1

PROBABILITY

Unit Structure:

1.0 OBJECTIVES

After completing this chapter, you will be able to understand probability theory, Some common discrete distributions, Some common continuous distributions and Joint probability distributions as well as transformations of random variables, Monte Carlo approximation, Information theory and Directed graphical models and mixture models and EM algorithm like Latent variable models, Mixture models, Parameter estimation for mixture models, The EM algorithm.

1.1 INTRODUCTION

Probability can have several meanings in everyday conversation. Probability theory has been developed and applied in two major ways. Simple games such as coins, cards, dice, and roulette wheels are examples of games that interpret probabilities as relative frequencies. There is some regularity to the results of many trials in a game of chance, although the outcome of any given trial cannot be predicted with certainty.

As an example, if a coin is tossed with a probability of one-half, according to the relative frequency interpretation, that means the probability of receiving "heads" is about one-half in a large number of tosses, despite not implying anything about the outcome of any given toss.

We all know that "the probability that a coin will land heads is 0.5". How does that work? Probability can be interpreted in at least two different

ways. The frequentist interpretation is one. Probabilities, in this view, describe the long run frequency of events. For example, the preceding statement implies that if we flip the coin several times, we may anticipate it to fall heads around half of the time. 1 The Bayesian interpretation of probability is the alternative interpretation. Probability, according to this viewpoint, is used to assess our uncertainty about something; hence, it is primarily tied to information rather than repeated trials (Jaynes 2003). According to the Bayesian perspective, the preceding sentence indicates that we expect the coin will land heads or tails on the next toss.

Probability is the degree of certainty that an unknown event will occur.

Eg: The probability of raining today is 0.3.

Event space: All-possible-outcomes space.

Eg: E= {1,2,3,4,5,6} for a dice; E={H, T} for a coin.

Random variable: a variable whose values depend on the outcome of a random phenomenon.

Two types of random variables: Discrete and continuous.

1.2 A BRIEF REVIEW OF PROBABILITY THEORY

Mathematical study of random phenomena is known as probability theory. A random event's outcome can take any of a number of different forms; it cannot be predicted before it happens. The final result is thought to have been determined by chance.

1.2.1 Discrete random variables

The probability that the event A occurs is denoted by the term p(A). A may be the logical phrase "it will rain tomorrow," for example. We require that $0 \leq p(A) \leq 1$, where $p(A)=0$ indicates that the event will almost certainly not occur, and $p(A)=1$ indicates that the event will almost certainly occur. The probability of the event not A is denoted by p(A), which is defined as $p(A)=1 - p(A)$. We will frequently write $A = 1$ to indicate that event A is true and $A = 0$ to indicate that event A is false.

We can extend the concept of binary events by introducing a discrete random variable X that can take any value from a finite or countably infinite collection X. The probability that $X = x$ is denoted by $p(X = x)$, or just p(x) for short. In this context, p() is called as a probability mass function, or pmf. This meets the properties $0 \leq p(x) \leq 1$ and !

1.2.2 Fundamental rules

1. Probability of a union of two events.

Given two events, A and B, we define the probability of A or B as follows:

$p(A \vee B) = p(A) + p(B) - p(A \wedge B)$

$= p(A) + p(B)$

If A and B are mutually exclusive

2. Joint probabilities

We define the probability of the joint event A and B as follows:

$p(A, B) = p(A \wedge B) = p(A|B)p(B)$

This is known as the product rule. The marginal distribution is defined as follows given a joint distribution on two occurrences p(A, B):

$$p(A) = \sum_b p(A, B) = \sum_A p(A|B = b)p(B = b)$$

where we are summing across all possible states of B. Similarly, we can define p(B). This is also known as the sum rule or the total probability rule.

3. Conditional probability

The conditional probability of event A given that event B is true is defined as follows:

$$p(A|B) = \frac{p(A, B)}{p(B)} \text{ if } p(B) > 0$$

1.2.3 Bayes rule

The Bayes rule, also known as the Bayes Theorem, is obtained by combining the definition of conditional probability with the product and sum rules.

$$p(X = x|Y = y) = \frac{p(X = x, Y = y)}{p(Y = y)} = \frac{p(X = x)p(Y = y|X = x)}{\sum_{x'} p(X = x')p(Y = y|X = x')}$$

Example.

Assume 15 men out of 300 and 25 women out of 1000 are good orators. A random orator is chosen. Determine the probability that a man will be chosen. Assuming there is an equal number of men and women.

Solution:

Let there be 1000 men and 1000 women.

Let E_1 and E_2 be the events of choosing a man and a woman respectively. Then,

$P(E_1) = 1000/2000 = 1/2$, and $P(E_2) = 1000/2000 = 1/2$

Let E be the event of choosing an orator. Then,

$P(E|E_1) = 50/1000 = 1/20$, and $P(E|E_2) = 25/1000 = 1/40$

Probability of selecting a male person, given that the person selected is a good orator

$P(E_1/E) = P(E|E_1) * P(E_1)/ P(E|E_1) * P(E_1) + P(E|E_2) * P(E_2)$

$= (1/2 * 1/20) /\{(1/2 * 1/20) + (1/2 * 1/40)\}$

$= 2/3$

Hence the required probability is 2/3.

1.2.4 Independence and conditional independence

Let us first define conditional independence:

If there are two conditionally independent events A and B given a third event Y, then the occurrence/non-occurrence of A provides no information about the occurrence and non-occurrence of B (given Y), i.e., A and B are conditionally independent iff knowledge of A's occurrence provides no information on the likelihood of B occurring and vice versa. In terms of probabilities:

1. $P(A \cap B|Y) = P(A|Y).P(B|Y)$

2. $P(A|B \cap Y) = P(A|Y)$

Be clear that conditional independence does not imply independence, and vice versa. Let's explore some of the basic definitions.

1. Independence:

The occurrence of one event for two occurrences A and B has no affect whatever on the other event's occurrence.

Example: $P(A \cap B)=P(A).P(B)$

$P(A|B)=P(A)$

2. Conditional Independence:

Sometimes it's impossible to determine whether an event is independent of another because a third occurrence causes them to become so.

Given that C means, A is conditionally independent of B

$P(A|B,C) = P(A|C)$

i.e., When C is observed, B has no impact on the value of A.

We say X and Y are unconditionally independent or marginally independent, denoted $X \perp Y$, if we can represent the joint as the product of the two marginal,

i.e., $X \perp Y \Leftrightarrow p(X, Y) = p(X)p(Y)$

Theorem:

X ⊥ Y |Z if there exist function g and h such that

p(x, y|z) = g(x, z)h(y, z)

for all x, y, z such that p(z) > 0

1.2.5 Continuous random variables

The two forms of random variables are continuous random variables and discrete random variables. A random variable is one whose value varies on every possibility that could occur during an experiment. A discrete random variable is defined at a precise value, whereas a continuous random variable is defined throughout a range of values.

For instance, how long it takes to finish an exam for a 60-minute test. Possible values = all real numbers on the interval [0,60]

A random variable with an infinite number of possible values is referred to as a continuous random variable. As a result, there is no chance that a continuous random variable will have an exact value. A continuous random variable's features are described using the probability density function and the cumulative distribution function.

The probabilities connected to a continuous random variable are expressed using the probability density function (pdf) and the cumulative distribution function (CDF). Here are the formulas for continuous random variables for these functions.

pdf (probability density function):

We often use (x) to denote the PDF of x

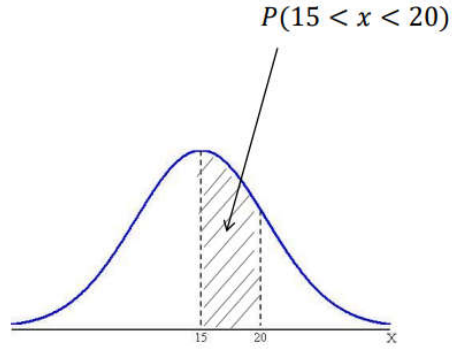
$(x) \geq 0$

whaer$f(x)$ can be larger than 1

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

$$\int_{x_1}^{x_2} f(x)dx = P(x_1 < x < x_2)$$

Example



PDF of Continuous Random Variable

A function that estimates the probability that a continuous random variable's value will fall within a given range of values is known as the probability density function. Given that X is assumed to be a continuous random variable, the pdf's formula, f(x), is as follows:

$$f(x) = \frac{dF(x)}{dx} = F'(x)$$

F(x) is the cumulative distribution function in this case.

The continuous random variable's pdf must meet the requirements listed below in order to be valid:

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

This specifies that the entire area under the PDF's graph must be equal to 1.

f(x) > 0. This implies that a continuous random variable's probability density function cannot be negative.

CDF of Continuous Random Variable

The probability density function can be integrated to obtain the cumulative distribution function of a continuous random variable. It can be characterised as the probability that the random variable, X, will have a value less than or equal to a specific value, x. The following is the formula for the cdf of a continuous random variable, evaluated between two points a and b**:**

$$P(a < X \leq b) = F(b) - F(a) = \int_a^b f(x)dx$$

1.2.6 Quantiles

The inverse of the cdf F, which we will refer to as F 1, exists since it is a monotonically increasing function. The the value of xα such that P(X ≤ xα) = α; if F is the cdf of X. Half of the probability mass is on the left and half is on the right, making the value F −1(0.5) the median of the distribution. The lower and upper quartiles are represented by the values F −1(0.25) and F −1(0.75).

The tail area probabilities can also be calculated using the inverse cdf. For example, if Φ is the cdf of the Gaussian distribution N (0, 1), then points to the left of Φ−1(α)/2) contain α/2 probability mass, as illustrated in Figure 2.3(b). By symmetry, points to the right of Φ−1(1−α/2) also contain α/2 of the mass. Hence the central interval (Φ−1(α/2), Φ−1(1 − α/2)) contains 1 − α of the mass. If we set α = 0.05, the central 95% interval is covered by the range(Φ−1(0.025), Φ−1(0.975)) = (−1.96, 1.96)

If the distribution is N (μ, σ2), then the 95% interval becomes (μ − 1.96σ, μ + 1.96σ). This is sometimes approximated by writing μ ± 2σ.

1.2.7 Mean and variance

A distribution's mean, or expected value, is its most well-known characteristic and is denoted by the symbol μ. It is described in terms of discrete RVs -

$$\mathbb{E}[X] \triangleq \sum_{x \in \mathcal{X}} x \; p(x)$$

and continuous RVs -

$$\mathbb{E}[X] \triangleq \int_{\mathcal{X}} x \; p(x) dx$$

The mean is not finite if this integral is not.

The variance is a measure of the "spread" of a distribution, denoted by σ^2.

This is defined as follows:

$$\begin{aligned} \text{var}[X] &\triangleq \mathbb{E}[(X-\mu)^2] = \int (x-\mu)^2 p(x) dx \\ &= \int x^2 p(x) dx + \mu^2 \int p(x) dx - 2\mu \int x p(x) dx = \mathbb{E}[X^2] - \mu^2 \end{aligned}$$

This gives us the beneficial outcome

$$\mathbb{E}[X^2] = \mu^2 + \sigma^2$$

This is how the standard deviation is described.

$$\text{std}[X] \triangleq \sqrt{\text{var}[X]}$$

It has the same units as X itself, which makes it helpful.

1.3 SOME COMMON DISCRETE DISTRIBUTIONS

There are numerous discrete probability distributions available for use in different scenarios.

1.3.1. Bernoulli Distribution

When we do an experiment just once, this distribution is produced. There are only two possible outcomes: success or failure. These kinds of trials are known as Bernoulli trials, and they serve as the foundation for many of the distributions detailed below. Let p represent the probability of success, and 1 - p represent the probability of failure.

PMF is provided as

$$\text{PMF} = \begin{cases} p, & \text{Success} \\ 1 - p, & \text{Failure} \end{cases}$$

One example of this would be a single coin flip. 1 - p is the probability of having a tail, and p is the probability of moving ahead. Please take note that how we define success and failure depends on the situation and is subjective.

1.3.2. Binomial Distribution

For random variables with just two possible outcomes, this is generated. Let p represent the probability that an event will succeed, which implies that 1 - p represents the probability that the event would fail. We obtain the Binomial distribution by repeating the experiment and charting the probability each time.

The most typical illustration of the Binomial distribution is the calculation of the probability of receiving a specific number of heads after tossing a coin n times. Other real-world examples are a company's number of productive sales calls or the efficacy of a medicine in treating a sickness.

PMF is provided as,

$$n_{C_x} p^x (1-p) x$$

where

n is the number of trials,

p is the probability of success,

x is the number of successes.

1.3.3. Hypergeometric Distribution

Think about the scenario where you draw a red marble from a box of marbles of various hues. The occurrence of drawing a red ball is successful, whereas the event of not drawing one is unsuccessful. The probability of drawing a marble in the following trial is impacted by the fact that each time a marble is drawn, it is not put back in the box. The probability of k successes over n trials, where each trial is carried out without replacement, is modelled by the hypergeometric distribution. Contrary to the binomial distribution, where the probability changes little during the course of the trials.

PMF is provided as,

$$P(X = x) = \frac{\binom{k}{x}\binom{N-k}{n-x}}{\binom{N}{n}}$$

where

k is the number of possible successes

x is the desired number of successes

N is the size of the population

n is the number of trials.

1.3.4. Poisson Distribution

The events that take place over a predetermined period of time or space are described by this distribution. This could be illustrated with an example. Think about the number of calls a customer service centre receives per hour. The average number of calls made per hour can be estimated, but the precise number and time of calls cannot be known. Every instance of an event occurs independently of all other instances.

PMF is provided as,

$$P(X = x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

where

λ is the average number of times the event has occurred in a certain period of time

x is the desired outcome

e is the Euler's number

1.3.5. Multinomial Distribution

There are just two possible outcomes in the above distributions: success and failure. Yet the random variables with numerous alternative outcomes are described by the multinomial distribution. Because each potential result is considered a distinct category, this is also frequently referred to as a categorical distribution. Think about the case where you play a game n times. We can calculate the probability that player 1 will win x_1 times, player 2 will win x_2 times, and player k will win x_k times using the multinomial distribution.

PMF is provided as,

$$P(X = x_1, X = x_2, \ldots . X = x_k) = \frac{n!}{x_1!\, x_2! \ldots . x_k!}\, {p_1}^{x_1}\, {p_2}^{x_2} \ldots . {p_k}^{x_k}$$

where

n is the number of trials

$p_{1,}$……p_k denote the probabilities of the outcomes x_1……x_k respectively.

1.4 SOME COMMON CONTINUOUS DISTRIBUTIONS

We present a few common univariate (one-dimensional) continuous probability distributions in this section.

1.4.1 Gaussian (normal) distribution

A probability distribution that is symmetric about the mean is the normal distribution, sometimes referred to as the Gaussian distribution. It demonstrates that data that are close to the mean occur more frequently than data that are far from the mean.

The normal distribution appears as a "bell curve" on a graph.

The normal distribution is defined by a number of important characteristics and attributes.

The first thing to note is that the data's mean, median, and mode (the most common observation) are all identical to one another. Furthermore, each of these values represents the distribution's peak, or highest point. The distribution then deviates symmetrically from the mean, with the standard deviation serving as a measure of its width.

The normal distribution uses the formula below. Keep in mind that only the mean (μ) and standard deviation (σ) numbers are required.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

where:

x = value of the variable or data being examined and f(x) the probability function

μ = the mean

σ = the standard deviation

1.4.2 Degenerate pdf

A degenerate random variable is a constant with probability of 1, and its distribution is known as a degenerate distribution (also known as a constant distribution). In other words, there is just one potential value for the random variable X**.**

The Gaussian becomes an indefinitely tall and infinitely thin "spike," centred atμ, in the limit that $\sigma^2 \to 0$.

$$\lim_{\sigma^2 \to 0} \mathcal{N}(x|\mu, \sigma^2) = \delta(x - \mu)$$

where δ is called a Dirac delta function, and is defined as

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases}$$

like that

$$\int_{-\infty}^{\infty} \delta(x)dx = 1$$

The sifting property of delta functions is a helpful characteristic since it allows one term to be chosen from a sum or integral:

$$\int_{-\infty}^{\infty} f(x)\delta(x - \mu)dx = f(\mu)$$

since x − μ = 0 is the only case when the integrand is not zero.

The log probability of the Gaussian distribution only decays quadratically with distance from the centre, which makes it susceptible to outliers. The Student t distribution is a more reliable distribution. Its pdf looks like this:

$$\mathcal{T}(x|\mu, \sigma^2, \nu) \propto \left[1 + \frac{1}{\nu}\left(\frac{x - \mu}{\sigma}\right)^2\right]^{-(\frac{\nu+1}{2})}$$

where

μ is the mean

$\sigma^2 > 0$ is the scale parameter

$\nu > 0$ is called the degrees of freedom.

Some properties of the distribution-

$$\text{mean} = \mu, \text{mode} = \mu, \text{ var} = \frac{\nu\sigma^2}{(\nu - 2)}$$

1.4.3 The Laplace distribution

The Laplace distribution, commonly referred to as the double-sided exponential distribution, is another distribution with heavy tails. This pdf includes the following:

$$\text{Lap}(x|\mu, b) \triangleq \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

where

μ is a location parameter

b > 0 is a scale parameter.

Here are some properties of this distribution:

mean = μ

mode = μ

var = $2b^2$

Some properties of the distribution-

$$\text{mean} = \mu,\ \text{mode} = \mu,\ \text{var} = 2b^2$$

1.4.4 The gamma distribution

For positive real valued rv's, with x > 0, the gamma distribution is a flexible distribution. The shape a > 0 and the rate b > 0 are the two parameters used to define it:

$$\text{Ga}(T|\text{shape} = a, \text{rate} = b) \triangleq \frac{b^a}{\Gamma(a)}T^{a-1}e^{-Tb}$$

where Γ(a) is the gamma function:

$$\Gamma(x) \triangleq \int_0^\infty u^{x-1}e^{-u}du$$

There are several distributions which are just special cases of the Gamma, which we discuss below:

- **Exponential distribution**

This is defined by

$$\text{Expon}(x|\lambda) \triangleq \text{Ga}(x|1, \lambda)$$

where λ is the rate parameter.

