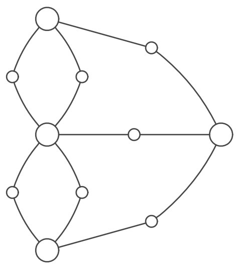
One of the very first results in graph theory appeared in Leonhard Euler's paper on Seven Bridges of Königsberg, published in 1736. The paper contained the complete solution to the problem whether, when given a graph, it is possible to locate an *Euler tour*, that is, a sequence of adjacent edges (each edge imagined to be traversed from one end to the other) that uses every edge exactly once. Figure 1 illustrates the four main parts of the city of Königsberg with the seven bridges connecting them; since this graph contains four vertices of odd degree, it does not allow an Euler tour.

Applications of graphs are numerous and widespread. Much of the success of graph theory is due to the ease at which ideas and proofs may be communicated pictorially in place of, or in conjunction with, the use of purely formal symbolism.

Theory

Isomorphism

A graph drawing should not be confused with the graph itself (the underlying abstract structure) as there are several ways to structure the graph drawing. It only matters which vertices are connected to which others by how many edges, the exact layout may be suited for the particular purpose at hand. It is often a problem of independent interest to optimize a drawing of a given graph in terms of aesthetic features.

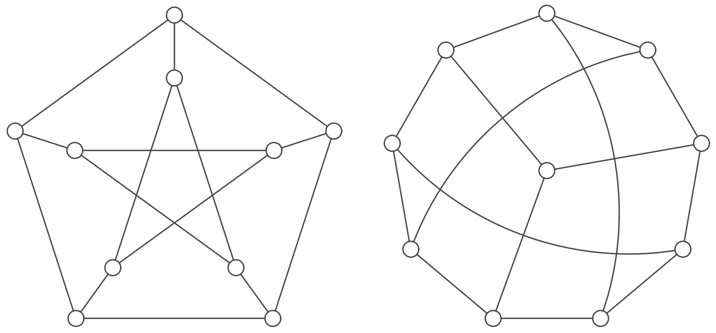

Graphs, Fig. 1 A graph of the city of Königsberg

In practice it is often difficult to decide if two drawings represent the same graph (as in Fig. 2). This decision problem has gained increasing status in complexity theory, with growing suspicion that this problem may fall in a new class of problems, which lies between the familar classes of polynomially solvable and NP-complete ▸ (NP-completeness) problems (supposing that these classes are indeed distinct; for issues related to the complexities of decision and optimization problems see Garey and Johnson 1979). Nonetheless it is customary in the treatment of abstract graphs to consider two graphs identical if they are isomorphic. A closely related problem, the *subgraph isomorphism problem*, an NP-complete problem, consists in finding a given graph as a subgraph of another given graph.

Whereas there seems common agreement in the graph theoretic community on what constitutes a drawing of a graph, it may be considered a weakness, and sometimes a source of confusion, that even the most central general sources on the fundamentals of graph theory, such as the monographs (Berge 1976; Bondy and Murty 2007; Diestel 2005), do not agree on a common formalization of the theory.

Classes of Graphs

Important special classes of graphs are *bipartite graphs*, for which the vertex set is partitionable into two classes A, B with every edge having one end in A and one in B; in particular the *complete bipartite graph* $K_{m,n}$ has $|A| = m, |B| = n$, and every vertex in A is joined to every vertex in B. The *complete graph* K_n consists of n vertices that are all pairwise adjacent. A *path* of length n consists of vertices $v_0, v_1, \ldots, v_n$ with edges $v_{i-1}v_i$ for $i = 1, 2, \ldots, n$; such a path *joins* its two endvertices v_0 and v_n. A *circuit* of length n consists of a path of length $n - 1$ together with an additional edge between the two endvertices of the path. A graph is *connected* if each pair of its vertices is joined by at least one path within the graph. Of central importance to the study of efficient search procedures in computer science is the class of *trees*, those connected graphs that


Graphs, Fig. 2 Two drawings of the same graph

contain no circuits. Most definitions have various natural counterparts for directed graphs, in particular a *tournament* is a directed graph in which each pair of vertices is joined by exactly one arc.

Properties of Graphs

Finding a complete subgraph of a given order in an input graph is called the *clique problem*. The complementary problem of finding an independent set is called the *independent set problem*. The *longest path problem* and the *longest circuit problem* have as special cases the *Hamilton path problem* and the *Hamilton circuit problem*, the latter two problems asking to find a path, respectively a circuit, that uses all vertices of the given graph. Each of these problems (or a suitable modification of it) belongs to the complexity class of NP-complete problems, hence is generally believed to be very difficult to solve efficiently. The weighted version of the Hamilton circuit problem, the so-called *travelling salesman problem* is of central importance in combinatorial optimization.

A graph is called *planar* if it may be drawn in the Euclidian plane without any two of its edges crossing except where they meet at a common endvertex. This is often a convenient way of representing a graph, whenever it is doable. A theorem of Kuratowski states that a graph is planar if and only if it contains homeomorphic copies of neither the complete bipartite graph $K_{3,3}$ (the three-houses-three-utilities-graph) nor the complete graph K_5. A main branch of graph theory is concerned with investigating relationships between the topological and combinatorial properties of graphs (Mohar and Thomassen 2001).

In 1852, Francis Guthrie posed the *four color problem*, asking if it is possible to color the countries of any map, using only four colors, in such a way that all pairs of bordering countries receive different colors. Equivalently, by representing dually every country as a vertex of a graph, with two vertices joined by an edge if their countries share a stretch of common border, the question is whether it is possible to color the vertices of a planar graph using four colors, so that any two adjacent vertices receive distinct colors. This problem, was solved a century later in 1976 by Kenneth Appel, Wolfgang Haken, and John Koch, who invested massive amounts of computing time to complete a graph theoretic approach developed by various mathematicians over a period of most of the preceding part of the twentieth century.

The problem of *coloring* a possibly nonplanar graph with a minimal number of colors, that is, to partition its vertex set into as few independent sets as possible, is a well-studied problem (e.g., see Jensen and Toft 1995), though NP-hard in general. In fact it is already an NP-complete problem to ask whether a given planar graph allows a coloring using at most three colors (see Garey et al. 1976). The recent strong perfect graph theorem provides one of quite few known examples of a fairly rich class of graphs, the *Berge graphs*, for which the coloring problem has a satisfactory solution (see Chudnovsky et al. 2006).

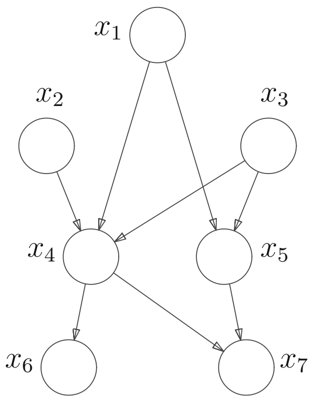
Other well-solved problems include finding a largest *matching* in a given graph; a largest set of edges no two of which share a common endvertex (see Lovász and Plummer (1986) for a thorough treatment of *matching theory*). The most interesting special case asks to find a *perfect matching*, having the property that every vertex is paired up with a unique vertex of the graph adjacent to it. For the special case of bipartite graphs (the *marriage problem*), the problem was solved by Dénes König in 1931. Even when given for every pair of vertices a measure of the desirability of pairing up these particular vertices (the *weighted matching problem*), there exists an efficient solution to the problem of finding an optimum matching of maximal total weight, discovered by Jack Edmonds in 1959.

Applications

As an example of a visualization application, Fig. 3 shows a digraph to symbolize for a collection of seven stochastic variables $x_1, \ldots, x_7$ that their joint distribution is given by the product

$$p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_3) \times p(x_6|x_4)p(x_7|x_4,x_5) \quad (1)$$

In addition to visualization of a network, a process, a search procedure, or any hierarchical structure, there are many applications using implementations of known graph algorithms on computers, so that the graph in question will only exist as an abstract datastructure within a program and thus remains invisible to the user.



Graphs, Fig. 3 Reproduced from Bishop (2006, p. 362)

There are different ways to store graphs in a computer. Often a combination of list and matrix structures will be preferred for storage and dynamic manipulation of a graph by an algorithm. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access but can consume a large amount of memory if a graph contains many vertices. In most cases it is convenient to represent a graph or digraph by an array containing, for each edge or arc, the pair of vertices that it joins, together with additional information, such as the weight of the edge, as appropriate. It may be an advantage in addition to store for each vertex a list of the vertices adjacent to it, or alternatively, a list of the edges incident to it, depending on the application.

The adjacency matrix of a graph, multigraph, or digraph on n vertices is an $n \times n$ matrix in which the *ij*-entry is the number of edges or arcs that join vertex i to vertex j (or more generally, the weight of a single such edge or arc). As a storage device this is inferior for sparse graphs, those with relatively few edges, but gains in importance when an application naturally deals with very dense graphs or multigraphs.

Future Directions

In recent years the theory of *graph minors* has been an important focus of graph theoretic research. A graph H is said to be a *minor* of a graph G if there exists a subgraph of G from which H can be obtained through a sequence of *edge contractions*, each consisting of the identification of the two ends of an edge e followed by the removal of e. A monumental effort by Neil Robertson and Paul Seymour has resulted in a proof of the Robertson–Seymour theorem (Robertson and Seymour 2004; see also Diestel 2005), with the important consequence that for any set $\mathcal{G}$ of graphs that is closed under taking minors, there exists a finite set of obstruction graphs, such that G is an element of $\mathcal{G}$ precisely if G does not contain any minor that belongs to the obstruction set. This theorem has several important algorithmic consequences, many still waiting to be fully explored.

A particularly challenging unsolved problem is the *Hadwiger conjecture* (see Jensen and Toft 1995), stating that any graph G that does not allow a vertex coloring with as few as k colors will have to contain the complete graph K_{k+1} as a minor. The special cases of $k \leq 5$ colors have been shown to be consequences of the four color theorem. But the problem remains open for all larger values of k.

Other central areas of research relate to the notoriously hard problems of vertex- and edge-coloring, and of Hamilton paths and circuits. These have important applications, but it is not expected that any satisfactory necessary and sufficient conditions will be found for their existence. Hence the study of sufficient conditions of practical value is lively pursued.

A list of open problems in graph theory can be found in Bondy and Murty (2007).

Recommended Reading

Bang-Jensen J, Gutin G (2000) Digraphs: theory, algorithms and applications. Springer monographs in mathematics. Springer, London. http://www.imada.sdu.dk/Research/Digraphs/
Berge C (1976) Graphs and hypergraphs. North-Holland mathematical library, vol 6. North-Holland Publishing Company, Amsterdam
Bishop CM (2006) Pattern recognition and machine learning. Springer, New York
Bondy JA, Murty USR (2007) Graph theory. Springer
Chudnovsky M, Robertson N, Seymour P, Thomas R (2006) The strong perfect graph theorem. Ann Math 164:51–229
Diestel R (2005) Graph theory, 3rd edn. Springer. http://www.math.uni - hamburg.de / home / diestel / books/graph.theory/GraphTheoryIII.pdf
Emden-Weinert T. Graphs: theory–algorithms–complexity. http://people.freenet.de/Emden-Weinert/graphs.html
Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York
Garey MR, Johnson DS, Stockmeyer LJ (1976) Some simplified NP-complete graph problems. Theor Comput Sci 1:237–267
Gimbel J, Kennedy JW, Quintas LV (eds) (1993) Quo vadis, graph theory? North-Holland, Amsterdam/New York
Harary F (1969) Graph theory. Addison-Wesley, Reading
Jensen TR, Toft B (1995) Graph coloring problems. Wiley, New York
Locke SC. Graph theory. http://www.math.fau.edu/locke/graphthe.htm
Lovász L, Plummer MD (1986) Matching theory. Annals of discrete mathematics, vol 29. North Holland, Amsterdam/New York
Mohar B, Thomassen C (2001) Graphs on surfaces. John Hopkins University Press, Baltimore
Robertson N, Seymour PD (2004) Graph minors. XX. Wagner's conjecture. J Comb Theory Ser B 92(2): 325–357
Weisstein EW. Books about graph theory. http://www.ericweisstein.com/encyclopedias/books/GraphTheory.html