This distribution describes the times between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate λ.

- **Erlang distribution**

This is the same as the Gamma distribution where a is an integer. It is common to fix a = 2, yielding the one-parameter Erlang distribution-

Erlang(x|λ) =Ga(x|2,λ)

where λ is the rate parameter.

- **Chi-squared distribution**

This is defined by

$$\chi^2(x|\nu) \triangleq \mathrm{Ga}(x|\tfrac{\nu}{2}, \tfrac{1}{2})$$

This is the distribution of the sum of squared Gaussian random variables. More precisely, if Zi ~N(0, 1), and

$$S = \sum_{i=1}^{\nu} Z_i^2, \text{ then } S \sim \chi_\nu^2$$

Some properties of the distribution-

$$\text{mean} = \frac{b}{a-1}, \quad \text{mode} = \frac{b}{a+1}, \text{var} = \frac{b^2}{(a-1)^2(a-2)}$$

1.4.5 The beta distribution

The definition of the beta distribution, which has support between [0, 1], is as follows:

$$\mathrm{Beta}(x|a,b) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}$$

Here B(p, q) is the beta function-

$$B(a,b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Some properties of the distribution-

$$\text{mean} = \frac{a}{a+b}, \quad \text{mode} = \frac{a-1}{a+b-2}, \quad \text{var} = \frac{ab}{(a+b)^2(a+b+1)}$$

1.5 JOINT PROBABILITY DISTRIBUTIONS

We have mostly focused on modelling univariate probability distributions up to this point. The more difficult task of creating joint probability distributions on numerous related random variables is introduced in this section, which will serve as the book's main focus. A joint probability distribution, which describes the (stochastic) correlations between the

variables, takes the form p(x1,...,xD) for a set of D > 1 variables. If each variable is discrete, the joint distribution can be represented as a large, multi-dimensional array with one variable per dimension. However, O(KD), where K is the number of states for each variable, is the minimum number of parameters required to define such a model.

1.5.1 Covariance and correlation

The covariance between two rv's, X and Y, gauges how closely (linearly) X and Y are connected. In terms of covariance,

$$\operatorname{cov}[X,Y] \triangleq \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] = \mathbb{E}[XY]-\mathbb{E}[X]\mathbb{E}[Y]$$

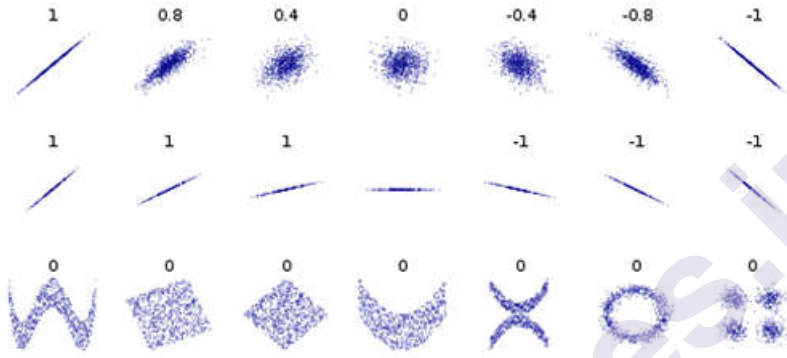


Image source:
https://en.wikipedia.org/wiki/File:Correlation_examples.png

In above image a number of sets of (x, y) points, together with the x and y correlation coefficients for each set. It should be noted that the correlation (top row) indicates the noise and direction of a linear relationship but not its slope or many other aspects of nonlinear relationships (bottom).

If x is a d-dimensional random vector, then the following symmetric, positive definite matrix is its covariance matrix:

$$\begin{aligned}\operatorname{cov}[\mathbf{x}] &\triangleq \mathbb{E}\left[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^T\right] \\ &= \begin{pmatrix} \operatorname{var}[X_1] & \operatorname{cov}[X_1,X_2] & \cdots & \operatorname{cov}[X_1,X_d] \\ \operatorname{cov}[X_2,X_1] & \operatorname{var}[X_2] & \cdots & \operatorname{cov}[X_2,X_d] \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{cov}[X_d,X_1] & \operatorname{cov}[X_d,X_2] & \cdots & \operatorname{var}[X_d] \end{pmatrix}\end{aligned}$$

Covariance's can range from 0 to infinity. Working with a normalised measure with a finite upper bound is sometimes more convenient. The formula for the (Pearson) correlation coefficient between X and Y is

$$\operatorname{corr}[X,Y] \triangleq \frac{\operatorname{cov}[X,Y]}{\sqrt{\operatorname{var}[X]\operatorname{var}[Y]}}$$

A correlation matrix has the following structure:

$$\mathbf{R} = \begin{pmatrix} \text{corr}[X_1, X_1] & \text{corr}[X_1, X_2] & \cdots & \text{corr}[X_1, X_d] \\ \vdots & \vdots & \ddots & \vdots \\ \text{corr}[X_d, X_1] & \text{corr}[X_d, X_2] & \cdots & \text{corr}[X_d, X_d] \end{pmatrix}$$

For example,

If $X=(X_1,X_2,X_3)^T$ and $Y=(Y_1,Y_2)^T$ are random vectors, then R_{XY} is a3x2 matrix whose $(i,j)^{-th}$ entry is $E[X_iY_j]$.

1.5.2 The multivariate Gaussian

The joint probability density function that is most frequently used for continuous variables is the multivariate Gaussian or multivariate normal (MVN). The pdf of the MVN in D dimensions is defined by the following:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \triangleq \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$

where

$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] \in \mathbb{R}^D$ is the mean vector

$\Sigma = \text{cov}[x]$ is the $D \times D$ covariance matrix.

Sometimes we'll work in terms of the concentration or precision matrix. Simply put, $\Lambda = \Sigma{-}1$ is the inverse covariance matrix. The pdf integrates to 1 due to the normalization constant-

$$(2\pi)^{-D/2}|\boldsymbol{\Lambda}|^{1/2}$$

1.5.3 Multivariate Student t distribution

The multivariate Student t distribution is a more accurate alternative for the MVN, and its pdf is given by-

$$\begin{aligned} \mathcal{T}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu) &= \frac{\Gamma(\nu/2 + D/2)}{\Gamma(\nu/2)} \frac{|\boldsymbol{\Sigma}|^{-1/2}}{\nu^{D/2}\pi^{D/2}} \times \left[1 + \frac{1}{\nu}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]^{-(\frac{\nu+D}{2})} \\ &= \frac{\Gamma(\nu/2 + D/2)}{\Gamma(\nu/2)}|\pi\mathbf{V}|^{-1/2} \times \left[1 + (\mathbf{x}-\boldsymbol{\mu})^T\mathbf{V}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]^{-(\frac{\nu+D}{2})} \end{aligned}$$

where

Σ is called the scale matrix (since it is not exactly the covariance matrix)

$V = \nu\Sigma$.

Compared to a Gaussian, this has fatter tails. The tails get fatter as the ν → ∞ gets smaller.

The distribution is more likely to be Gaussian, as stated. These are the properties of the distribution.

$$\text{mean} = \mu, \quad \text{mode} = \mu, \quad \text{Cov} = \frac{\nu}{\nu - 2}\Sigma$$

1.6 TRANSFORMATIONS OF RANDOM VARIABLES

What is the distribution of y if x is some random variable, x ~p(), and y=f(x)?

The following will reveal the answer-

1.6.1 Linear transformations

If $f()$ is a linear function, then:

$y=f(x)=Ax+b$

The mean and covariance of y in this situation can be easily derived as follows. First, we have for the mean:

$E[y]=E[Ax+b]=A\mu+b$

where μ = E [x]. This is called the linearity of expectation.

If $f()$is a scalar-valued function,

$f(x)= a^Tx + b$, the corresponding result is

$$\mathbb{E}\left[\mathbf{a}^T\mathbf{x} + b\right] = \mathbf{a}^T\boldsymbol{\mu} + b$$

For the covariance, we have

$$\operatorname{cov}\left[\mathbf{y}\right] = \operatorname{cov}\left[\mathbf{A}\mathbf{x} + \mathbf{b}\right] = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T$$

where Σ = cov [x].

We leave the proof of this as an exercise. If $f()$ is scalar valued, the result becomes

$$\operatorname{var}\left[y\right] = \operatorname{var}\left[\mathbf{a}^T\mathbf{x} + b\right] = \mathbf{a}^T\boldsymbol{\Sigma}\mathbf{a}$$

1.6.2 General transformations

The probability mass for all the x's such that f(x) = y can be summed together to obtain the pmf for y if X is a discrete rv.

$$p_y(y) = \sum_{x:f(x)=y} p_x(x)$$

1.6.3 Central limit theorem

Now imagine N random variables, each with mean and variance σ^2, and pdfs (not necessarily Gaussian) $p(x_i)$. We take for granted that all of the variables have independent and identical distributed, or iid.

Let

$$S_N = \sum_{i=1}^{N} X_i$$

be the sum of the rv's. This conversion of RVs is simple yet very common. One can demonstrate how the distribution of this sum approaches uniformity as N rises.

$$p(S_N = s) = \frac{1}{\sqrt{2\pi N\sigma^2}} \exp\left(-\frac{(s - N\mu)^2}{2N\sigma^2}\right)$$

And hence, the quantity's distribution

$$Z_N \triangleq \frac{S_N - N\mu}{\sigma\sqrt{N}} = \frac{\overline{X} - \mu}{\sigma/\sqrt{N}}$$

the standard normal is reached, where

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

is the sample mean.

This is called the central limit theorem.

1.7 MONTE CARLO APPROXIMATION

In general, it might be challenging to compute the distribution of a function of a rv using the change of variables formula. Here is a simple yet effective alternative. We first create S samples from the distribution, denoted by the letters $x_1,..., x_S$. Using the empirical distribution of, we may approximatively determine the distribution of f(X) given the data.

$$\{f(x_s)\}_{s=1}^{S}$$

This is known as a Monte Carlo approximation, after the European city famous for its plush gambling casinos. Although they were initially created in the field of statistical physics, specifically during the development of the atomic bomb, Monte Carlo techniques are now widely used in both statistics and machine learning.

Any function of a random variable can have an expected value, and we can use Monte Carlo to approximate it. We only create samples, after which we calculate the function's arithmetic mean when it is applied to the samples. The following can be written:

$$\mathbb{E}[f(X)] = \int f(x)p(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x_s)$$

where $x_s \sim p(X)$.

This process, known as Monte Carlo integration, has the benefit of only evaluating the function at locations where there is a non-zero probability, as opposed to numerical integration, which is predicated on doing so at a set grid of points.

Many values of interest can be approximated by changing the value of the function f(), such as:

$$\overline{x} = \frac{1}{S}\sum_{s=1}^{S} x_s \to \mathbb{E}[X]$$

$$\frac{1}{S}\sum_{s=1}^{S}(x_s - \overline{x})^2 \to \text{var}[X]$$

$$\frac{1}{S}\#\{x_s \leq c\} \to P(X \leq c)$$

1.7.1 Example: change of variables, the MC way

Let y = x2 and x ~ Unif(-1, 1). By taking many samples from p(x), squaring them, and then estimating the resulting empirical distribution, we can approximate p(y), explained in the image below.

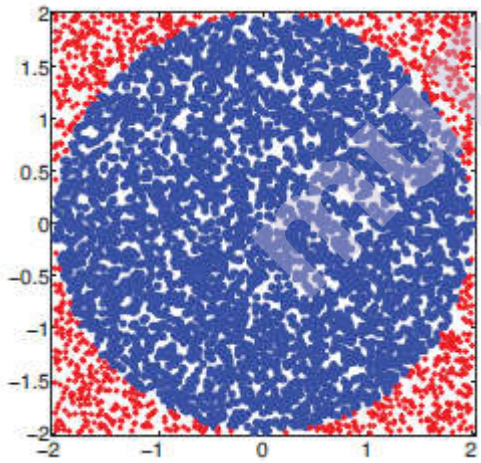


Image source: Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).

Monte Carlo integration is used to estimate π. The circle has red crosses outside and blue points inside.

1.7.2 Example: estimating π by Monte Carlo integration

Not only for statistical purposes, but also for a wide range of other uses. Let's say we wish to calculate π. We are aware that the area of a circle

with radius r is equal to πr^2, but it is also equal to the following definite integral-

$$I = \int_{-r}^{r} \int_{-r}^{r} \mathbb{I}(x^2 + y^2 \leq r^2) dx dy$$

Hence π = I/(r2). Let us approximate this by Monte Carlo integration. Let f(x, y) =

I(x2 + y2 ≤ r2) be an indicator function that is 1 for points inside the circle, and 0 outside, and let p(x) and p(y) be uniform distributions on [−r, r], so p(x) = p(y) = 1/(2r). Then

$$\begin{aligned} I &= (2r)(2r) \int \int f(x,y) p(x) p(y) dx dy \\ &= 4r^2 \int \int f(x,y) p(x) p(y) dx dy \\ &\approx 4r^2 \frac{1}{S} \sum_{s=1}^{S} f(x_s, y_s) \end{aligned}$$

We find ˆπ = 3.1416 with standard error 0.09

1.7.3 Accuracy of Monte Carlo approximation

Example 1:

The probability that the actual return will be within one standard deviation of the rate that is considered to be the most likely ("expected") is 68%. There is a 95% chance that it will be within two standard deviations and a 99.7% chance that it will be within three.

Yet, there is no guarantee that the outcome will be as expected or that real movements won't exceed the most extreme predictions.

Example 2:

If we denote the exact mean by μ = E [f(X)], and the MC approximation by ˆμ, one can show that, with independent samples,

$$(\hat{\mu} - \mu) \rightarrow \mathcal{N}(0, \frac{\sigma^2}{S})$$

where

$$\sigma^2 = \text{var}\left[f(X)\right] = \mathbb{E}\left[f(X)^2\right] - \mathbb{E}\left[f(X)\right]^2$$

This is a consequence of the central-limit theorem. Of course, $\sigma^2$is unknown in the above expression, but it can also be estimated by MC:

$$\hat{\sigma}^2 = \frac{1}{S}\sum_{s=1}^{S}(f(x_s) - \hat{\mu})^2$$

Then we have

$$P\left\{\mu - 1.96\frac{\hat{\sigma}}{\sqrt{S}} \leq \hat{\mu} \leq \mu + 1.96\frac{\hat{\sigma}}{\sqrt{S}}\right\} \approx 0.95$$

The term $\sqrt{\frac{\hat{\sigma}^2}{S}}$ is called the (numerical or empirical) **standard error**, and is an estimate of our uncertainty about our estimate of μ.

1.8 INFORMATION THEORY

Data compression, also known as source coding, is the process of encoding data in a little amount of space. Information theory is also concerned with how to transport and store data in a way that is robust to errors (a task known as error correction or channel coding). Although at first glance it might appear that this has little to do with probability theory's concerns, there is actually a close connection. Consider the fact that compactly expressing data necessitates allocating short code words to highly probable bit strings and reserving longer code words for less probable bit strings to demonstrate this. Similar to how frequent words (as such "a," "the," and "and") likely to be much shorter than rare ones in natural language.

1.8.1 Entropy

A random variable's entropy, indicated by □(X) or sometimes □(p), is a measure of its uncertainty. It is defined by, in specifically, for a discrete variable with K states-

$$\mathbb{H}(X) \triangleq -\sum_{k=1}^{K} p(X = k)\log_2 p(X = k)$$

Typically, log base 2 is used, in which case the units are referred to as bits (short for binary digits). The units are referred to as nats if we use log base e. For instance, we find □ = 2.2855 if X ∈ {1, . . . , 5} with histogram distribution p = [0.25, 0.25, 0.2, 0.15, 0.15]. The uniform distribution is the discrete distribution with the highest entropy. The entropy is therefore maximum for a K-ary random variable if p(x = k) = 1/K; in this case, □(X) = log2 K. In contrast, any delta-function that concentrates all of its mass in one state is the distribution with minimum entropy, which is zero. There is no uncertainty in this distribution.

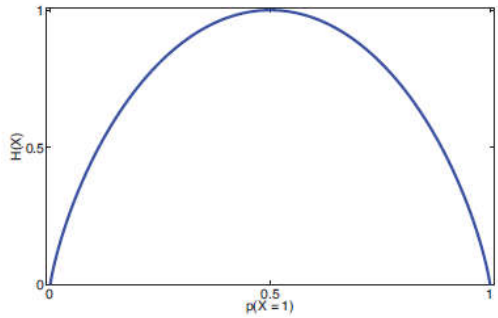


Image source: Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).

Entropy of a Bernoulli random variable as a function of θ.

The maximum entropy is $\log_2 2 = 1$.

For the special case of binary random variables, $X \in \{0, 1\}$, we can write $p(X = 1) = \theta$ and $p(X = 0) = 1 - \theta$. Hence the entropy becomes

$$\begin{aligned}\mathbb{H}(X) &= -[p(X=1)\log_2 p(X=1) + p(X=0)\log_2 p(X=0)] \\ &= -[\theta \log_2 \theta + (1-\theta)\log_2(1-\theta)]\end{aligned}$$

This is called the binary entropy function, and is also written H(θ).

1.8.2 KL divergence

The Kullback-Leibler Divergence score, or KL divergence score, quantifies how much one probability distribution differs from another probability distribution.

The KL divergence between two distributions q and p is often stated using the following notation:

KL(p || q)

Where the "||" operator indicates "divergence" or p's divergence from q.

This is defined as follows:

$$\mathbb{KL}(p||q) \triangleq \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}$$

We can rewrite this as

$$\mathbb{KL}(p||q) = \sum_k p_k \log p_k - \sum_k p_k \log q_k = -\mathbb{H}(p) + \mathbb{H}(p,q)$$

where H (p, q) is called the cross entropy,

$$\mathbb{H}(p,q) \triangleq -\sum_k p_k \log q_k$$

The value within the sum is the divergence for a given event.