Greedy Search

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

At each step in its search, a greedy algorithm makes the best decision it can at the time and continues without backtracking. For example, an algorithm may perform a ▸ general-to-specific search and at each step, commits itself to the specialization that best fits that training data, so far. It continues without backtracking to change any of its decisions. Greedy algorithms are used in many machine-learning algorithms, including decision tree learning (Breiman et al. 1984; Quinlan 1993) and ▸ rule learning algorithms, such as sequential covering.

Cross-References

▸ Rule Learning
▸ Learning as Search

Recommended Reading

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth International Group, Belmont
Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo

Greedy Search Approach of Graph Mining

Lawrence Holder
Washington State University, Pullman, WA, USA

Definition

▶ Greedy search is an efficient and effective strategy for searching an intractably large space when sufficiently informed heuristics are available to guide the search. The space of all subgraphs of a graph is such a space. Therefore, the greedy search approach of ▶ graph mining uses heuristics to focus the search toward subgraphs of interest while avoiding search in less interesting portions of the space. One such heuristic is based on the compression afforded by a subgraph; that is, how much is the graph compressed if each instance of the subgraph is replaced by a single vertex. Not only does compression focus the search, but it has also been found to prefer subgraphs of interest in a variety of domains.

Motivation and Background

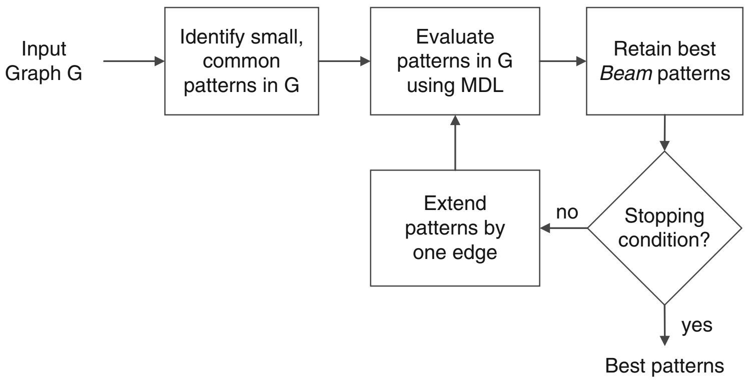
Many data mining and machine learning methods focus on the attributes of entities in the domain, but the relationships between these entities also represents a significant source of information, and ultimately, knowledge. Mining this relational information is an important challenge both in terms of representing the information and facing the additional computational obstacles of analyzing both entity attributes and relations. One efficient way to represent relational information is as a graph, where vertices in the graph represent entities in the domain, and edges in the graph represent attributes and relations among the entities. Thus, mining graphs is an important approach to extracting relational information. The main alternative to a graph-based representation is first-order logic, and the methods for mining this representation fall under the area of inductive logic programming. Here, the focus is on the graph representation.

Several methods have been developed for mining graphs (Washio and Motoda 2003), but most of these methods focus on finding the most frequent subgraphs in a set of graph transactions (e.g., FSG (Kuramochi and Karypis 2001), gSpan (Yan and Han 2002), Gaston (Nijssen and Kok 2004)) and use efficient exhaustive, rather than heuristic search. However, there are other properties besides frequency of a subgraph pattern that are relevant to many domains. One such property is the amount of compression afforded by the subgraph pattern, when each instance of the pattern is replaced by a single vertex. Searching for the most frequent subgraphs can be made efficient mainly through the exploitation of the downward closure property, which essentially says one can prune any extension of a subgraph that does not meet the minimum support frequency threshold. Unfortunately, the compression of a subgraph does not satisfy the downward closure property; namely, while a small extension of a subgraph may have less compression, a larger extension may have greater compression. Therefore, one cannot easily prune extensions and must search a larger portion of the space of subgraphs. Thus, one must resort to a greedy search method to search this space efficiently.

As with any greedy search approach, the resulting solution may sometimes be suboptimal, that is, the resulting subgraph pattern is not the pattern with maximum compression. The extent to which optimal solutions are missed depends on the type of greedy search and the strength of the heuristics used to guide the search. One approach is embodied in the graph-based induction (GBI) method (Matsuda et al. 2002; Yoshida et al. 1994). GBI continually compresses the input graph by identifying frequent triples of vertices, some of which may represent previously compressed portions of the input graph. Candidate triples are evaluated using a measure similar to information gain.

A similar approach recommended here is the use of a beam search strategy coupled with a compression heuristic based on the ▶ minimum description length (MDL) principle (Rissanen

Greedy Search Approach of Graph Mining, Fig. 1 Structure of the greedy search approach of graph mining


Input Graph G
Identify small, common patterns in G
Evaluate patterns in G using MDL
Retain best *Beam* patterns
Stopping condition?
no
Extend patterns by one edge
yes
Best patterns

1989). The goal is to perform unsupervised discovery of a subgraph pattern that maximizes compression, which is essentially a tradeoff between frequency and size. Once the capability to find such a pattern exists, it can be used in an iterative discovery-and-compress fashion to perform hierarchical conceptual clustering, and it can be used to perform supervised learning, that is, find patterns that compress the positive graphs, but not the negative graphs. This approach has been well studied (Cook and Holder 2000, 2007; Gonzalez et al. 2002; Holder and Cook 2003; Jonyer et al. 2001; Kukluk et al. 2007) and has proven successful in several domains (Cook et al. 2001; Eberle and Holder 2006; Holder et al. 2005; You et al. 2006).

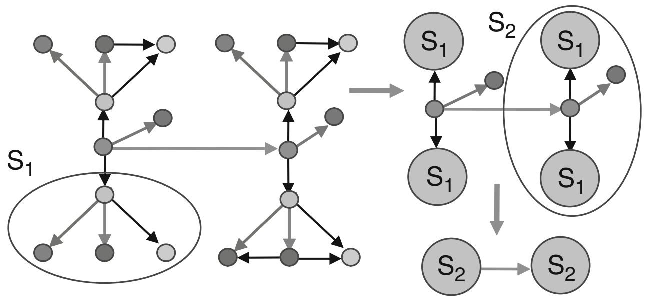
Structure of Learning System

Figure 1 depicts the structure of the greedy search approach of graph mining. The input data is a labeled, directed graph G. The search begins by identifying the set of small common patterns in G, that is, all vertices with unique labels having a frequency greater than one. The algorithm then iterates by evaluating the patterns according to the search heuristic, retaining the best patterns, and extending the best patterns by one edge until the stopping condition is met.

The search is guided by the minimum description length (MDL) principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best pattern is the one that minimizes the description length of the input graph when compressed by the pattern. The description length of the pattern S given the input graph G is calculated as $DL(G, S) = DL(S) + DL(G|S)$, where $DL(S)$ is the description length of the pattern, and $DL(G|S)$ is the description length of the input graph compressed by the pattern. The search seeks a pattern S that minimizes $DL(G,S)$.

While several greedy search strategies apply here (e.g., hill climbing, stochastic), the strategy that has been found to work best is the ▶ beam search. Of the patterns currently under consideration, the system retains only the best *Beam* patterns, where *Beam* is a user-defined parameter. These patterns are then extended by one edge in all possible ways according to the input graph, the extended patterns are evaluated, and then again, all but the best *Beam* patterns are discarded. This process continues until the stopping condition is met. Several stopping conditions are applicable here, including a user-defined limit on the number of patterns considered, the exhaustion of the search space, or the case in which all extensions of a pattern evaluate to a lesser value than their parent pattern. Once meeting the stopping condition, the system returns the best patterns. Note that while the naïve approach to implementing this algorithm would require an NP-complete subgraph isomorphism procedure to collect the instances of each pattern, a more efficient approach takes advantage of the fact that new patterns are always one-edge extensions of existing patterns, and, therefore, the instances of the extended patterns can be identified by search-

Greedy Search Approach of Graph Mining, Fig. 2 Example of the greedy search approach of graph mining



ing the extensions of the parent's instances. This process does require several isomorphism tests, which is the computational bottleneck of the approach, but avoids the subgraph isomorphism problem.

Once the search terminates, the input graph can be compressed using the best pattern. The compression procedure replaces all instances of the pattern in the input graph by single vertices, which represent the pattern's instances. Incoming and outgoing edges to and from the replaced instances will point to, or originate from the new vertex that represents the instance. The algorithm can then be invoked again on this compressed graph.

Figure 2 illustrates the process on a simple example. The system discovers pattern S_1, which is used to compress the data. A second iteration on the compressed graph discovers pattern S_2. Because instances of a pattern can appear in slightly different forms throughout the data, an inexact graph match, based on graph edit distance, can be used to address noise by identifying similar pattern instances.

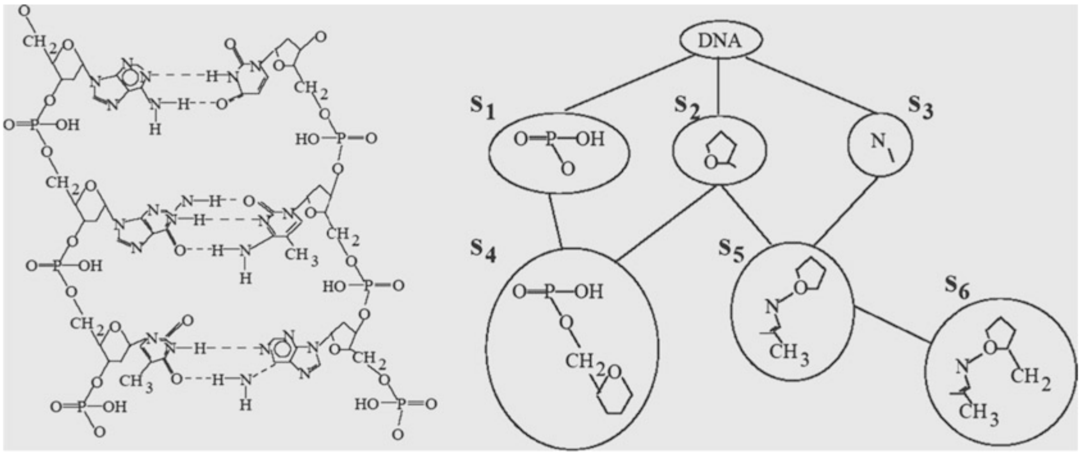
Graph-Based Hierarchical Conceptual Clustering

Given the ability to find a prevalent subgraph pattern in a larger graph and then compress the graph with this pattern, iterating over this process until the graph can no longer be compressed will produce a hierarchical, conceptual clustering of the input data (Jonyer et al. 2001). On the ith iteration, the best subgraph S_i is used to compress the input graph, introducing new vertices labeled S_i in the graph input to the next iteration. Therefore, any subsequently discovered subgraph S_j can be defined in terms of one or more of S_is, where $i < j$. The result is a *lattice*, where each cluster can be defined in terms of more than one parent subgraph. For example, Fig. 3 shows such a clustering done on a DNA molecule.