1.8.3 Mutual information

Think about the two random variables X and Y. Let's say we want to see how much understanding one variable can reveal about the other. We could compute the correlation coefficient, but this is a highly limiting measure of dependence since it is only defined for random variables with real values. Determine how comparable the joint distribution p(X, Y) is to the factored distribution p(X)p(Y). The mutual information, or MI, is what is meant by the following:

$$\mathbb{I}(X;Y) \triangleq \mathbb{KL}(p(X,Y)||p(X)p(Y)) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

We have I (X; Y) ≥ 0 with equality iff p(X, Y) = p(X)p(Y).

In other words, if the variables are independent, the MI is zero. It is helpful to re-express MI in terms of joint and conditional entropies to better understanding of its significance. The pointwise mutual information, or PMI, is a quantity that is closely connected to MI. This is defined as follows for two occurrences (not random variables) x and y:

$$\text{PMI}(x,y) \triangleq \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}$$

This calculates the difference between the probability of these occurrences happening together and what would be anticipated by chance.

1.9 REFERENCES

- Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).
- https://en.wikipedia.org/wiki/Probability
- https://www.analyticsvidhya.com/blog/2021/01/discrete-probability-distributions/
- https://en.wikipedia.org/wiki/Inequalities_in_information_theory
- https://theclevermachine.wordpress.com/2012/09/22/monte-carlo-approximations/
- Introducing Monte Carlo Methods with R, Christian P. Robert, George Casella, Springer, 2010.
- Introduction to Machine Learning (Third Edition): EthemAlpaydın, The MIT Press (2015).

➢ Pattern Recognition and Machine Learning: Christopher M. Bishop, Springer (2006).

1.10 QUESTIONS

- Explain Bayes rule in Probability.
- Write a note on Independence and conditional independence. Give example.
- Describe Continuous random variables and Quantiles.
- What is Mean and variance? Explain Some common discrete distributions in short.
- Write a note on Bernoulli distributions and Binomial distributions.
- Write a note on Some common continuous distributions.
- Write a note on gamma distribution and beta distribution.
- Describe Multivariate Student t distribution.
- Explain Linear transformations and General transformations.
- Write a note on Monte Carlo approximation.
- What is Entropy? Explain KL divergence.

2

DIRECTED GRAPHICAL MODELS

Unit Structure:

2.0 OBJECTIVES

After completing this chapter, you will be able to understand directed graphical models (Bayes nets), inference, learning, conditional independence properties of DGMs. Mixture models and EM algorithm in that Latent variable models, Mixture models, Parameter estimation for mixture models, The EM algorithm.

2.1 DIRECTED GRAPHICAL MODELS (BAYES NETS):

A Bayesian network is a directed acyclic graph(DAG), and directed graphical models are sometimes referred to as directed edges give causality links between random variables.

2.1.1 Introduction

Graphical models give a visual representation of a joint probability distribution's underlying structure. The structure encodes information on the conditional independence relationships between the random variables,

as we'll see shortly. Remember that these links between independence are crucial for comprehending the computational costs of representation and inference for a given joint probability distribution. Our first objective is to utilise the model to respond to straightforward inquiries like "Are the random variables X_A and X_B independent?" or "Is the random variable X_A independent of the random variable X_B conditioned on the random variable X_C?" Although these questions appear straightforward, the Bayes rule is the only method we have found so far to provide the answer.

Chain rule

The general product rule is another name for the Chain Rule of Conditional Probabilities. Any number of the associated distributions of a set of random variables can be calculated using it. By only using conditional probabilities, it is possible.

The Chain rule can be obtained by rearranging the conditional probability formula:

P (A, B) = P (A|B) P (B)

This can be scaled for three different variables:

P(A,B,C) = P(A| B,C) P(B,C) = P(A|B,C) P(B|C) P(C)

and commonly to n variables:

P(A1, A2, …, An) = P(A1| A2, …, An) P(A2| A3, …, An) P(An-1|An) P(An)

This is generally referred to as the chain rule.

For Bayesian Belief Nets, this formula is important. It provides a method for finding out the complete joint probability distribution. A probability measure is the conditional probability of the aforementioned.

Example

The chain rule is applicable to four events (n=4).

$$\begin{aligned}\mathrm{P}(A_1 \cap A_2 \cap A_3 \cap A_4) &= \mathrm{P}(A_4 \mid A_3 \cap A_2 \cap A_1) \cdot \mathrm{P}(A_3 \cap A_2 \cap A_1) \\ &= \mathrm{P}(A_4 \mid A_3 \cap A_2 \cap A_1) \cdot \mathrm{P}(A_3 \mid A_2 \cap A_1) \cdot \mathrm{P}(A_2 \cap A_1) \\ &= \mathrm{P}(A_4 \mid A_3 \cap A_2 \cap A_1) \cdot \mathrm{P}(A_3 \mid A_2 \cap A_1) \cdot \mathrm{P}(A_2 \mid A_1) \cdot \mathrm{P}(A_1)\end{aligned}$$

Conditional independence

Applying some assumptions regarding conditional independence(CI) is essential for effectively representing large joint distributions . If and only if (iff) the conditional joint can be represented as a product of the conditional marginal, then X and Y are conditionally independent given Z, denoted $X \perp Y|Z$,

$$X \perp Y|Z \iff p(X,Y|Z) = p(X|Z)p(Y|Z)$$

Let's examine how this might be helpful. Assume that xt+1 ⊥ x1:t−1|xt, or, to put it another way, "the future is independent of the past given the present." This is referred to as the (first order) Markov assumption. The joint distribution can be expressed as follows using this assumption and the chain rule:

$$p(\mathbf{x}_{1:V}) = p(x_1) \prod_{t=1}^{V} p(x_t|x_{t-1})$$

A (first-order) Markov chain is what this is. A state transition matrix p(xt = j|xt−1 = i) in combination to an initial distribution over states, p(x1 = i) can be used to describe them.

Graphical models

A framework for reasoning about uncertain quantities and their structural links is provided by graphical models. They combine graph theory and probability. Random variables are represented as nodes, while their connections or relationships are shown as edges.

Similar to a circuit diagram, graphic representations of a problem are recorded to help with visualisation and comprehension.

Graphical models can be viewed as a:

- A communication tool that makes it easier to succinctly convey how many opinions about a system are interconnected.
- A tool for reasoning that enables the extraction of connections that were not immediately apparent when the problem was formulated. Visualizing conditional independence is made possible, in particular, by graphical models.
- A computational skeleton that improves the way we compute with random variables.

Graph terminology

A **graph** G = (V, E) includes a set of nodes or vertices, V = {1, . . . , V }, and a set of **edges**, E = {(s, t) : s, t ∈ V}. The graph's adjacency matrix allows us to represent it, in which we write G(s, t) = 1 to denote (s, t) ∈ E, that is, if s → t is an edge in the graph.

1. Parent: Node an is the parent of node b if there is an edge connecting them.
2. Child: If node a and node b are connected by an edge, then node b is a child of node a.
3. Root: A root is a node that has no parents. It only has outgoing edges.

4. Leaves or Leaf: A leaf is a node that has no children. There are no outgoing edges.

5. Ancestors and Descendants: If a directed path connects node a to node b, node a is a descendant of node b and vice versa.

6. Acyclic: For each node I anc(i) does not contain i, i.e. $\forall i, i /\in anc(i)$.

7. Degree: A node's degree is determined by how many neighbours it has. We use the terms in-degree and out-degree, which count the number of parents and kids, respectively, for directed graphs.

8. Cycle and loop: An edge connecting a vertex to itself is called a loop. A cycle is a path that starts and ends at the same node.

9. DAG: A directed graph without any directed cycles is known as a directed acyclic graph, or DAG.

10. Topological ordering: For a DAG, a topological ordering, also known as a total ordering, is a node numbering in which parents are given less nodes than their children.

11. Path or trail: From $s \rightsquigarrow t$, a path or trail is formed by a series of directed edges.

12. Tree: An undirected tree is an undirectecd graph with no cycles.

13. Forest: A forest is a set of trees.

14. Clique: For an undirected graph, a clique is a set of nodes that are all neighbours of each other.

2.1.2 Examples

In this part, we demonstrate how many different commonly used probabilistic models can be simply described as DGMs.

Bayes net model describing the performance of a student on an exam. The distribution can be represented a product of conditional probability distributions specified by tables. The form of these distributions is described by edges in the graph:

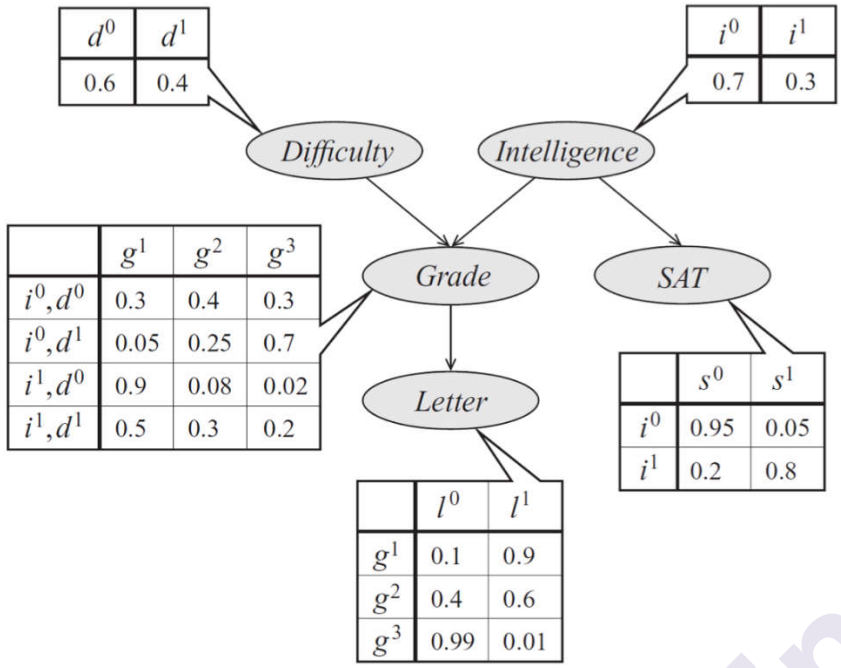


Image Source: https://ermongroup.github.io/cs228-notes/assets/img/grade-model.png

Naive Bayes classifiers

A collection of classification algorithms built on the Bayes' Theorem are known as naive Bayes classifiers. It is a family of algorithms rather than a single method, and they are all based on the idea that every pair of features being classified is independent of the other. The following joint distribution:

$$p(y, \mathbf{x}) = p(y) \prod_{j=1}^{D} p(x_j|y)$$

The naïve Bayes assumption is relatively naive because it thinks the characteristics are conditionally independent. Using a graphical model is one method of capturing correlation between the features. If the model is a tree, the method is known as a tree-augmented naive Bayes classifier, or TAN model.

Markov and hidden Markov models

A hidden Markov model (HMM) is one in which you witness a sequence of emissions but have no idea what states the model went through to generate the emissions. The goal of hidden Markov model analyses is to reconstruct the sequence from the observed data.

The assumption that the near past, xt1xt−1, has all of the information we need to know about the entire history, x1:t−2, is a bit too strong. By including a dependency from xt-2 to xt as well, we can loosen it up a bit;

this is known as a second order Markov chain. The relevant joint is shaped as follows:

$$p(\mathbf{x}_{1:T}) = p(x_1, x_2)p(x_3|x_1, x_2)p(x_4|x_2, x_3)\ldots = p(x_1, x_2)\prod_{t=3}^{T} p(x_t|x_{t-1}, x_{t-2})$$

Similar techniques can be used to build higher-order Markov models.

Example

Imagine you have a bag of marbles with two red and two blue marbles inside, totaling four marbles. A marble is drawn at random from the bag, its colour noted, and it is then placed back in the bag. When you go through this process repeatedly, you start to see a pattern: A red marble is always two out of four times, or 50%, likely to be chosen. This is so because the quantity of a certain marble colour in the bag affects the likelihood of choosing that colour.

This example demonstrates the Markov model concept: the future state of a system is determined by its current state and past history. The present state of the bag of marbles is defined by the number of each colour of marble in the bag. The contents of the bag symbolise the previous history, and they determine the chances of selecting each colour of marble.

Medical Scenario: Hidden Markov models are utilized in various of medical applications to try to discover the hidden states of a human body system or organ. Cancer diagnosis, for example, can be done by examining specific sequences and deciding how dangerous they may be to the patient. Hidden Markov models are also used to evaluate biological data such as RNA-Seq, ChIP-Seq, and others that assist researchers understand gene regulation. Doctors can forecast people's life expectancy based on their age, weight, height, and body type using the hidden Markov model.

2.1.3 Inference

We've seen how graphical models can be used to define joint probability distributions in a concise manner.

What can we do with such a joint distribution? The major application of such a joint distribution is to do probabilistic inference. This task involves estimating unknown quantities from known quantities. In general, the inference problem can be stated as follows. Consider a collection of correlated random variables with the joint distribution p(x1:V |θ). Let us divide this vector into visible variables xv that are observed and hidden variables xh that are unobserved. Inference is the process of calculating the posterior distribution of unknowns given knowns:

$$p(\mathbf{x}_h|\mathbf{x}_v, \boldsymbol{\theta}) = \frac{p(\mathbf{x}_h, \mathbf{x}_v|\boldsymbol{\theta})}{p(\mathbf{x}_v|\boldsymbol{\theta})} = \frac{p(\mathbf{x}_h, \mathbf{x}_v|\boldsymbol{\theta})}{\sum_{\mathbf{x}'_h} p(\mathbf{x}'_h, \mathbf{x}_v|\boldsymbol{\theta})}$$

We're just conditioning the data by clamping the visible variables to their observed values, xv, and then normalising to get from p(xh, xv) to p(xh|xv). The likelihood of the data, also known as the probability of the evidence, is represented by the normalisation constant p(xv|θ).

Example- A Bernoulli (Boolean) random variable, could express the event that John has cancer. A variable of this type could have a value of 1 (John has cancer) or 0. (John does not have cancer). Infernce use probabilistic inference to calculate the likelihood that the random variable will take the value 1: a probability of 0.78 indicates that John is 78% likely to develop cancer.

2.1.4 Learning

Inference and learning are frequently distinguished in the works on graphical models. Computing (functions of) p(xh|xv, θ), where v are the visible nodes, h are the hidden nodes, and are the model's parameters, which are assumed to be known, is what is meant by inference. Most of the time, learning involves calculating a MAP estimate of the parameters given data:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(\mathbf{x}_{i,v}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

where xi,v are the visible variables in case i.

If we have a uniform prior, p(θ) ∝ 1, this reduces to the MLE, as usual. According to a Bayesian perspective, the parameters are likewise unknown variables that need to be inferred. Hence, there is no difference between inference and learning to a Bayesian. In fact, all we have to do is add the parameters as nodes to the graph, apply a condition based on D, and infer the values of each node.

According to this perspective, the primary distinction between hidden variables and parameters is that the number of hidden variables typically increases with the amount of training data (because there is typically a set of hidden variables for each observed data case) (at least in a parametric model). This indicates that in order to prevent overfitting, we must integrate out the hidden variables, but for the parameters, which are few in number, we might be able to get away with point estimation methods.

Plate Notation

In a graphical model, repeating variables are represented using plate notation. A plate or rectangle is used to organise variables into a subgraph that repeats collectively rather than drawing each repeated variable separately, and a number is drawn on the plate to show the number of repetitions of the subgraph in the plate.

Think about the next simple model. A Gaussian with mean μ and standard deviation σ is used to create a data set of N points:

$$p(x_1, \ldots, x_N, \mu, \sigma) = p(\mu)p(\sigma) \prod_{n=1}^{N} p(x_n|\mu, \sigma)$$

This is illustrated graphically as follows:

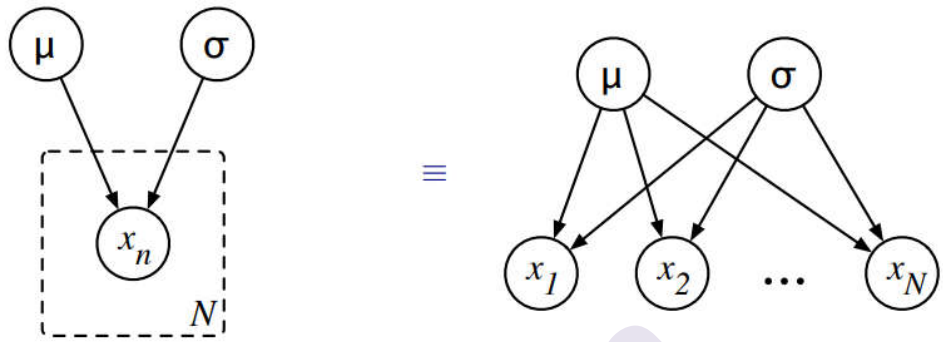
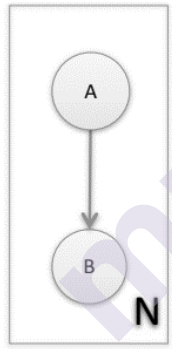


Image source: http://mlss.tuebingen.mpg.de/2017/speaker_slides/Zoubin3.pdf

Example

Students and their Grades.



A=student, B=grade

Learning from complete data

We say the data is complete if all variables are fully observed in each case, therefore there is no missing data and no hidden variables. The probability for a DGM with complete data is provided by:

$$p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(\mathbf{x}_i|\theta) = \prod_{i=1}^{N} \prod_{t=1}^{V} p(x_{it}|\mathbf{x}_{i,\mathrm{pa}(t)}, \theta_t) = \prod_{t=1}^{V} p(\mathcal{D}_t|\theta_t)$$

where Dt is the data associated with node t and its parents, i.e., the t'th family.

This is a collection of terms, one for each CPD. The probability decomposes according to the graph structure, as the name implies.

Assume that the preceding factorises as well:

$$p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\boldsymbol{\theta}_t)$$

The posterior then obviously factors as well:

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = \prod_{t=1}^{V} p(\mathcal{D}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t)$$

This enables us to independently calculate the posterior of each CPD. In other words, factored prior and probability together imply factored posterior.

Learning with missing and/or latent variables

When we have missing data and/or hidden variables, the probability no longer factorises and is no longer convex. This means that we can usually only compute a locally optimal ML or MAP estimate. Bayesian parameter inference is even more difficult.