Graph-Based Supervised Learning

Extending a graph-based data mining approach to perform ▶ supervised learning involves the need to handle negative examples (focusing on the two-class scenario). In the case of a graph the negative information can come in three forms. First, the data may be in the form of numerous smaller graphs, or graph transactions, each labeled either positive or negative. Second, data may be composed of two large graphs: one positive and one negative. Third, the data may be one large graph in which the positive and negative labeling occurs throughout. The first scenario is closest to the standard supervised learning problem in that one has a set of clearly defined examples (Gonzalez et al. 2002). Let G^+ represent the set of positive graphs, and G^- represent the set of negative graphs. Then, one approach to supervised learning is to find a subgraph that appears often in the positive graphs, but not in the negative graphs. This amounts to replacing the information-theoretic measure with simply an error-based measure. This approach will lead the search toward a small subgraph that discriminates well. However, such a subgraph does not necessarily compress well, nor represent a characteristic description of the target concept.

One can bias the search toward a more characteristic description by using the information-theoretic measure to look for a subgraph that compresses the positive examples, but not the



Greedy Search Approach of Graph Mining, Fig. 3 Iterative application of the greedy search approach of graph mining yields the hierarchical, conceptual clustering on the right given an input graph representing the portion of DNA structure depicted on the left

negative examples. If $I(G)$ represents the description length (in bits) of the graph G, and $I(G|S)$ represents the description length of graph G compressed by subgraph S, then one can look for an S that minimizes $I(G^{+}|S) + I(S) + I(G^{-}) - I(G^{-}|S)$, where the last two terms represent the portion of the negative graph incorrectly compressed by the subgraph. This approach will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples.

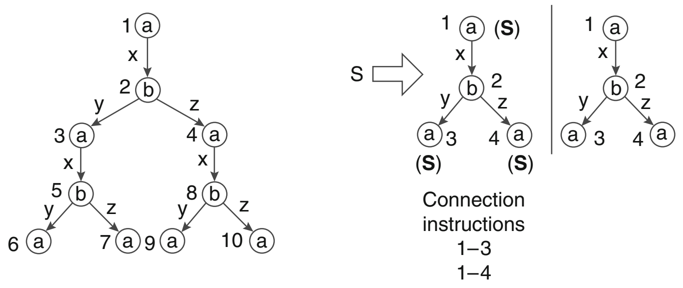
Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. If using the error measure, then any positive example containing the learned subgraph would be removed from subsequent iterations. If using the information-theoretic measure, then instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex. Note that the compression is a lossy one, that is, one does not keep enough information in the compressed graph to know how the instance was connected to the rest of the graph. This approach is consistent with the goal of learning general patterns, rather than mere compression.

Graph Grammar Inference

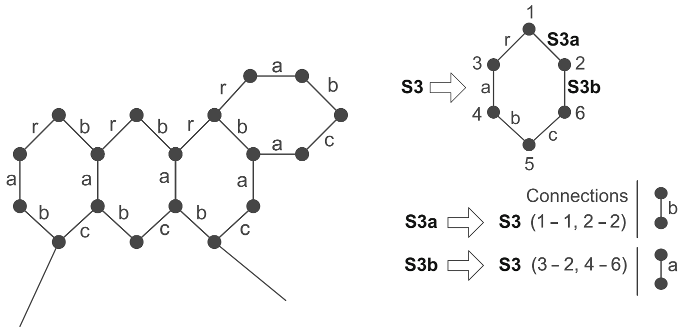
In the above algorithms the patterns are limited to non-recursive structures. In order to learn subgraph motifs, or patterns that can be used as the building blocks to generate arbitrarily large graphs, one needs the ability to learn graph grammars. The key to the inference of a graph grammar is the identification of overlapping structure. One can detect the possibility of a recursive graph-grammar production by checking if the instances of a pattern overlap. If a set of instances overlap by a single vertex, then one can propose a recursive node-replacement graph grammar production. Figure 4 shows an example of a node-replacement graph grammar (right) learned from a simple, repetitive input graph (left). The input graph in Fig. 4 is composed of three overlapping substructures. Based on how the instances overlap, one can also infer connection instructions that describe how the pattern can connect to itself. For example, the connection instructions in Fig. 4 indicate that the graph can grow by connecting vertex 1 of one pattern instance to either vertex 3 or vertex 4 of another pattern instance.

If a set of pattern instances overlap by an edge, then one can propose a recursive edge-replacement graph grammar production. Figure 5 shows an example of an edge-replacement graph grammar (right) learned from the input graph (left). Connection instructions describe how the motifs can connect via the edge labeled "a" or the edge labeled "b."

Apart from the inclusion of recursive patterns, the greedy search approach of graph mining is



Greedy Search Approach of Graph Mining, Fig. 4 The node-replacement graph grammar (*right*) inferred from the input graph (*left*). The connection instructions indicate how the pattern can connect to itself



Greedy Search Approach of Graph Mining, Fig. 5 The edge-replacement graph grammar (*right*) inferred from the input graph (*left*). The connection instructions indicate how the pattern can connect to itself

unchanged. Both recursive and non-recursive patterns are evaluated according to their ability to compress the input graph using the MDL heuristic. After several iterations of the approach, the result is a graph grammar consisting of recursive and non-recursive productions that both describe the input graph and provide a mechanism for generating graphs with similar properties.

Programs and Data

Most of the aforementioned functionality has been implemented in the SUBDUE graph-based pattern learning system. The SUBDUE source code and numerous sample graph data files are available at http://www.subdue.org.

Applications

Many relational domains, from chemical molecules to social networks, are naturally represented as a graph, and a graph mining approach is a natural choice for extracting knowledge from such data. Three such applications are described below.