2.1.5 Conditional independence properties of DGMs

A set of conditional independence (CI) assumptions is at the heart of any graphical model. Using the semantics provided below, we write xA⊥G xB|xC if A is independent of B given C in the graph G. Let I(G) be the collection of all CI statements encoded by the graph. G is an I-map (independence map) for p, or p is Markov with respect to G, iff I(G) ⊆ I(p), where I(p) is the set of all CI statements that hold for distribution p. In other words, the graph is an I-map if it makes no CI assertions that are false about the distribution. When reasoning about p's CI properties, we can use the graph as a safe proxy for p. This is essential for designing algorithms that operate for a large variety of distributions, regardless of their specific numerical parameters θ.

d-separation and the Bayes Ball algorithm (global Markov properties)

First, we introduce some definitions. We say an undirected path P is d-separated by a set of nodes E (containing the evidence) iff at least one of the following conditions hold:

1. P contains a chain, s → m → t or s ← m ← t, where m ∈ E

2. P contains a tent or fork, s ↙m↘ t, where m ∈ E

3. P contains a collider or v-structure, s ↘m↙ t, where m is not in E and nor is any descendant of m.

Next, we say that a set of nodes A is d-separated from a different set of nodes B given a third observed set E iff each undirected path from every node a ∈ A to every node b ∈ B is d-separated by E. Finally, we define the CI properties of a DAG as follows:

xA⊥G xB|xE⇐⇒ A is d-separated from B given E

2.2 MIXTURE MODELS AND EM ALGORITHM:

A mixture model in statistics is a probabilistic model for describing the presence of subpopulations within an aggregate population that does not require an observed data set to identify the subpopulation to which an individual observation belongs. A mixture model is defined as the mixture distribution, which represents the probability distribution of observations in the whole population.

2.2.1 Latent variable models

A statistical model that links a group of observable variables to a group of latent variables is known as a latent variable model. An alternate strategy is to assume that the correlation between the observed variables results from a shared "cause" that is hidden. Latent variable models, or LVMs, are another name for models with hidden variables. They may, however, offer a number of benefits for two major reasons. Secondly, compared to models that directly reflect correlation in the visible space, LVMs frequently have fewer parameters. The computation of a compressed version of the data is slowed down by the hidden variables in an LVM, which might act as a bottleneck.

There are L latent variables, $z_{i1},..., z_{IL}$, and D visible variables, $x_{i1},..., x_{iD}$, where D ≫ L is usually L. If L > 1, we have a many-to-many mapping since each observation is affected by numerous latent variables. Z_i is typically discrete in this scenario, and we have a one-to-many mapping if L = 1. Otherwise, we only have a single latent variable.

2.2.2 Mixture models

A mixture model is a type of probabilistic model that assumes the data were generated by the following process:

- Choose at random one of the mixture's ingredients.

-Get a sample of the data from the distribution corresponding to that mixed component.

Let's say our goal is to simulate the cost of a particular book. It could make sense to model the price of paperback books separately from hardback books since paperback books are often less expensive than hardbacks. We'll use a mixture model to simulate the cost of a book in this example. Our model will have two mixture components: one for hardbacks and one for paperback books.

The simplest form of LVM is when $z_i \in \{1, \ldots, K\}$, representing a discrete latent state. We will use a discrete prior for this, $p(z_i) = \text{Cat}(\pi)$. For the likelihood, we use $p(x_i|z_i = k) = p_k(x_i)$, where p_k is the k'th base distribution for the observations; this can be of any type. The overall model is known as a mixture model, since we are mixing together the K base distributions as follows:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}_i|\boldsymbol{\theta})$$

This is a convex combination of the pk's, since we are taking a weighted sum, where themixing weights π_k satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$.

Mixtures of Gaussians

The mixture of Gaussians (MOG), often known as a Gaussian mixture model or GMM, is the most frequently used mixture model. Each base distribution in the mixture in this model is a multivariate Gaussian with a mean μ_k and a covariance matrix of length Σ_k. the model has the following form:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Several sets of eliptical contours are used to represent the various mixture components. A GMM can be used to approximate any density defined on $\mathbb{R}^D$ if there are enough mixing components.

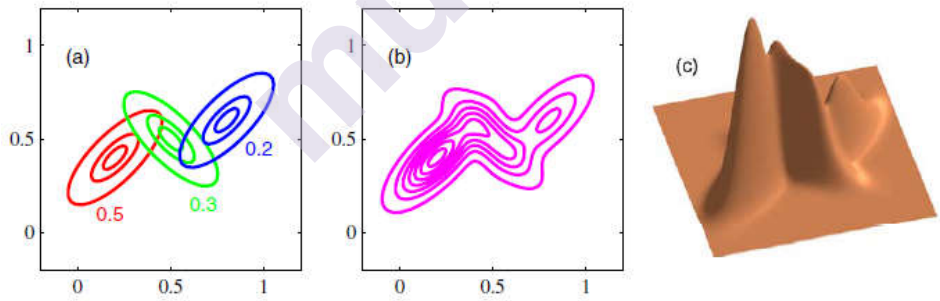


Image Source: https://ermongroup.github.io/cs228-notes/assets/img/gmm2.png

Illustration of a mixture of 3 Gaussians in a two-dimensional space. (a) Contours of constant density for each of the mixture components, in which the 3 components are denoted red, blue and green, and the values of the mixing coefficients are shown below each component. (b) Contours of the marginal probability density p(x) of the mixture distribution. (c) A surface plot of the distribution p(x).

2.2.3 Parameter estimation for mixture models

The concept of optimization is used to offer a strategy for parameter estimation. Assuming the parameters are known, we have shown how to compute the posterior over the hidden variables given the observed variables.

We shown in Section Learning from Complete Data that when we have complete data and a factored prior, the posterior over the parameters likewise factors, making calculation very straightforward. Unfortunately, if we have hidden variables and/or missing data, this is no longer the case.If the z_i were seen, then the posterior will factorise since, according to d-separation, $\theta z \perp \theta x|D$. The posterior does not factorise and the parameters are no longer independent in an LVM because of the hidden z_i, which makes computation much more difficult. Moreover, this makes it more difficult to calculate MAP and ML estimates.

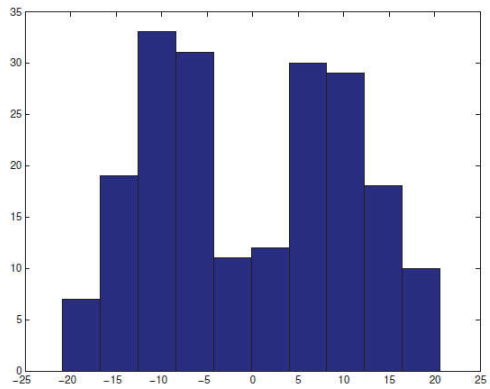
Unidentifiability

The fundamental issue with determining $p(\theta|D)$ for an LVM is that the posterior may have several modes. Think about a GMM to see why. If all of the zi were observed, the parameters would have a unimodal posterior:

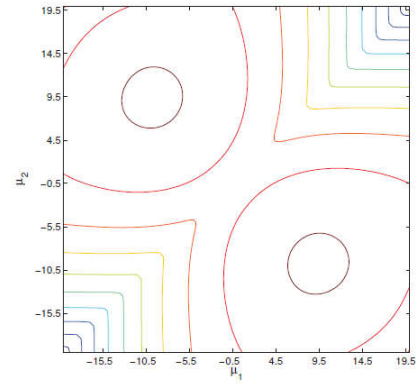
$$p(\theta|\mathcal{D}) = \mathrm{Dir}(\pi|\mathcal{D}) \prod_{k=1}^{K} \mathrm{NIW}(\mu_k, \Sigma_k|\mathcal{D})$$

Hence, we can quickly identify the MAP estimate that is globally optimal (and hence globally optimal MLE).

But let's say the z_i's aren't visible. In this scenario, we obtain a different unimodal probability for each potential method of "filling in" the z_i's. As a result, when we ignore the z_i's, weget a multi-modal posterior for $p(\theta|D)$.These modes correspond to various cluster labelings. This is illustrated in following Figure (b), where we plot the likelihood function, $p(D|\mu 1, \mu 2)$,for a 2D GMM with K = 2 for the data is shown in following Figure (a).



(a)



(b)

Left: N = 200 data points sampled from a mixture of 2 Gaussians in 1d, with πk = 0.5, σk = 5, μ1 = −10 and μ2 = 10. Right: Likelihood surface p(D|μ1, μ2), with all other parameters set

to their true values. We see the two symmetric modes, reflecting the unidentifiability of the parameters.

Image Source: Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).

We see two peaks, onecorresponding to the case where μ1 = −10, μ2 = 10, and the other to the case where μ1 = 10,μ2 = −10. We say the parameters are not **identifiable**, since there is not a unique MLE.

Therefore there cannot be a unique MAP estimate (assuming the prior does not rule out certainlabelling), and hence the posterior must be multimodal. The question of how many modes there are in the parameter posterior is hard to answer. There are K! possible labelings, but some of the peaks might get merged. Nevertheless, there can be an exponential number, since finding the optimal MLE for a GMM is NP-hard (Aloise et al. 2009; Drineas et al. 2004).

Unidentifiability can cause a problem for Bayesian inference. For example, suppose we draw some samples from the posterior, θ(s) ∼p(θ|D), and then average them, to try to approximate the posterior mean, $\overline{\boldsymbol{\theta}} = \frac{1}{S}\sum_{s=1}^{S}\boldsymbol{\theta}^{(s)}$ If the samples come from different modes, the average will be meaningless. Note, however, that it is reasonable to average the posterior predictive distributions, $p(\mathbf{x}) \approx \frac{1}{S}\sum_{s=1}^{S} p(\mathbf{x}|\boldsymbol{\theta}^{(s)})$, since the likelihood function is invariant to which mode the parameters came from.

Only two latent parameters, each of which receives N data points, are present. As a result, the posterior uncertainty about the parameters is usually significantly smaller than the posterior uncertainty regarding the latent variables.

The parameters have the ability to communicate with one another. This would not be achievable if we were to use a point estimate.

Computing a MAP estimate is non-convex

We have stated, rather heuristically, in the preceding sections that getting a MAP or ML estimate will be challenging since the likelihood function has numerous modes. In this section, we demonstrate this finding using more algebraic techniques, which provides some additional insight for the issue:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_i \log\left[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})\right]$$

Unfortunately, it is challenging to achieve this goal. therefore we are unable to insert the log into the sum. This rules out some algebraic

inferences, but it doesn't show that the issue is difficult. Now suppose the joint probability distribution $p(\mathbf{z}_i, \mathbf{x}_i|\boldsymbol{\theta})$ is in the exponential family, which means it can be written as follows:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp[\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}, \mathbf{z})]$$

where φ(x, z) are the sufficient statistics, and Z(θ) is the normalization constant. With this assumption, the complete data log likelihood can be written as follows:

$$\ell_c(\boldsymbol{\theta}) = \sum_i \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \boldsymbol{\theta}^T (\sum_i \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{z}_i)) - NZ(\boldsymbol{\theta})$$

The first term in θ is obviously linear. As Z(θ) can be demonstrated to be a convex function (thanks to the minus sign), the total objective is concave and so has a single maximum. Now think about what transpires when we lack data. The likelihood of the observed data is provided by:

$$\ell(\boldsymbol{\theta}) = \sum_i \log \sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \sum_i \log \left[\sum_{\mathbf{z}_i} e^{\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{z}_i, \mathbf{x}_i)} \right] - N \log Z(\boldsymbol{\theta})$$

One can show that the log-sum-exp function is convex, and we know that Z(θ) is convex. On the other hand, the difference between two convex functions is typically not convex. Hence, the objective has local optima and is neither convex nor concave.

The downside of non-convex functions is that it is sometimes difficult to identify their global optimum.

A local optimum is all that the majority of optimization algorithms can find; which one they find depends on where they start.

In real-world scenarios, we'll run a local optimizer and possibly employ a number of random restarts to improve our chances of locating a "good" local optimum. Obviously, careful initialization can be quite beneficial as well.

2.2.4 The EM algorithm

For many models in machine learning and statistics, computing the ML or MAP parameter estimate is easy if we see all the values of all the relevant random variables, i.e., if we have complete data.When we have missing data and/or latent variables, however, computing the ML/MAP estimate becomes difficult. Finding a local minimum of the negative log likelihood, or NLL, as provided by-

$$\text{NLL}(\boldsymbol{\theta}) = - \triangleq \frac{1}{N} \log p(\mathcal{D}|\boldsymbol{\theta})$$

Constraints like the need that covariance matrices be positive definite and mixing weights total to one, among others, must frequently be fulfilled, which can be tricky. The expectation maximisation algorithm, or EM for short, is frequently significantly easier (though not always faster) under such circumstances. The procedure is straightforward and iterative, frequently requiring closed-form updates at each stage. Moreover, the algorithm automatically applies the required constraints.The fact that the ML/ MAP estimate would be simple to calculate if all of the data were observed is used by EM. In particular, EM is an iterative algorithm which alternates between inferring the missing values given the parameters (E step), and then optimizing the parameters given the "filled in" data (M step).

Basic idea

Automatically, the latent variables Z_i should help us find the MLEs. In the beginning, we try to compute the posterior distribution of $Z_i$given the observations:

$$P(Z_i = k|X_i) = \frac{P(X_i|Z_i = k)P(Z_i = k)}{P(X_i)} = \frac{\pi_k N(\mu_k, \sigma_k^2)}{\sum_{k=1}^{K} \pi_k N(\mu_k, \sigma_k)} = \gamma_{Z_i}(k)$$

Equation 1

The derivative of the log-likelihood with respect to μ_k in equation (1) can now be written as follows:

$$\sum_{i=1}^{n} \gamma_{Z_i}(k) \frac{(x_i - \mu_k)}{\sigma_k^2} = 0$$

Equation 2

Even though $\gamma_{Zi}(k)$ is dependent on μ_k, we can assume that it is not. We can now find μ_k in this equation by solving for it:

$$\hat{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{z_i}(k) x_i}{\sum_{i=1}^{n} \gamma_{z_i}(k)} = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{z_i}(k) x_i$$

Equation 3

Where we set $N_k = \sum_{i=1}^{n} \gamma_{z_i}(k)$, N_k represents the actual number of points given to component k. We see that $\hat{\mu_k}$ is thus a weighted average of the data with weights$\gamma Zi(k)$.Similarly, if we use a similar approach to find $\hat{\sigma_k^2}$ and $\hat{\pi_k}$ we find that:

$$\hat{\sigma_k^2} = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{z_i}(k)(x_i - \mu_k)^2$$

Equation 4

$$\hat{\pi_k} = \frac{N_k}{n}$$

Equation 5

The two observations mentioned above serve as the motivation for the EM algorithm, which goes like this:

1. Initialize the μk's, σk's and πk's and evaluate the log-likelihood with these parameters.

2. E-step: Evaluate the posterior probabilitiesγZi(k) using the current values of the μk's and σk'swith equation (2)

3. M-step: Estimate new parameters $\hat{\mu_k}$, $\hat{\sigma_k^2}$ and $\hat{\pi_k}$ with the current values ofγZi(k) using following equations (3), (4) and (5).

4. Evaluate the log-likelihood with the new parameter estimates. If the log likelihood has changed by less than some small, stop. Otherwise, go back to step2.

2.3 REFERENCES

- https://stephens999.github.io/fiveMinuteStats/intro_to_em.html
- Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).
- https://en.wikipedia.org/wiki/Probability
- https://www.analyticsvidhya.com/blog/2021/01/discrete-probability-distributions/
- https://en.wikipedia.org/wiki/Inequalities_in_information_theory
- https://theclevermachine.wordpress.com/2012/09/22/monte-carlo-approximations/
- Introducing Monte Carlo Methods with R, Christian P. Robert, George Casella, Springer, 2010.
- Introduction to Machine Learning (Third Edition): EthemAlpaydın, The MIT Press (2015).
- Pattern Recognition and Machine Learning: Christopher M. Bishop, Springer (2006).

2.4 QUESTIONS

- Write a note on Chain rule.
- Explain Unidentifiabilityin Parameter estimation for mixture models
- Describe Graph terminology.
- Write a note on Mixtures of Gaussians.
- Explain how many different commonly used probabilistic models can be simply described as DGMs with an example.
- Describe Naive Bayes classifiers.
- Write a note on Markov and hidden Markov models.
- Write a note on Inference
- Explain learning on Graphical Models.
- Describe Parameter estimation for mixture models
- Write a note on EM algorithm.
- How to compute a MAP estimate is non-convex?
- Explain Plate Notation with an example.

3

KERNELS

Unit Structure

3.0 OBJECTIVE

- To study the importance and benefits of Kernel in machine learning
- To study support vector machine
- To study various methods of kernel

3.1 INTRODUCTION

In machine learning, Kernel is a method that will help us to apply linear classifiers to the non-linear problems by using mapping from non-linear data into a higher-dimensional space.

The study of kernels gives computers the ability to learn without being explicitly programmed. Kernel technique or trick are useful to enter the dataset into a higher dimensional space, and then use the different classification methods for the algorithm.

In the large field of machine learning, we want a machine to learn without having to be explicitly programmed. Regression, classification, and pattern recognition issues are dealt with in ML. We have a variety of techniques for solving classification problems, where the goal is to divided into several classes based on input labels that are known (supervised learning). One is SVM (Support Vector Machine), which employs kernel techniques. To deal with the nonlinearity in the dataset, machine learning uses kernels. The dataset gains a new dimension through the addition of a user-specified kernel function (similarity function), which allows the dataset to be classified using a linear hyperplane.

Need of Kernel

Kernels are very important in machine learning as it is used to transform the data from one dimensional to the other dimensional so that the classification of the data set is very easy and we can get the good performance of the algorithm.

Kernel in Machine Learning is a measuring the similarities between the given two points, it depends on the task as well, for example suppose, for instance, that one's objective is to identify several categories. Using kernel will help to give one set of items in the data a low value and another set of objects with a high value. The most important thing is in this case that kernel offers a quicker method of finding similarities than comparing similarity point by point.