A huge amount of biological data that has been generated by long-term research encourages one to move one's focus to a systems-level understanding of bio-systems. A biological network, containing various biomolecules and their relationships, is a fundamental way to describe bio-systems. Multi-relational data mining finds the relational patterns in both the entity attributes and relations in the data. A graph consisting of vertices and edges between these vertices is a natural data structure to represent biological networks. The greedy search approach of graph mining has been applied to find patterns in metabolic pathways (You et al. 2006). Graph-based supervised learning finds the unique substructures in a specific type of pathway, which help one understand better how pathways differ. Unsupervised learning shows hierarchical clusters that describe the common substructures in a specific type of pathway, which allow one to better understand the common features in pathways.

Social network analysis is the mapping and measuring of relationships and flows between people, organizations, computers, or other information processing entities. Such analysis is naturally done using a graphical representation of the domain. The greedy approach of graph mining has been applied to distinguish between criminal and legitimate groups based on their mode of communication (Holder et al. 2005). For example, terrorist groups tend to exhibit communications chains; whereas, legitimate groups (e.g., families) tend to exhibit more hub-and-spoke communications.

▶ Anomaly detection is an important problem for detecting fraud or unlawful intrusions. However, anomalies are typically rare and, therefore, present a challenge to most mining algorithms that rely on regularity and frequency to detect patterns. With the graph mining approach's ability to iteratively compress away regularity in the graph, what is left can be construed as anomalous. To distinguish this residual structure from noise, one can compare its regularity with the probability that such structure would appear randomly. The presence of rare structure that is unlikely to appear by chance suggests an anomaly of interest. Furthermore, most fraudulent activity attempts to disguise itself by mimicking legitimate activity. Therefore, another method for finding such anomalies in graphs is to first find the normative pattern using the greedy search approach of graph mining and then find unexpected deviations to this normative pattern. This approach has been applied to detect anomalies in cargo data (Eberle and Holder 2006).

Future Directions

One of the main challenges in approaches to graph mining is scalability. Since most relevant graph operations (e.g., graph and subgraph isomorphism) are computationally expensive, they can be applied to only modest-sized graphs that can fit in the main memory. Clearly, there will always be graphs larger than can fit in main memory, so efficient techniques for mining in such graphs are needed. One approach is to keep the graph in a database and translate graph mining operations into database queries. Another approach is to create abstraction hierarchies of large graphs so that mining can occur at higher-level, smaller graphs to identify interesting regions of the graph before descending down into more specific graphs. Traditional high-performance computing techniques of partitioning a problem into subproblems, solving the subproblems, and then recomposing a solution do not always work for graph mining problems, because partitioning the problem means breaking links which may later turn out to be important. New techniques and architectures are needed to improve the scalability of graph mining operations.

Another challenge for graph mining techniques is dynamic graphs. Most graphs represent data that can change over time. For example, a social network can change as people enter and leave the network, new links are established and old links are discarded. First, one would like to be able to mine for static patterns in the presence of the changing data, which will require incremental approaches to graph mining. Second, one would like to mine patterns that describe the evolution of the graph over time, which requires mining of time slice graphs or the stream of graph transaction events. Third, the dynamics can reside in the attributes of entities (e.g., changing concentrations of an enzyme in a metabolic pathway), in the relation structure between entities (e.g., new relationships in a social network), or both. Research is needed on efficient and effective techniques for mining dynamic graphs.

Cross-References

▶ Grammatical Inferences

Recommended Reading

Cook D, Holder L (2000) Graph-based data mining. IEEE Intell Syst 15(2):32–41

Cook D, Holder L (eds) (2007) Mining graph data. Wiley, New Jersey

Cook D, Holder L, Su S, Maglothin R, Jonyer I (2001) Structural mining of molecular biology data. IEEE Eng Med Biol Spec Issue Genomics Bioinform 20(4):67–74

Eberle W, Holder L (2006) Detecting anomalies in cargo shipments using graph properties. In: Proceedings of the IEEE intelligence and security informatics conference, San Diego, May 2006

Gonzalez J, Holder L, Cook D (2002) Graph-based relational concept learning. In: Proceedings of the nineteenth international conference on machine learning, Sydney, July 2002

Holder L, Cook D (2003) Graph-based relational learning: current and future directions. ACM SIGKDD Explor 5(1):90–93

Holder L, Cook D, Coble J, Mukherjee M (2005) Graph-based relational learning with application to security. Fundamenta Informaticae, Spec Issue Min Graphs Trees Seq 66(1–2):83–101

Jonyer I, Cook D, Holder L (2001) Graph-based hierarchical conceptual clustering. J Mach Learn Res 2:19–43

Kukluk J, Holder L, Cook D (2007) Inference of node replacement graph grammars. Intell Data Anal 11(4):377–400

Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the IEEE international conference on data mining (ICDM), San Jose, pp 313–320

Matsuda T, Motoda H, Yoshida T, Washio T (2002) Mining patterns from structured data by beam-wise graph-based induction. In: Proceedings of the fifth international conference on discovery science, Lubeck, pp 323–338

Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD), Seattle, (pp 647–652)

Rissanen J (1989) Stochastic complexity in statistical inquiry. World Scientific, New Jersey

Washio T, Motoda H (2003) State of the art of graph-based data mining. ACM SIGKDD Explor 5(1):59–68

Yan X, Han J (2002) gSpan: graph-based substructure pattern mining. In: Proceedings of the IEEE international conference on data mining (ICDM), Maebashi City, pp 721–724

Yoshida K, Motoda H, Indurkhya N (1994) Graph-based induction as a unified learning framework. J Appl Intell 4:297–328

You C, Holder L, Cook D (2006) Application of graph-based data mining to metabolic pathways. In: Workshop on data mining in bioinformatics, IEEE international conference on data mining, Hong Kong, Dec 2006

Group Detection

Hossam Sharara and Lise Getoor
University of Maryland, College Park, MD, USA

Synonyms

Community detection; Graph clustering; Modularity detection

Definition

Group detection can defined as the clustering of nodes in a graph into groups or communities. This may be a hard partitioning of the nodes, or may allow for overlapping group memberships. A community can be defined as a group of nodes that share dense connections among each other, while being less tightly connected to nodes in different communities in the network. The importance of communities lies in the fact that they can often be closely related to modular units in the system that have a common function, e.g., groups of individuals interacting with each other in a society (Girvan and Newman 2002), WWW pages related to similar topics (Flake et al. 2002), or proteins having the same biological function within the cell (Chen and Yuan 2006).