Let takes the examplefor text processing if we use the kernel then kernel will assign the high value to the similar types of the data strings and for non-similar type of data strings will get the low-value.

Kernel function takes data from the original dimension and provides scalar output by using dot products of the vector in a higher dimension. So, the output of a kernel method is a scalar, in this way the higher dimensionality is reduced, and we can easily avoid high dimensional computation to classify categories. This is the magic of the kernel trick.

Let's see a simple an example:

I = (i1, i2, i3);

J = (j1, j2, j3).

Simple function to address nonlinearity: a refers to i,j

f = (a1a1, a1a2, a1a3, a2a1, a2a2, a2a3, a3a1, a3a2, a3a3)

kernel method is K(i, j) = (i.j)^2

we will use some arbitrary data.

i = (1, 2, 3);

j = (4, 5, 6).

Then:
f(i) = (1, 2, 3, 2, 4, 6, 3, 6, 9)

f(j) = (16, 20, 24, 20, 25, 30, 24, 30, 36)

f(i). f(j) = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

A lot of calculation, because f is trying to map from a 3-D to a 9-D space.

Now if we use kernel trick then:

K(i. j) = (4 + 10 + 18) ^2 = 32^2 = 1024

For every classification problem with higher dimensionality and nonlinearity, we cannot use the kernel, without putting any extra effort. It increases flexibility in the model if we use the kernel for complex and higher dimensions. So, idea is to use simple kernels which can reduce computation time and complexity. Because with more flexibility there are chances of overfitting on the training set. Overfitting ruins the model.

It is hard to choose which kernel one should be used for a specific problem. Generally, it is recommended to try all possible kernels in the small-small training set and use the best one.

Let us see another example why do we need a kernel?

Suppose we have two-dimensional dataset that contains two different classes of observations, and we need to find a specific function which will use for separating the two classes. The data is not linearly separable in two-dimensional space.

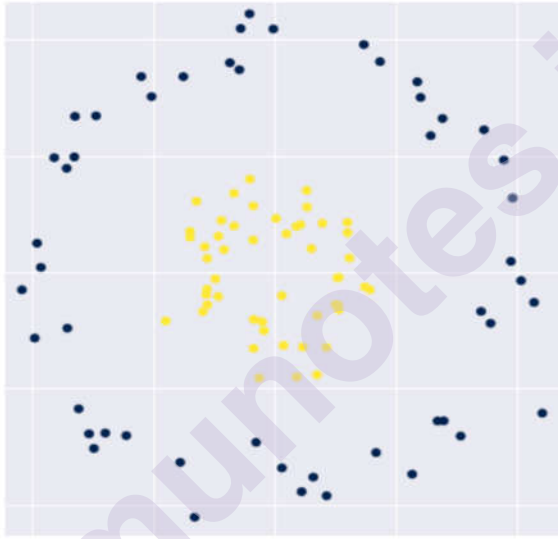


Figure 3.1 2D dataset

[source: programmathically.com]

Now our aim is to fit a polynomial function to separate the data, which complicates our classification problem, if we transform this data into the higher-dimensional (3D) space where the data is separable by a linear classifier?

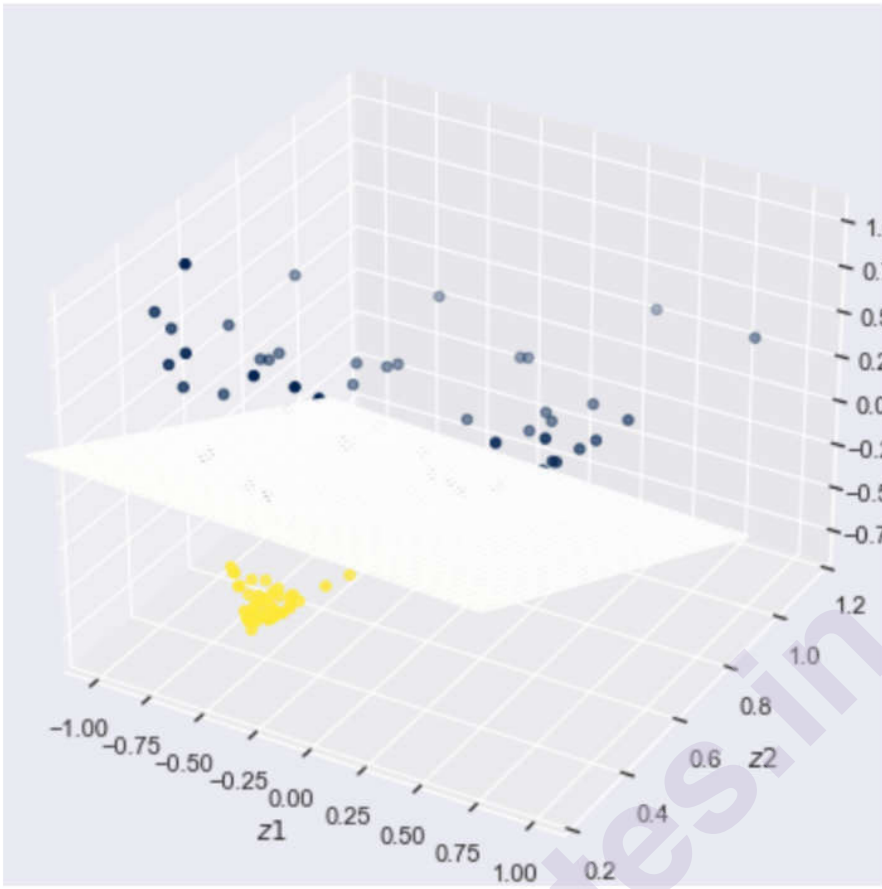


Figure: 3.2 3D Dataset

[source:programmathically.com]

If we find a mapping from 2Dimentional space into the 3Dimentional space where we can find our observations are linearly separable, whereas

- Transform the given data from 2 dimensional into 3 dimensional
- Find the linear decision boundary by fitting a linear classifier (a plane separating the data) in the 3-dimensional space.
- Map the linear decision boundary back into 2-dimensional space. The resultant gives a non-linear decision boundary in 2 dimensional

  We found that a non-linear decision boundary while doing the work of finding a linear classifier.

  Aim is to build a linear classifier; we can transform our input data into 3-dimensional space.

  With the help of Kernels and Kernel trick, we can find a linear decision boundary in a 3-dimensional space while working with input data in the form of 2 dimensional.

  Let us see how we can do this,

  Use of Kernel helps us to separate data with a non-linear decision boundary using a linear classifier.

Working of Kernel

Consider a linear regression model in the following form:

$$y_i = w_0 + w_1 x_i + w_2 x_i + \epsilon_1$$

If we package all weights into a vector w = {w_0,w_1,w_2}, we can express this as a simple dot product between the weight vector and the observation x_i

$$y_i = w^T x_i + \epsilon_1$$

The dot product between xl and the weights gives us the predicted point o the line for the actual observation yi. The difference is the error €_1

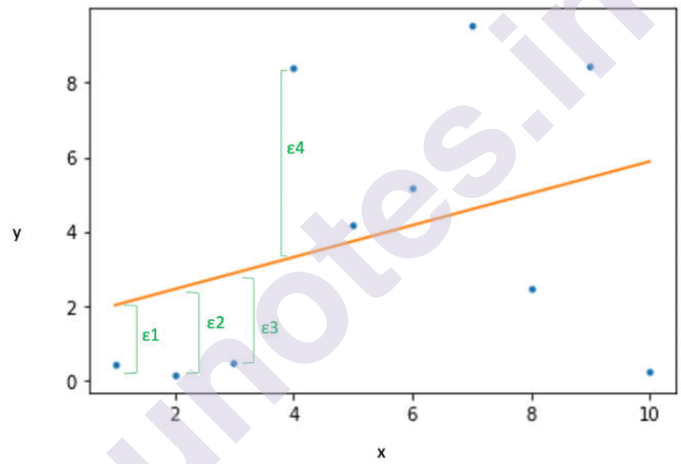


Figure 3.3 Linear classifier

[source:researchgate]

Dot product is central to the prediction operation in a linear classifier.

Role of Kernel

Let's assume we have two vectors, namely x and x^*, in 2-dimensional space, and we want to perform a dot product between them to find a linear classifier. But our data is not linearly separable in our current 2-dimensional vector space.

To solve this problem, we can map the two vectors to a 3-dimensional space.

$$x \rightarrow \phi(x)$$
$$x* \rightarrow \phi(x*)$$

Where φ(x) and φ(x*) are 3 dimensional representations of x and x*

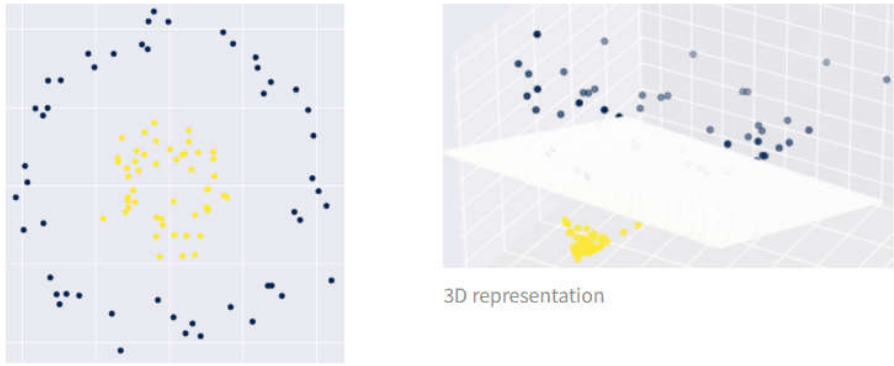


Figure 3.4 2D representation and 3D representation

[programmathically.com]

Now we can perform the dot product between φ(x) and φ(x*) to find the linear classifier into 3-dimensional space and then map back to the 2-Dimensional space.

$$x^T x* \rightarrow \phi(x)^T \phi(x*)$$

But mapping our features explicitly into the higher dimensional space is very expensive. The exact use of kernel is that, representation of higher dimensional mapping without actually performing this type of mapping.

A Kernel is a function of lower-dimensional vectors x, and x* that represents a dot product of φ(x) and φ(x*) in higher-dimensional space.

$$K(x, x*) = \phi(x)^T \phi(x*)$$

To simplify this, we do the square of dot product.

$$k(x, x*) = (x^T x*)^2$$

Recall that x and x* are vectors in a 2-dimensional input space.

$$x = [x_1, x_2]$$
$$x* = [x*_1, x*_2]$$

If we expand the function, we get the following result.

$$(x^T x*)^2 = (x_1 x *_1 + x_2 x *_2) = x_1^2 x *_1^2 + 2 x_1 x *_1 x_2 x *_2 + x_2^2 x *_2^2$$

This result can be neatly decomposed into the product of 2 3-dimensional vectors.

$$\phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix} \quad \phi(x*) = \begin{bmatrix} x*_1^2 \\ \sqrt{2} x *_1 x *_2 \\ x *_2^2 \end{bmatrix}$$

For Kernel, we never have to create the full feature map. Instead, we just insert the original kernel function into out calculation in place of the dot product between x and x^*.

$$k(x, x*) = (x^T x*)^2$$

This is very useful method of kernel; it is the heart of support vector machines.

Benefits

Benefits of using the kernel trick in ML.

- Kernel reduces the complexity of calculation and makes it faster.
- The kernel gives an output that is scalar.
- We can use the kernel to address infinite dimensions.
- Kernel helps in dealing with nonlinear data by introducing linearity.
- Kernel helps to distinguish similar objects easily.

3.2 KERNEL FUNCTION

Kernel function is a method that takes the data as input and transform it into the required form of processing data.

For understanding the kernel function first, we understand the terms like SVM (support vector machines) in those classifications with the supervised learning algorithm for the machine learning.

Let us understand with the help of example

From various types of machines learning the task is to predict the particular breed of a dog with the help of the supervised learning algorithm. Firstly, we have to load all the details of various types of breeds of dogs with the information or the properties like type, skin color, height, body hair length and more details about dog. In machine learning it is

known as features, Single entry of these list of various features are known as a data instance, whereas the collection of everything is the training data set which is used for the basis if your prediction. It means if you know the skin color along with the body hair length as well as height and more details of a particular dog then you can predict the breed it will probably belong to.

Support vector machine are supervised learning models with the associated learning algorithms are generally used. That analyzes the data for classification.

Classifications means knowing that what belong to what for example 'banana' belongs to class 'fruit' whereas 'cat' or 'dog' belongs to class 'animals'

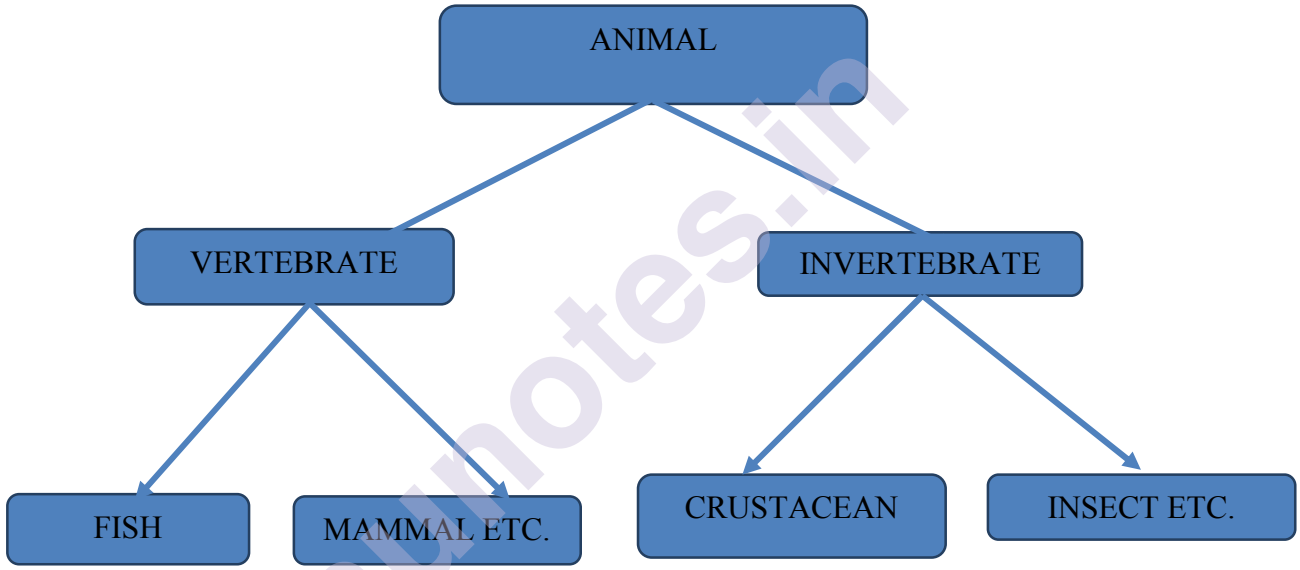


Figure3.5: classification with the help of support vector machine

[source: towardsdatascience.com]

The use of support vector machine is as a classifier formally defined bya separating hyperplane. Hyperplane is a subspace of one dimension less than its ambient place. The term dimension in mathematical space or object is defined as the minimum number of co ordinatesi.e. x, y, z axis which needed to specify any point for example blue color point and the red color point i.e. the mathematical object which has an ambient space.

A mathematical object is an abstract object which arising in mathematics. An abstract object is an object which does not exist at any particular time or place, but rather exists as a type of thing, may be an idea or abstraction etc.

Let us see the hyperplane of a two-dimensional space in the following figure where one dimensional line dividing the red color dots and the blue color dots.

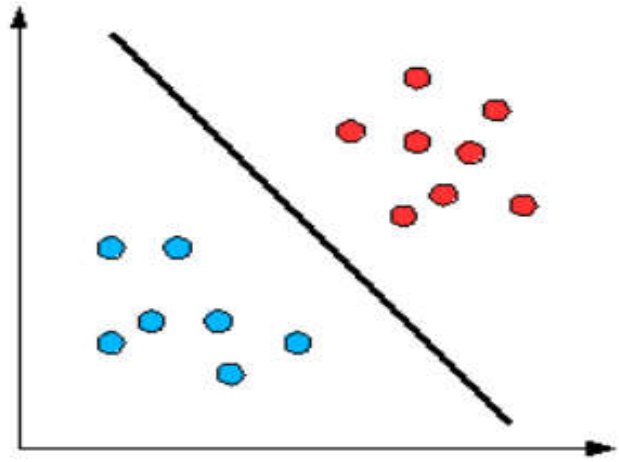


Figure 3.6: SVM for breeds of dog

[source: towardsdatascience.com]

From this figure we are trying to predict the breed of a particular dog, it goes like this

1. Data of all breeds of dog
2. Features like the hair length, skin color and other features
3. Learning algorithm

With the help of following figure, we understand the need of Kernel

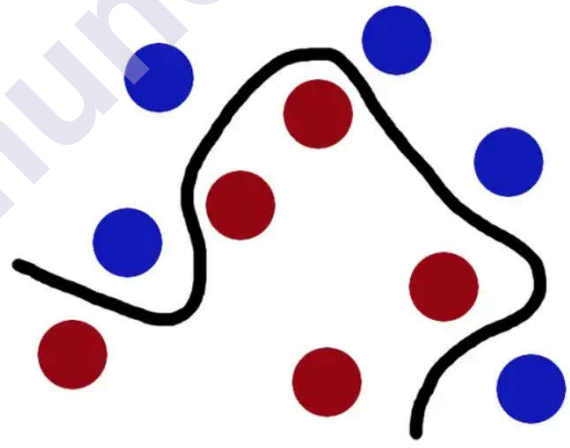


Figure 3.7: Need of Kernel
[source: towardsdatascience.com]

With the help of figure SVM for breed of dog we are not able to solve the problem linearly.

Red color dots and the blue color dots cannot be separated by a straight line as they are randomly distributed. In real world problem as well, the data are randomly distributed.

With the help of machine learning, a kernel is used with the help of kernel trick the various method for linear classifier is used to solve a non-linear problem.

The kernel function is applied on each data instance to map the original non linear observations into a higher-dimensional space in which they become separable.

Example of dog breed prediction, kernel offers a better alternative, defining the various features of a dog instead of that we can define a single kernel function to compute similarity between breeds of dog. Kernel will work with the data and the labels to the learning algorithm, and shows the classifier.

Working:

To understand the Kernel working, we will take the help of Lili Jiang's mathematical model.

It shows:

Mathematical definition: K(x, y) = <f(x), f(y)>. Here K is the kernel function, x, y are n dimensional inputs. f is a map from n-dimension to m-dimension space. <x,y> denotes the dot product. usually, m is much larger than n.

normally calculating <f(x), f(y)> requires us to calculate f(x), f(y) first, and then do the dot product. These two computation steps can be quite expensive as they involve manipulations in m dimensional space, where m can be a large number. But after all the trouble of going to the high dimensional space, the result of the dot product is really a scalar: we come back to one-dimensional space again! Now, the question we have is: do we really need to go through all the trouble to get this one number? do we really have to go to the m-dimensional space? The answer is no, if you find a clever kernel.

Simple Example: x = (x1, x2, x3); y = (y1, y2, y3). Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3), the kernel is K(x, y) = (<x, y>)².

Let's plug in some numbers to make this more intuitive: suppose x = (1, 2, 3); y = (4, 5, 6). Then:

f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)

f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)

<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

A lot of algebra, mainly because f is a mapping from 3-dimensional to 9-dimensional space.

Now let us use the kernel instead:

$K(x, y) = (4 + 10 + 18)\ ^2 = 32^2 = 1024$

With the help of Kernel, we can calculate easily.

3.3 KERNEL TRICK

It transforms the data into an understandable or easily readable form. Because of mapping input space to another feature space, it is possible to transform in understandable form.

In support vector machine (SVM) the inner product is calculated of two vectors and the result of this is always a single number, so when we replace this product of inner by using kernel the it is called as kernel trick.

Use of Kernel tricks for transforming the data set which are in nonlinearly, to reduce the number of calculations of tasks, Kernel tricks are used with the linearity. Kernel always provides a similarity of dataset function which are further helps in data categorization easily with the help of providing scalar output.

Support Vector Machine

support vector machine classifies the observations by constructing a hyperplane that separate out these observations. These observations are lie on the margin surrounding the data separating hyperplane. The margin defines the minimum distance observations should have from the plane, the observations that lie on the margin impact the orientation and position of the hyperplane.

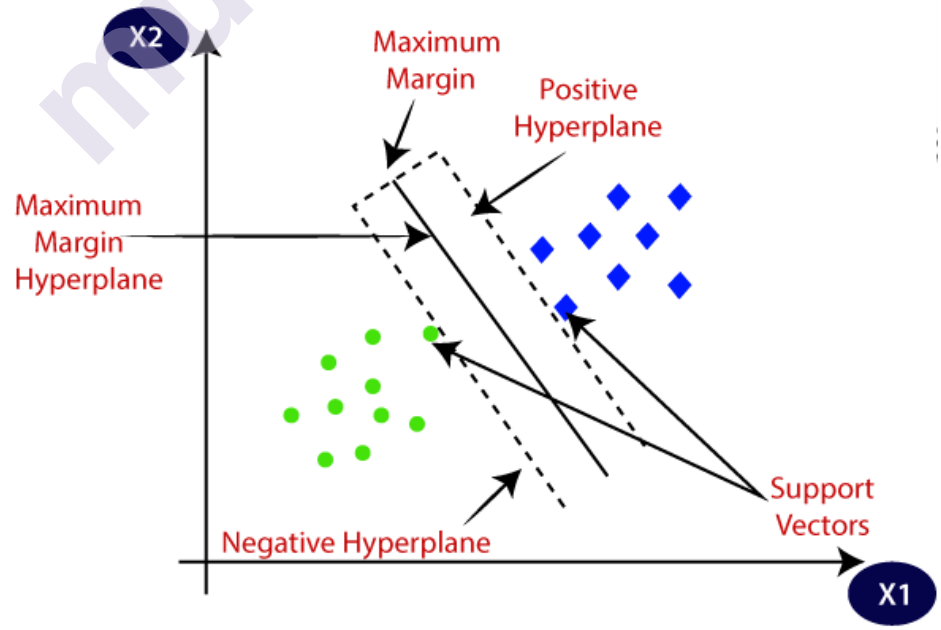


Figure 3.8: Support Vector Machine

[source: abalyticsvidhya.com]

If the data is in linear for then the separation of data points are very easy with the help of hyperplane.

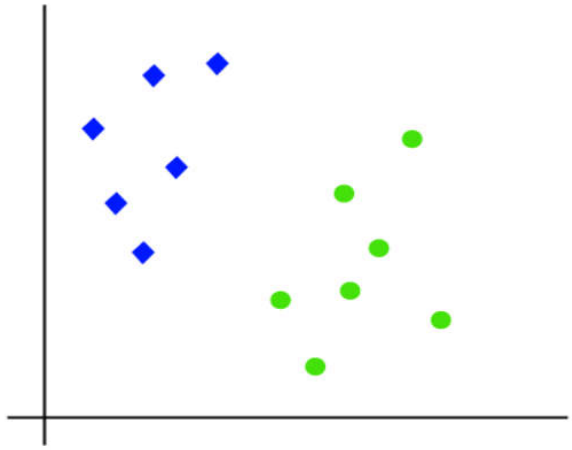


Figure 3.9:Classification of data points

[source: analyticsvidhya.com]

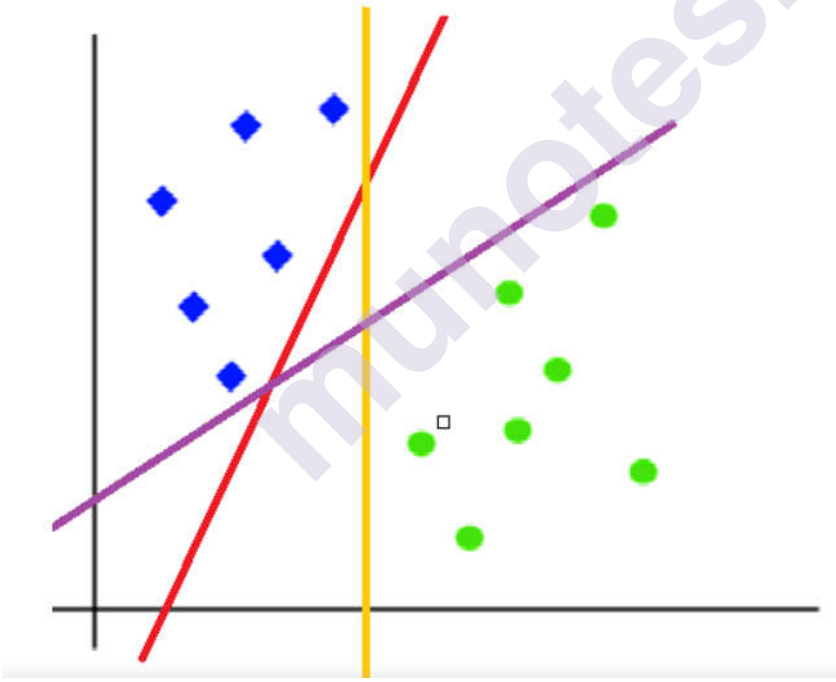


Figure 3.10: Hyperplanes in SVM

[source: analyticsvidhya.com]

In the above example, hyperplanes are perfectly separate the data points or the observations.

We want the classifier to optimize the overall distance between the classifier and the points in addition to separating the training data. This provides us with a specific margin that increases our level of confidence in the forecast. We can be more certain that any subsequent observations that

retain that minimal distance are classified correctly if all the training sites have had a specific minimum distance from the hyperplane.

On the other hand, if some observations are very close to the hyperplane, fresh observations with slightly off-plane characteristics may wind up on the opposite side. As a result, the red hyperplane is preferable than the green ones.

Data points or the observations that are closest to the hyperplane are very important because they lie directly on the margin. They influence the orientation and position of the hyperplane the most and determine how wide the margin is.

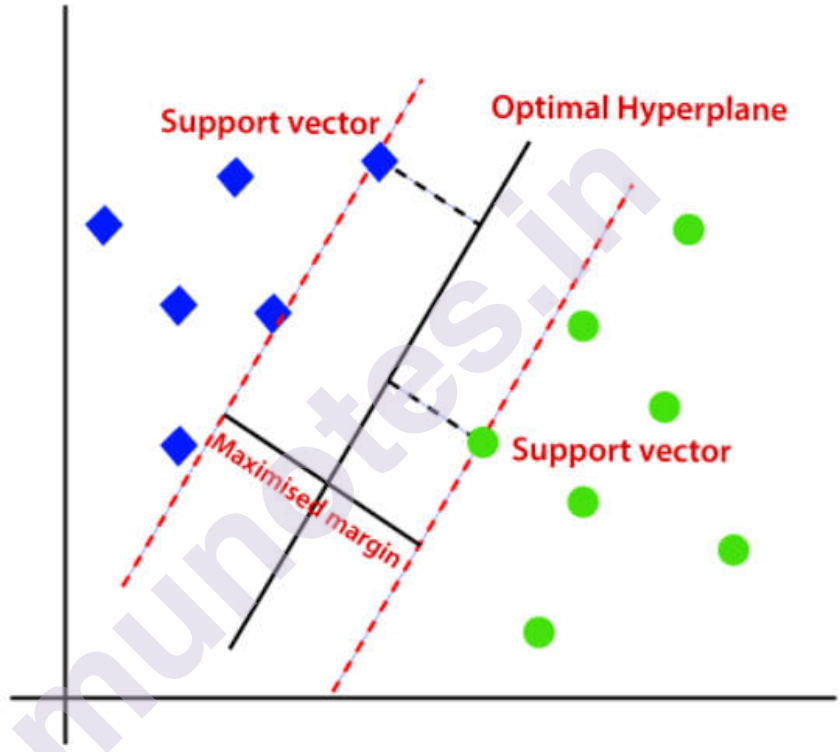


Figure 3.11: Optimal Hyperplane

[source: analyticsvidhya.com]

If we add one more data-points or observation that is closer to the margin, then the hyperplane gives changeable result.

Uses of Support Vector Machine

SVM is useful for classification as well as regression analysis. Generally, it is preferred for classification. The main aim of Support vector Machine is to find a hyper plane from the created a boundary between the various types of data.

In 2-dimensionaldata set or space, the hyper plane is a line but we can plot each item in the data set in N-dimensional data set or space, with number of various features or the attributes of the data set (N).

With the help of support vector machine, we can find the hyperplane it is known as optimal hyperplane. By using hyperplane, we can separate the various data set. We can able to perform by using classification of binary numbers or datasets, for this we have to select any two classes and then for multi-class classification problem we can resolve by using different techniques.

Multiclass classification for Support Vector Machine:

We have to create a binary classifier for each class of the set of data. Then we can get two results of each classifier as follows:

1. Class belonging data points
2. Class does not belong data points

Example
Suppose there is a class of fruits, for multi class classification performance, creation of binary classifier for each fruit. example class as Mango, then there will be a binary classifier we will use for prediction, if it is a mango or it is not a mango. The classifier with the highest score is selected as the output of the SVM.

SVM for complex i.e., for non-linearly separable model works well without any modifications and without any error for linearly separable data set.

With the help of straight line in support vector machine we can plot a graph of linearly separable data, into classes.

Support Vector Machine Kernel
The Support Vector Machine kernel function is that converts non-separable problems into separable problems by taking low-dimensional input space and transforming it into higher-dimensional space. It works best in non-linear separation issues. Simply explained, the kernel determines how to split the data depending on the labels or outputs defined after performing some incredibly sophisticated data transformations.

Advantages of Support Vector Machine:
1. Effective in high dimensional cases

2. Different kernel functions can be specified for the decision functions and possible to specify custom kernels.

3. Its memory efficient as it uses a subset of training points in the decision function called support vectors.

3.5 COMPARISON OF DISCRIMINATIVE KERNEL METHODS

1. Polynomial kernel
   It is mostly used in image processing methods.
   Polynomial kernel is represented as,

$$\mathbf{K}(\mathbf{x_1} \cdot \mathbf{x_\eta}) = (\mathbf{x_1} \cdot \mathbf{x_\eta} + \mathbf{1})^{\mathbf{r}}$$

   Here P represents the degree of the polynomial
2. Gaussian kernel
   There are some applications where prior knowledge is not available. For this type of applications Gaussian kernel is used.
   Gaussian kernel is defined as,

$$K(X, Y) = exp\left(\frac{\|x - y\|^2}{2\sigma^2}\right)$$

3. Gaussian radial basis function (RBF)

   This is also used for the applications where prior knowledge is not available.

   Gaussian radial basis function is defined as,

$$K(x_i, x_j) = \exp(-\gamma \|x - y\|^2) \text{ for } \gamma > 0$$

   Sometimes it is parametrized using the value of y as 1/20"

4. Laplace RBF kernel
   Laplace RBF kernel is defined as,

$$K(x, y) = \exp\left(-\frac{|x - y|}{\sigma}\right)$$

5. Hyperbolic tangent kernel
   It is used in neural networks, and is defined as,

$$K(x_i, x_i) = \tanh(kx_i * x_i + c), \text{ for some (not every) } k > 0 \text{ and } c < 0.$$

6. Sigmoid kernel
   It can be used as a proxy for neural networks, and defined as,

$$K(x, y) = \tanh(a x^d y + c)$$

7. Bassel function of the first kind Kernel
   Cross terms in mathematical functions can be removed by using this type of kernel function, and is defined as,

$$\mathbf{K}(x, y) = \frac{\mathbf{J_{y+1}}(\sigma |x - y|)}{||x - y||^{-n(v+1)}}$$

Here j represented the Bessel function of first type.

8. Anova radial basis kernel

In case of regression problem Anova radial basis kernel can be used, and is defined as,

$$K(X,Y) = \sum_{k=1}^{n} \exp\left(-\sigma(X^{K} - Y^{K})^{2}\right)^{d}$$

3.6 SUMMARY

Kernel in machine learning helps computers to learn without being explicitly programmed. Basically, we use a kernel method or trick to move the input dataset into a higher dimensional space, and then we use any of the available classification algorithms in this higher dimensional area. This is how a hyperplane that divides the two categories linearly is created. In the Figure it shows how this hyperplane can now clearly distinguish between the two groups.

In other words, the kernel in machine learning is a task-dependent measure of similarity between two points. Suppose, for instance, that one's objective is to identify several categories. When using machine learning, the kernel will attempt to give one set of items in the data a low value and another set of objects a high value. The important thing to note in this case is that kernel offers a quicker method of finding similarity than comparing similarity point by point.

3.7 QUESTIONS

1) What is Kernel?
2) Explain Kernel function in detail.
3) What is kernel trick? Explain in detail.
4) Describe Support Vector Machine
5) Explain different Kernel methods.
6) Discuss comparison of discriminative kernel methods.
7) Elaborate the use of support vector machine in machine learning.

3.8 REFERENCES

1. Machine Learning: A Probabilistic Perspective: Kelvin P Murphy, The MIT Press Cambridge (2012).
2. Introduction to Machine Learning (Third Edition) EthemAlpaydin, The MIT Press (2015).
3. https://programmathically.com/what-is-a-support-vector/
4. https://programmathically.com/what-is-a-kernel-in-machine-learning/
5. Introduction to Support Vector Machines (SVM) - GeeksforGeeks
6. What is Kernel in Machine Learning? | why do we need | Benefits | (educba.com)
7. https://towardsdatascience.com/kernel-function-6f1d2be6091
8. Machine Learning step by step guide to implement machine learning algorithm wih python by Rudolph Russell
9. Machine Learning (A Probabilistic Perspective) by Kevin P. Murphy

4

MARKOV AND HIDDEN MARKOV MODELS

Unit Structure :

4.0 OBJECTIVE

1. To study the Markov model in detail.

2. To study the Hidden Markov Models (HMM).

3. To understand various applications of Conditional Random Fields (CRFs).

4.1 MARKOV MODELS

It is a discrete finite system that has N distinct states. The model starts at time t=1 called as initial state.

As per the time step increases the system moves from present state to next state with the transition probabilities that are assigned to present state. Such type of system is known as a discrete, or finite Markov model.

Discrete Markov Model every a_{ij}indicates the probability of transition to state j from state i. The $a_{ij}$are stored in A = $[a_{ij}]$ matrix. P_1 is the probability to begin from a given state i. These start probabilities are indicated by vector p.

4.2 HIDDEN MARKOV MODELS (HMM)

Markov model is an un précised model where it is used in the systems that does not have any fixed patterns of occurrence i.e. randomly changing system. Markov Model is based on the fact of having random probability distribution or pattern that can be analyzed by statistical methods but cannot be predicted precisely.

With the help of Markov models, we can consider that the future states are only depends on the current states and does not depend on the previously occurred states.

There are four common Markov models generally used in the Hidden Markov model.

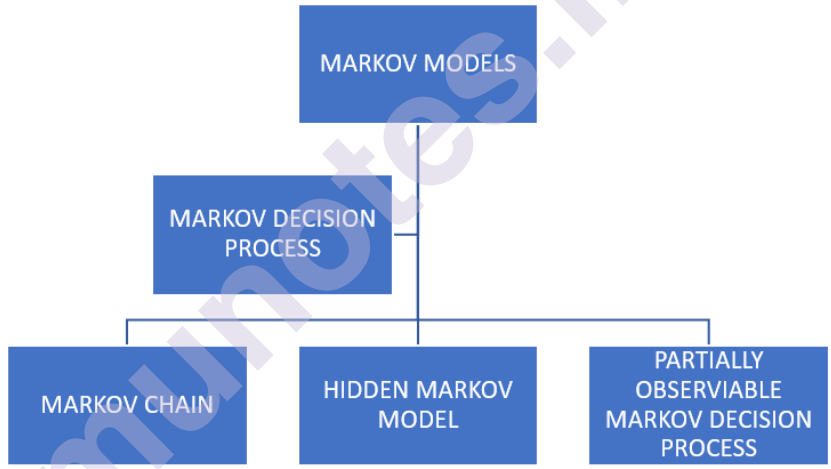


Figure :4.1 Markov Model

Markov Model Property

At t+1 time the state of the system depends only on the state of the system at time t.