Motivation and Background

The work done in group detection goes back as early as the 1920s when Stuart Rice clustered data by hand to investigate political blocks (Rice 1927). Another early example is the work of George Homans (1950) who illustrated how simple rearrangement of the rows and columns of data matrices helped to reveal their underlying structure. Since then, group detection has attracted researchers from different areas such as sociology, mathematics, physics, marketing, statistics, and computer science.

Group detection techniques vary from simple similarity-based ▸ clustering algorithms that follow the classical assumption that the data points are independent and identically distributed, to more advanced techniques that take into consideration the existing relationships between nodes in addition to their attributes, and try to characterize the different distributions present in the data.

Theory Solution

A network is defined as a graph $G = (V, E)$ consisting of a set of nodes $v \in V$, and a set of edges $e \in E$. In the case of weighted networks,

$w(v_i, v_j)$ denotes the weight of the edge connection nodes v_i and v_j. A community, or a group, C is a subgraph $C(V', E')$ of the original graph $G(V, E)$ whose nodes and edges are subsets of the original graph's nodes and edges; i.e., $V' \subset V$ and $E' \subset E$.

Following the definition of the community, we can expect that all the vertices in any community must be connected by a path within the same community. This property is referred to in literature as *connectedness*, which implies that in the case of disconnected graphs, we can analyze each connected component separately, as communities cannot span different components.

Another important property that follows from the definition of a community is that the group of vertices within a community should share denser connections among each other, and fewer connections with the other vertices in the network. To quantify this measure, the link density of a group $\delta(C)$ is defined as the ratio between the number of internal edges in that group and the maximum number of possible internal edges:

$$\delta(C) = \frac{|E'|}{|V'| \times (|V'| - 1)/2} \tag{1}$$

Thus, for any community C, we require that $\delta(C) > \delta(G)$; where $\delta(G)$ is the average link density of the whole network. Similarly, the average link density between different communities, calculated using the ratio between the number of edges emanating from a group and terminating in another, and the maximum number possible of such edges, should generally be low.

Approaches

Beyond the intuitive discussion above, the precise definition of what constitutes a community involves multiple aspects. One important aspect is whether communities form hard partitions of the graph or nodes can belong to several communities. Overlapping communities do commonly occur in natural settings, especially in social networks. Currently, only a few methods are able to handle overlapping communities (Palla et al. 2005).

Other aspects should also be taken into consideration when defining community structure, such as whether link weights and/or directionalities are utilized, and whether the definition allows for hierarchical community structure, which means that communities may be parts of larger ones. However, one of the most important aspect that comes into consideration in community detection is whether the definition depends on global or local network properties. The main difference between the two approaches is whether the communities are defined in the scope of the whole network structure, such as methods based on centrality measures (Girvan and Newman 2002), global optimization methods (Newman and Girvan 2004), spectral methods (Arenas et al. 2006), or information-theoretic methods (Rosvall and Bergstrom 2008). Local methods, on the other hand, define communities based on purely local network structure, such as detecting cliques of different sizes, clique percolation method (Palla et al. 2005), and subgraph fitness method (Lancichinetti et al. 2009).

Local Techniques

Local methods for community detection basically rely on defining a set of properties that should exist in a community, then finding maximal subgraphs for which these set of properties hold. This formulation corresponds to finding maximal cliques in the network, where a clique is a subgraph in which all its vertices are directly connected.

However, there are some issues that rises from the previous formulation. First, finding cliques in a graph is an NP-Complete problem, thus most solutions will be approximate based on heuristic methods. Another more semantic issue is the interpretation of communities, especially in the context of social networks, where different individuals have different centralities within their corresponding groups, contradicting with the degree symmetry of the nodes in cliques. To overcome these drawbacks, the notion of a clique is relaxed to n-clique, which is a maximal subgraph where each pair of vertices are at most n-steps apart from each other.

Clustering Techniques

► Data clustering is considered one of the earliest techniques for revealing group structure, where data points are grouped based on the similarity between their corresponding features according to a given similarity measure. The main objective of traditional clustering methods is to obtain clusters or groups of data points possessing high intra-cluster similarity and low inter-cluster similarity. Classical data clustering techniques can be divided into partition-based methods such as k-means clustering (MacQueen 1967), spectral clustering algorithms (Alpert et al. 1999), and hierarchical clustering methods (Hartigan 1975), which are the most popular and the most commonly used in many fields.

One of the main advantages of the hierarchical clustering techniques is their ability to provide multiple resolutions at which the data can be grouped. In general, hierarchical clustering can be divided into agglomerative and divisive algorithms. The agglomerative algorithm is a greedy bottom-up one that starts with clusters including single data points then successively merge the pairs of clusters with the highest similarity. Divisive algorithms work in an opposite direction, where initially all the data points are regarded as one cluster, which is successively divided into smaller ones by splitting groups of nodes having the lowest similarity. In both algorithms, clusters are represented as a dendrogram, whose depths indicate the steps at which two clusters are joined. This representation clarifies which communities are built up from smaller modules, and how these smaller communities are organized, which can be particularly useful in the case of the presence of a normal hierarchy of community structure in the data. Hierarchical clustering techniques can easily be used in network domains, where data points are replaced by individual nodes in the network, and the similarity is based on edges between them.