$P(X_{t+1} = x_{t+1} \mid X_t = x_t. X_{t-1} = x_{t-1} \ldots\ldots\ldots X_1 = x_1. X_0 = x_0)$

$=P(X_{t+1} = x_{t+1} \mid X_t = x_t)$

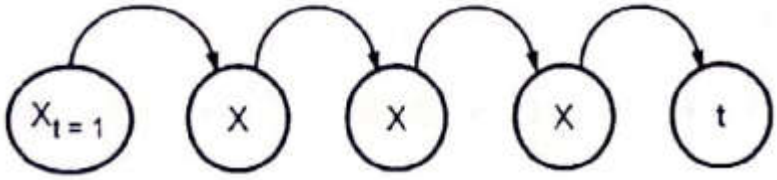


Figure 4.2 Markov chains

Markov Chains: It is probabilities independent of t when process is "stationary"

So, for all t, P ($X_{t+1} = x_{t+1} \mid X_t = x_t$) = P_{ij}

This can be inferred as P_{ij}represents the probability with which the system will be present in the next system without depending on the value of t, if the present system is in state i.

The probability of being in a state j depends only on the previous state, and not on what happened before.

Example

1) Suppose a person has purchased milk, then there is a 90% chance that his next purchase will also be milk.
If the same person purchased bread, then there is an 80% chance that his next purchase will also be bread.
Let us assume that a person currently purchased milk, what is the probability that he will purchase bread two purchase from now and three purchases from now?

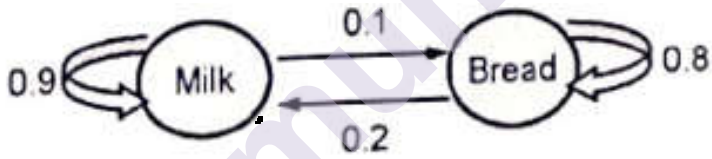


Figure 4.3 Example 1

Solution

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

$A^2 =$

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

$A^2 =$

$$\begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix}$$

The probability of he will purchase bread two purchases from now is 0.17.

$A^3 =$

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix}$$

$=$

$$\begin{bmatrix} 0.781 & 0.219 \\ 0.438 & 0.562 \end{bmatrix}$$

The probability of he will purchase bread three purchases from now is 0.219

Example 2) Consider Markov chain model for 'Rain' and 'Dry' is shown in the following figure Two states: 'Rain' and 'Dry' transition probabilities : P('Rain'|'Rain') = 0.2, P('Dry'|'Rain') = 0.65, P('Rain'|'Dry') = 0.3, P('Dry'|'Dry') = 0.7, Initial probabilities: say P('Rain') = 0.4, P('Dry') = 0.6. Calculate a probability of a sequence of states {'Dry', 'Rain', 'Rain', 'Dry'}.

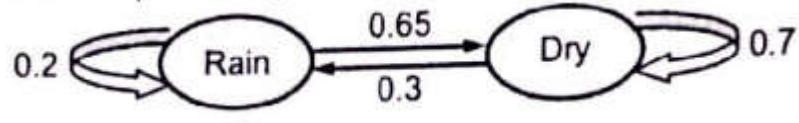


Figure 4,4 Example 2

Solution: P(O | Model) = P (Dry, Rain, Rain, Dry | Model)
= P (Dry) P (Rain | Dry) P (Rain | Rain) P (Dry | Rain)
= 0.6 * 0.3 * 0.2 * 0.65 = 0.0234

4.3 INFERENCE IN HIDDEN MARKOV MODEL

A hidden Markov Model (HMM) is graphical model as shown in the diagram below. The top chain is a Markov chain representing the sate of the system (some). We can notobserved state directly. Whereas we can observe the function of the state which is known as probabilistic function of the state.

For example, the Markov chain can represent the health status of a patient and the observations are symptoms such as temperature, blood pressure etc.

One more example, As the Markov chain can represent the part of speech of the words in a text, and the observation is the actual word.

It is also known as left to right chain model.

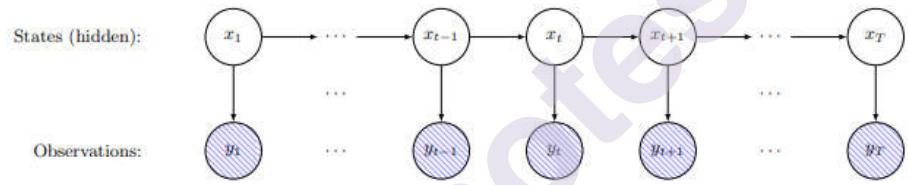


Figure 4.5 Left to right chain model

Probability distribution for the chain model is factorizes as follows:

$$p(xPx_1^T y_1^T;\theta) = P(x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})\prod_{t=1}^{T} P(y_t \mid x_t)$$

$$p(x_1^T, y_1^T;\theta) = p(x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})\prod_{t=1}^{T} p(y_t|x_t).$$

Assume that the Markov chain and the observations are both on discrete spaces, we can complete the model by specifying □ = (π, A,B), where,

- The probability distribution π for x1,

$$\pi_i = p\,(x1 = i).$$

- The transition matrix A of the Markov chain,
$A_{ij} = p\,(x_{t+1} = j \mid x_t = i)$.

- The emission matrix B describing the probabilities of the observations given in the state as,

$B_{ij} = p(y_t = j \mid x_t = i)$

By using three common inference problems associated with Hidden Markov Models and the methods for solving them. We will not derive the solutions but they can be found in one of the above statements

1. Evaluation : $PCx_t|y_1^T;\theta)$→ forward – backward algorithm (sum-product).
2. Decoding : $arg\,max\, x_1^T\, p(x_1^T|y_1^T;\theta)$→ Viterbi algorithm (max-product).
3. Learning: $arg\,max\, \theta P(y_1^T;\theta)$→ Baum – Welch algorithm (EM)

4.4 Learning for Hidden Markov Models

With the help of learning parameters for Hidden Markov models we are able to maximizes the performance of the model.

Learning Hidden Markov Models from data

- Parameter estimation
- If we knew the state sequence then it is very easy to estimate the parameters.
- But when we need to work with hidden state sequences
- Use "expected" counts of state transitions

The approach is maximum likelihood, and we would like to calculate λ∗ that maximizes the likelihood of the sample of training sequences, X={Ok}K k=1, namely, P (X|λ). We start by defining a new variable that will become handy later on. We define ξt(i, j) as the probability of being in Si at time t and in Sj at time t + 1, given the whole observation O and λ:

ξt(i, j) ≡ P (qt = Si (15.25) , qt+1 = Sj |O, λ)

which can be computed as follows (see the following figure)

ξt(i, j) ≡ P (qt = Si, qt+1 = Sj |O, λ)

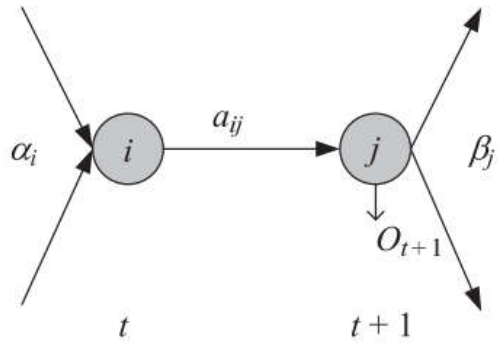


Figure 4.6: Computation of arc probabilities, ξt(i, j)

$\xi t(i, j) \equiv P\ (qt = Si, qt+1 = Sj\ |O, \lambda)$

$$
\begin{aligned}
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda)P(q_t = S_i, q_{t+1} = S_j|\lambda)}{P(O|\lambda)} \\
&= \frac{P(O|q_t = S_i, q_{t+1} = S_j, \lambda)P(q_{t+1} = S_j|q_t = S_i, \lambda)P(q_t = S_i|\lambda)}{P(O|\lambda)} \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t|q_t = S_i, \lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda) \\
&\quad P(O_{t+2} \cdots O_T|q_{t+1} = S_j, \lambda)a_{ij}P(q_t = S_i|\lambda) \\
&= \left(\frac{1}{P(O|\lambda)}\right) P(O_1 \cdots O_t, q_t = S_i|\lambda)P(O_{t+1}|q_{t+1} = S_j, \lambda) \\
&\quad P(O_{t+2} \cdots O_T|q_{t+1} = S_j, \lambda)a_{ij} \\
&= \frac{\alpha_t(i)b_j(O_{t+1})\beta_{t+1}(j)a_{ij}}{\sum_k \sum_l P(q_t = S_k, q_{t+1} = S_l, O|\lambda)} \\
&= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_k \sum_l \alpha_t(k)a_{kl}b_l(O_{t+1})\beta_{t+1}(l)}
\end{aligned}
$$

αt(i) explains the first t observations and ends in state Si at time t. We move on to state Sj with probability aij , generate the (t+1)st observation, and continue from Sj at time t + 1 to generate the rest of the observation sequence. We normalize by dividing for all such possible pairs that can be visited at time t and t + 1. If we want, we can also calculate the probability of being in state Si at time t by marginalizing over the arc probabilities for all possible next states:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j)$$

Note that if the Markov model were not hidden but observable, both γt(i) and ξt(i, j) would be 0/1. In this case when they are not, we estimate them with posterior probabilities that give us soft counts. This is just like the difference between supervised classification and unsupervised clustering where we did and did not know the class labels, respectively. In unsupervised clustering using EM algorithm, it not knowing the class labels, we estimated them first E-step and then calculated the parameters with these estimates in the M-step.

Baum-Welch algorithm: In this algorithm an EM procedure is same, where at each iteration, first the E-step where we compute ξt(i, j) and γt(i) values for the given current equation λ = (A, B, Π), and then in the M-step, we recalculate λ given ξt(i, j) and γt(i).

These are the two steps which will iterate alternate until convergence during which, it has been shown, P (O|λ) never decreases.

Assume indicator variables z_i^t as

$$z_i^t = \begin{cases} 1 & \text{if } q_t = S_i \\ 0 & \text{otherwise} \end{cases}$$

And

$$z_{ij}^t = \begin{cases} 1 & \text{if } q_t = S_i \text{ and } q_{t+1} = S_j \\ 0 & \text{otherwise} \end{cases}$$

These are 0/1 in the case of an observable Markov model and are hidden random variables in the case of an Hidden Markov Model. For later case of E-step as,

$$E[z_i^t] = \gamma_t(i)$$
$$E[z_{ij}^t] = \xi_t(i,j)$$

In the M-step, we calculate the parameters given these estimated values. The expected number of transitions from Si to Sj is $\sum_t \xi_t(i, j)$ and the total number of transitions from Si is $\sum_t \gamma_t(i)$. The ratio of these two gives us the probability of transition from Si to Sj at any time:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

The probability of observing vm in Sj is the expected number of times vm is observed when the system is in Sj over the total number of times the system is in Sj :

$$\hat{b}_j(m) = \frac{\sum_{t=1}^{T} \gamma_t(j) 1(O_t = v_m)}{\sum_{t=1}^{T} \gamma_t(j)}$$

When there are multiple observation sequences

$$\mathcal{X} = \{O^k\}_{k=1}^K$$

Which we assume to be independent

$$P(\mathcal{X}|\lambda) = \prod_{k=1}^{K} P(O^k|\lambda)$$

The parameters are now averages over all observations in all sequences:

$$\hat{a}_{ij} = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\xi_t^k(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\gamma_t^k(i)}$$

$$\hat{b}_j(m) = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k}\gamma_t^k(j)1(O_t^k = v_m)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k}\gamma_t^k(j)}$$

$$\hat{\pi}_i = \frac{\sum_{k=1}^{K}\gamma_1^k(i)}{K}$$

4.5 UNDIRECTED GRAPHICAL MODELS (MARKOV RANDOM FIELDS)

Undirected graphical model or Markov network: It is a set of random variables with the Markov property which is described by an undirected graph.

Or

A random field is said to be a Markov random field which satisfies Markov properties. The original concept comes from the Sherrington-Kirkpatrick model.

It is similar to a Bayesian network where it shows the representation of dependencies.

The difference between Bayesian network and directed and acyclic graph whereas in Markov networks are undirected and may be cyclic.

A Markov network or MRF is similar to a Bayesian Network in its representation of dependencies; the differences being that Bayesian networks are directed and acyclic, whereas Markov networks are undirected and may be cyclic. Thus, a Markov network can represent certain dependencies that a Bayesian network cannot (such as cyclic dependencies; on the other hand, it can't represent certain dependencies that a Bayesian network can such as induced dependencies; The underlying graph of a Markov random field may be finite or infinite.

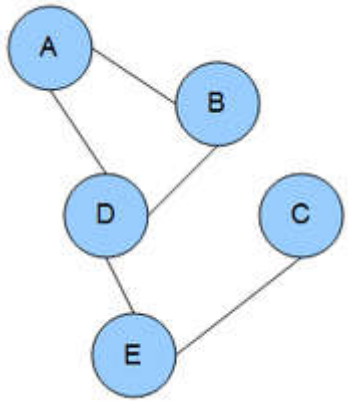


Figure 4.7: An example of a Markov random field. Each edge represents dependency. In this example: A depends on B and D. B depends on A and D. D depends on A, B, and E. E depends on D and C. C depends on E.

When the joint probability density of the random variables is strictly positive, it is also referred to as a Gibbs random field, because, according to the Hammersley Clifford theorem, it can then be represented by a Gibbs measure for an appropriate (locally defined) energy function. The prototypical Markov random field is the Ising model; indeed, the Markov random field was introduced as the general setting for the Ising model. In the domain of artificial intelligence, a Markov random field is used to model various low to medium level tasks related to the image processing and the computer vision.

4.5.1 Conditional Independence properties of UGM

Definition

Given an undirected graph G = (V,E), a set of random variables X = $(x_v)_{v\in V}$ indexed by V form a Markov random field with respect to G if they satisfy the local Markov properties.

Pairwise Markov Property:

Any two non-adjacent variables are conditionally independent given all other variables:

$$X_u \perp\!\!\!\perp X_v \mid X_{V\setminus\{u,v\}}$$

Local Markov Property: A variable is conditionally independent of all other variables given its neighbours:

$$X_v \perp\!\!\!\perp X_{V\setminus \mathrm{N}[v]} \mid X_{\mathrm{N}(v)}$$

Where N(v) is the set of neighbours of v, and N[v] = v U N(v) is the closed neighbourhood of v.

Global Markov Property: Any two subsets of variables are conditionally independent given a separating subset:

$$X_A \perp\!\!\!\perp X_B \mid X_S$$

Where every path from a node in A to a node in B passes through S.

The relation between the three Markov properties is particularly clear in the following formulations:

1. Pairwise
2. Local
3. Global

4.5.2 Parameterization of Markov Random Fields:

To calculate the Markov parameters $\gamma_0, \gamma_1, \gamma_2, \ldots$ from the system

Firstly

$$G(z^{-1}) = \sum_{i=0}^{\infty} y_i z^{-i}$$

For the case when $G(z^{-1})$ is parameterized in the Markov Field form (i.e. $G(z^{-1})$

$= Q^{-1}(z^{-1})R(z^{-1})$), the system Markov parameters can be determined from the equation:

$$(\Sigma_{i=0}^{p} Q_i z^{-i})\,(\Sigma_{i=0}^{\infty} Y_i z^{-i}) = \Sigma_{i=0}^{p} R_i z^{-i},$$

By the following iterative calculations starting from $\gamma_0 = R_0$:

$$Y_k = \begin{cases} R_k - \sum_{i=1}^{k} Q_i Y_{k-i}, & \text{for } k = 1, \ldots, p. \\ -\sum_{i=1}^{p} Q_i Y_{k-i}, & \text{for } k = p+1, \ldots, \infty. \end{cases}$$

If the parameterized in the derivation of the system in Markov parameters is almost the same as:

one starts with $\gamma_0 = B_0$, and continues with the following iterative procedure:

$$Y_k = \begin{cases} B_k - \sum_{i=1}^{\infty} a_i Y_{k-i}, & \text{for } k = 1, \ldots, p. \\ -\sum_{i=1}^{p} a_i Y_{k-i}, & \text{for } k = p+1, \ldots, \infty. \end{cases}$$

4.5.3 Examples of Markov Random Fields

Let's try to understand what these steps correspond to in our chain example. In that case, the chosen ordering was x1, x2,……,xn−1. Starting with x1, we collected all the factors involving x1, which were p(x1) and p(x2 | x1). We then used them to construct a new factor $\tau(x2)=\sum x1 p(x2 \mid x1)$. This can be seen as the results of steps 1 and 2 of the VE algorithm: first we form a large factor $\sigma(x2,x1)=p(x2 \mid x1)p(x1)$; then we eliminate x1 from that factor to produce τ. Then, we repeat the same procedure for x2x2, except that the factors are now p(x3 | x2),τ(x2).

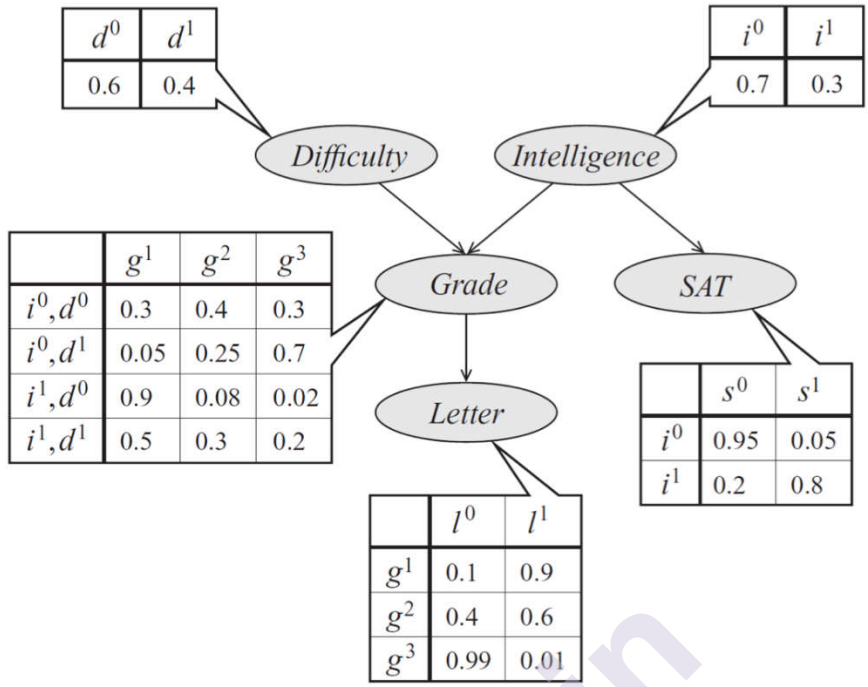


Figure 4.8: Bayes net model of a student's grade gg on an exam; in addition to gg, we also model other aspects of the problem, such as the exam's difficulty dd, the student's intelligence ii, his SAT score ss, and the quality ll of a reference letter from the professor who taught the course. Each variable is binary, except for gg, which takes 3 possible values.