Centrality-Based Techniques

One of the methods for community detection that is based on the global network structure is the one proposed by Girvan and Newman (2002), where they proposed an algorithm based on the betweenness centrality of edges to be able to recover the group structure within the network. Betweenness centrality is a measure of centrality of nodes in networks, defined for each node as the number of shortest paths between pairs of nodes in the network that run through it. The Girvan–Newman algorithm extended this definition for edges in the network as well, where the betweenness centrality of an edge is defined as the number of shortest paths between pairs of nodes that run along it.

The basic idea behind the algorithm is exploiting the fact that the number of edges connecting nodes from different communities is sparse. Following from that, all shortest paths between nodes from different communities should pass along one of these edges, increasing their edge betweenness centrality measure. Therefore, by following a greedy approach and removing edges with highest betweenness centrality from the network successively, the underlying community structure will be revealed. One of the major drawbacks of the algorithm is the time complexity, which is $O(|E|^2|V|)$ generally, and $O(|V|^3)$ for sparse networks. The fact that the edge betweenness needs only to be recalculated only for the edges affected by the edge removal can be factored in, which makes the algorithm efficient in sparse networks with strong community structure, but not very efficient on dense networks.

Modularity-Based Techniques

The concept of modularity was introduced by Newman and Girvan (2004) as a measure to evaluate the quality of a set of extracted communities in a network, and has become one of the most popular quality functions used for community detection. The basic idea is utilizing a *null model*: a network having the same set of nodes as the original one, but with random edges placed between them taking into account preserving the original node degrees. The basic idea is that the created random network is expected to contain no community structure, thus by comparing the number of edges within the extracted communities against the expected number of edges in the same communities from the random network, we can judge the quality of the extracted community structure. More specifically, the modularity Q is defined as follows

$$Q = \frac{1}{2|E|} \sum_{ij} \left[A_{ij} - \frac{\deg(i) \times \deg(j)}{2|E|} \right] \delta_k(c_i, c_j) \quad (2)$$

where A_{ij} is the element of the adjacency matrix of the network denoting the number of edges between nodes i and j, $\deg(i)$ and $\deg(j)$ are the degrees of nodes i and j respectively, c_i and c_j are the communities to which nodes i and j belong respectively, and δ_k refers to the kronecker delta. The summation runs over all pairs of nodes within the same community.

Clearly, a higher modularity value indicates that the average link density within the extracted community is larger than that of the random network where no community structure is present. Thus, modularity maximization can be used as the objective for producing high-quality community structure. However, modularity maximization is an NP-hard problem. Nevertheless, there have been several algorithms for finding fairly good approximations of the modularity maximum in reasonable amount of time.

One of the first modularity maximization algorithms was introduced by Newman in 2004. It is a greedy hierarchical agglomerative clustering algorithm, which starts with individual nodes and merges them in the order of increasing the overall modularity of the resulting configuration. The time complexity of this greedy algorithm is $O(|V|(|E| + |V|))$ or $O(|V|^2)$ for sparse networks, which enables the user to run community detection on large networks in a reasonable amount of time.

Issues

One of the main issues with the methods of group detection in network setting is the focus on the network structure, without taking into consideration other properties of nodes and edges in the network. This issue often results in a lack of correspondence between the extracted communities and the functional groups in the network (Shalizi et al. 2007). This also leads to another common problem which is how to validate the resulting communities produced by any of the proposed techniques.

Although in network settings there are often different types of interactions between entities of different natures, most group detection methods work on single-mode networks, which have just a single node and edge type. Fewer works focus on finding groups in more complex, multimodal settings, where nodes from different types have multiple types of interactions with each other. One of the most common approaches to deal with these types of networks is projecting them into a series of individual graphs for each node type. However, this approach results in losing some of the information that could have been retained by operating collectively on the original multi-relational network.

Another issue also gaining interest is developing methods for group detection in dynamic network settings (Tantipathananandh and Berger-Wolf 2009), where the underlying network structure changes over time. Most of the previous work on group detection focused on static networks, and handles the dynamic case by either analyzing a snapshot of the network at a single point in time, or aggregating all interactions over the whole time period. Both approaches do not capture the dynamics of change in the network structure, which can be an important factor in revealing the underlying communities.

Cross-References

▶ Graph Clustering
▶ Graph Mining

Recommended Reading

Alpert C, Kahng A, Yao S (1999) Spectral partitioning: the more eigenvectors, the better. Discret Appl Math 90:3–26

Arenas A, Daz-Guilera A, Prez-Vicente CJ (2006) Synchronization reveals topological scales in complex networks. Phys Rev Lett 96(11):114102
Chen J, Yuan B (2006) Detecting functional modules in the yeast protein–protein interaction network. Bioinformatics 22(18):2283–2290
Flake GW, Lawrence S, Giles CL, Coetzee F (2002) Self-organization and identification of web communities. IEEE Comput 35:66–71
Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99:7821–7826
Hartigan JA (1975) Clustering algorithms. Wiley, New York
Homans GC (1950) The human group. Harcourt, Brace, New York
Lancichinetti A, Fortunato S, Kertesz J (2009) Detecting the overlapping and hierarchical community structure in complex networks. N J Phys 11:033015
MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of fifth Berkeley symposium on mathematical statistics and probability, vol 1. University of California Press, Berkeley, pp 281–297
Newman MEJ (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6):066133
Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69:026113
Palla G, Dernyi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814–818
Rice SA (1927) The identification of blocs in small political bodies. Am Pol Sci Rev 21: 619–627
Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105: 1118–1123
Shalizi CR, Camperi MF, Klinkner KL (2007) Discovering functional communities in dynamical networks. In: Statistical network analysis: models, issues, and new directions. Springer, Berlin, pp 140–157
Tantipathananandh C, Berger-Wolf TY (2009) Algorithms for identifying dynamic communities. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris. ACM, New York

Grouping

▶ Categorical Data Clustering

Growing Set

Definition

A growing set is a subset of a ▶ training set containing data that are used by a learning system to develop models that are then evaluated against a ▶ pruning set.

Cross-References

▶ Data Set

Growth Function

▶ Shattering Coefficient