Let us take one example, recall the graphical model of a student's grade that we introduce earlier. The probability specified by the model is of the form

P(l, g, i, d, s) = p(l | g)p(s | i)p(i)p(g | I,d)p(d).

Let us assume that we are computing p(l) and are eliminating variables in their topological ordering in the graph. Firstly we eliminate d, which corresponds to creating a new factor

τ(g,i)=∑dp(g|i,d)p(d). Next is to eliminate i to produce a factor τ2(g,s)=∑iτ1(g,i)p(i)p(s |i). Then the next step is to eliminate s yielding τ3(g)=∑sτ2(g, s) and so on..

These operations are equivalent to summing out the factored probability distribution as follows:

$$p(l) = \sum_g p(l \mid g) \sum_s \sum_i p(s \mid i) p(i) \sum_d p(g \mid i, d) p(d).$$

p(l)=∑gp(l | g) ∑s∑ip(s|i) p(i)∑dp (g|i,d)p(d).p (l)=∑gp(l|g)∑s∑ip (s|i) p(i)∑dp(g|i,d)p(d).

This example calculates at the most k^3 operations per step. Since each factor is at the most over two variable is summed out at each step, dimensionality of k for this example is either 2 or 3.

4.5.4 Conditional Random Field

Conditional random fields (CRFs) are a class of statistical modeling methods often applied in pattern recognition and machine learning and used for structured prediction. Whereas a classifier predicts a label for a single sample without considering "neighbouring" samples, a CRF can take context into account. To do so, the predictions are modelled as a graphical model, which represents the presence of dependencies between the predictions. What kind of graph is used depends on the application. For example, in natural language processing, "linear chain" CRFs are popular, for which each prediction is dependent only on its immediate neighbours. In image processing, the graph typically connects locations to nearby and/or similar locations to enforce that they receive similar predictions.

Other examples where CRFs are used are: labelling or parsing of sequential data for natural language processing or biological sequences, part-of-speech tagging, shallow parsing, named entity recognition, gene finding, peptide critical functional region finding, and object recognition and image segmentation in computer vision.

CRFs are a type of discriminative undirected probabilistic graphical model.

Lafferty, McCallum and Pereira definition of CRF on observations X and random variables Y as follows:

Let G = (V, E) be a graph such that Y = $(Y_v)_{v \in V}$, so that Y is indexed by the vertices of G.

Then (X,Y) is a conditional random field when each random variable Yv, conditioned on X,

With the help of Markov random field property w.r.to the graph (probability is dependent only on its neighbours in G)

G : $P(Yv \mid X, \{Y\omega : \omega \neq v\}) = P(Yv|X, \{Y\omega : \omega \sim v\})$,

Where $\omega \sim v$: ω *and* v *are the neighbours in* G.

It means a CRF is an undirected graphical model whose nodes can be divided into exactly two disjoint sets X and Y, the observed and output variables, respectively; the conditional distribution p(Y|X) is then modelled.

4.5.5 Applications of Hidden Markov model

1. Motion Sensing and Analysis
2. Speech Recognition
3. Language Recognition
4. Gesture Recognition
5. Search Engine Algorithm
6. Marketing applications like Analysis of consumer brand switching (It is based on loyalty of consumer to a particular brand of a product, store or supplier)
7. Weather Forecast (real life application)
8. Finance (Share market stock price movement)

4.6 SUMMARY

Hidden Markov model is an temporal probabilistic model for which a single discontinuous random variable determines all the states of the system. The possible values of variable = Possible states in the system. For example, sunlight can be the variable and sun can be the only possible state. The structure of Hidden Markov model is restricted to the fact that basic algorithms can be implemented using matrix representation.

1. In Hidden Markov Model, every individual states has limited number of transitions and emissions.
2. Probability is assigned for each transition between states.
3. Hence, the past states are totally independent of future states.
4. The fact that Hidden Markov Model is called hidden because of its ability of being a memory less process i.e. its future and past states are not dependent on each other.
5. Since, HMM is rich in mathematical structure it can be implemented for practical applications.
6. This can be achieved on two algorithms called as:

1. Forward Algorithm.
2. Backward Algorithm.

4.7 QUESTIONS:

Q.1) What is Markov Model? Explain in detail.

Q,2) Elaborate in detail Hidden Markov Model.

Q.3) Explain Markov chain.

Q.4) Explain Markov Random Field (MRF) in detail.

Q.4) Give the applications of Hidden Markov Model.

Q.5) What is CRF. Explain in detail.

4.8 REFERENCES

1. Baldi, P., and S. Brunak. 1998. Bioinformatics: The Machine Learning Approach. Cambridge, MA: MIT Press
2. Bilmes, J. A. 2006. “What HMMs Can Do.” IEICE Transactions on Information and Systems E89-D:869–891
3. http://www.ece.virginia.edu/~ffh8x/docs/teaching/esl/12-Inference-in-Hidden-Markov-Models.pdf
4. Markov random field - Wikiwand
5. https://www.sciencedirect.com/topics/computer-science/markov-parameter
6. https://ermongroup.github.io/cs228-notes/inference/ve/

❖❖❖❖

5

MONTE CARLO INFERENCE-SAMPLING

Unit structure :

5.0 OBJECTIVES

After going through this chapter, students will able to learn

- Techniques for randomly sampling a probability distribution.
- To find the expectation of some function f(x) with respect to a probability distribution p(x).
- To enhance decision-making under highly vague conditions.

5.1 INTRODUCTION

In this chapter, we discourse an alternate class of algorithms that are based on the knowledge of Monte Carlo approximation.There are numerous problem areas where estimating or describing the probability distribution as in earlier chapters is relatively straightforward, but calculating a desired quantity is obstinate. This may be due to several reasons, such as the stochastic or random in nature of the domain or an exponential number of random variables.

As an alternative, a desired quantity can be estimated by using random sampling, described as Monte Carlo methods. These methods were originally used around the time that the first computers were created and remain persistent through all fields of science and engineering, including

artificial intelligence and machine learning.Monte Carlo methods are a class of techniques for randomly sampling a probability distribution.

We can attain any desired level of accuracy we need by generating adequate samples,. The main issue is how do we effectively particularly in high dimensions generate samples from a probability distribution? In this chapter, we consider non-iterative methods for generating independent samples. In the next chapter, we discuss an iterative method known as Markov Chain Monte Carlo, or MCMC for short. This method produces dependent samples which works greatly in high dimensions.

5.2 SAMPLING FROM STANDARD DISTRIBUTIONS

In this section, we will discuss in brief some ways to sample from 1 or 2 dimensional distributions of standard form. These methods are often used as subprograms by more complex methods.

The easiest method for sampling from a univariate distribution is based on the inverse probability transform. Let F be a CDF(cumulative Distribution Function) of some distribution we want to sample from.Let F^{-1}be its inverse. Then we have the following result :

Theorem 1: If U ~U(0, 1) is a uniform random variable , then

$F^{-1}(U) \sim F$.

Proof :$Pr(F^{-1}(U) \leq x) = Pr(U \leq F(x))$ (applying F to both sides)

$= F(x)$ (because $Pr(U \leq y) = y$

where the first line follows since F is a monotonic function, and the second line follows since U is uniform on the unit interval.

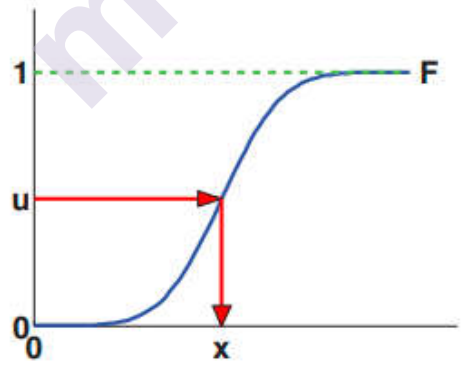


Figure 1 : Sampling using an inverse CDF

Therefore, we can sample from any univariate distribution. In order to find this we can evaluate its inverse cdf, as follows:

- generate a random number u ~U(0, 1) using a pseudo random number generator.
- Let u represent the height up the y axis.

- Then "slide along" the x axis until you intersect the F curve, and then "drop down" and return the corresponding x value.
- This corresponds to computing x = F −1(u). Look into Figure 1

Example : consider the exponential distribution given by :

$E(X) = \lambda e^{-\lambda x}$ ($x \geq 0$, λ is parameter)

The cdf is $F(x) = 1 - e^{-\lambda x}$ ($x \geq 0$)

whose inverse is the quantile function $F^{-1}(p) = -\ln(1 - p)/\lambda$

By the above theorem, if U ∼Unif(0, 1), we know that F −1(U) ∼ E(λ). Besides, since 1 − U ∼Unif(0, 1) as well, we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using − ln(u)/λ.

Next we describe a method to sample from a Gaussian. The aim here is we sample uniformly from a circle of radius one and then use the change of variables formula to derive samples from a spherical 2d Gaussian. This can be thought of as two samples from a 1d Gaussian.Also known as a Box Muller method or transform,it takes a continuous, two dimensional uniform distribution and transforms it to a normal distribution.

It is extensively used in statistical sampling. It is an easy to run, smart way to come up with a standard normal or Gaussian model. Since it can be used to generate normally distributed random numbers, it was originally developed as a better and computationally efficient alternative to inverse sampling. To discuss in detail we do the following :

- First sample $z_1, z_2 \in (-1, 1)$ uniformly.
- Then discard pairs that do not satisfy the condtion : $(z_1)^2 + (z_2)^2 \leq 1$.
- The result will be points uniformly distributed inside the unit circle, so $p(z) = 1/\pi$ (z inside circle).
- Now define for i =1:2, where $r^2 = (z_1)^2 + (z_2)^2$.

$$x_i = z_i \left(\frac{-2 \ln r^2}{r^2} \right)^{\frac{1}{2}}$$

Applying the multivariate change of variables formula, we have

$$p(x_1, x_2) = p(z_1, z_2) \left| \frac{\partial(z_1, z_2)}{\partial(x_1, x_2)} \right| = \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} x_1^2) \right] \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} x_2^2) \right]$$

Hence x_1 and x_2 are two independent samples from a univariate Gaussian. This is known as the Box-Muller method.

To sample from a multivariate Gaussian, we first compute the Cholesky decomposition of its covariance matrix, $\Sigma = LL^T$, where L is lower triangular. Next we sample $x \sim N(0, I)$ using the Box-Muller method. Finally we set $y = Lx + \mu$.

5.3 REJECTION SAMPLING

In last section we discussed inverse cdf method for sampling .When this method cannot be used one alternative that can be used is rejection sampling.

Rejection sampling is a Monte Carlo algorithm to sample data from a difficult to sample from distribution with the help of a proxy distribution.

Rejection sampling is based on the observation that to sample a random variable in one dimension, one can perform a uniformly random sampling of the two-dimensional Cartesian graph, and keep the samples in the region under the graph of its density function.

We will use the following notations here:

- Target (distribution) function f(x) — The "difficult to sample from" distribution. That is our distribution of interest.
- Proposal (distribution) function g(x) — The proxy distribution from which we can sample.

The basic idea that is present in almost all Monte Carlo methods is that if you can not sample from your target distribution function then use another distribution function (and hence called as proposal function).

However, a sampling procedure must "**follow the target distribution**". Following a "target distribution" indicates that we should end up with several samples per their likelihood of occurrence. In simple words, there should be more samples from regions of high probability.

This also means that when we use a proposal function we must introduce the necessary corrections to make sure that our sampling procedure is following the target distribution function! This "corrective" aspect then takes the form of an **acceptance criterion**.

The algorithm for rejection sampling for drawing a sample from the target density f is

1) Simulate $U \sim Unif(0,1)$
2) Simulate a candidate $X \sim g$ from the candidate density
3) If $U \leq f(X)/[cg(X)]$ then "accept" the candidate X.
4) Otherwise, "reject" X and go back to the beginning.

The algorithm can be repeated until the desired number of samples from the target density f has been accepted.

Example : suppose we want to generate samples from a N(0,1) density.

We could use the t2 distribution as our candidate density as it has heavier tails than the Normal. Plotting those two densities, along with a sample from the t2 density gives us the figure below.

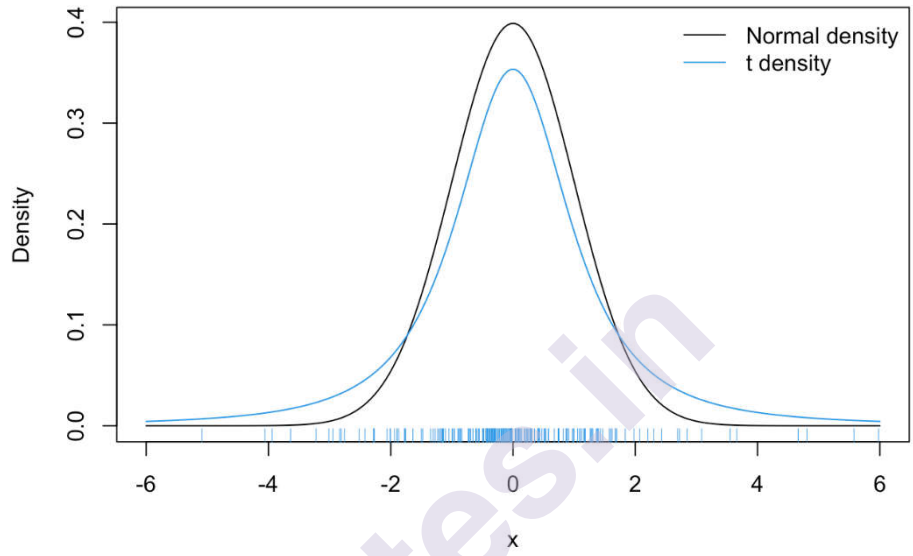


Figure 2 : Normal and t densities

Example source :https://bookdown.org/rdpeng/advstatcomp/rejection-sampling.html

Given what we know about the standard Normal density, most of the samples should be between −3 and +3, except perhaps in very large samples (this is a sample of size 200).

From the figure, there are samples in the range of 4–6. In order to transform the t2 samples into N(0,1) samples, we will have toto reject many of the samples out in the tail. On the other hand, there are too *few* samples in the range of [−2,2] and so we will have to disproportionaly accept samples in that range until it represents the proper N(0,1)density.

Next we describe a method called Adaptive rejection sampling (ARS).It is a method for efficiently sampling from any univariate probability density function which is log-concave. It is very useful in applications of Gibbs sampling, where full-conditional distributions are algebraically very messy yet often log-concave.

The idea is to upper bound the log density with a piecewise linear function, as illustrated in Figure 3(a). We choose the initial locations for the pieces based on a fixed grid over the support of the distribution. We

then evaluate the gradient of the log density at these locations, and make the lines be tangent at these points.

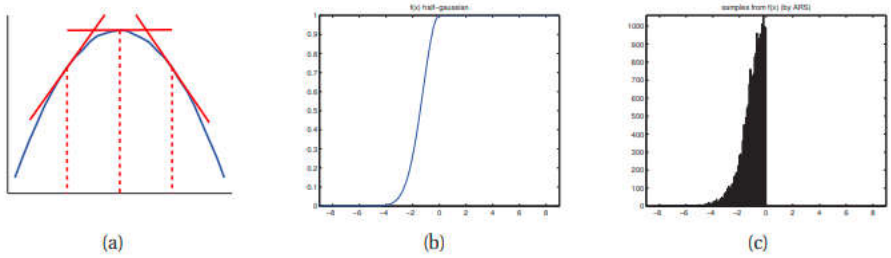


Figure 3 : (a) Idea behind adaptive rejection sampling

(b) and (c) Using ARS to sample from a half-Gaussian

Since the log of the envelope is piecewise linear, the envelope itself is piecewise exponential:

$$q(x) = M_i\lambda_i \exp(-\lambda_i(x - x_{i-1})),\ x_{i-1} < x \leq x_i$$

where x_i are the grid points.

It is reasonably simple to sample from this distribution. If the sample x is rejected, we create a new grid point at x, and thereby refine the envelope. The tightness of the envelope improves,as the number of grid points is increased and the rejection rate goes down. This is known as adaptive rejection sampling (ARS) Figure 3(b and c) gives an example of the method in action. Similar to standard rejection sampling, it can be applied to unnormalized distributions.

It is evident that we want to make our proposal function g(x) as close as possible to the target distribution f(x), while still being an upper bound. But this is quite hard to achieve, especially in high dimensions.We will describe MCMC sampling, which is a more efficient way to sample from high dimensional distributions in next chapter. Sometimes this uses (adaptive) rejection sampling as a subroutine, which is known as adaptive rejection Metropolis sampling.

5.4 IMPORTANCE SAMPLING

Importance sampling is an approximation method instead of sampling method. It derives from a little mathematic transformation and is able to formulate the problem in another way.

We will now describe a Monte Carlo method known as importance sampling for approximating integrals of the form

$$I = \mathbb{E}[f] = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$$

The basic idea is to draw samples x in regions which have high probability, p(x), but also where |f(x)| is large. The result can be super

efficient, means it needs less samples than if we were to sample from the exact distribution p(x).

The reason is that the samples are focussed on the important parts of space. For example, suppose we want to estimate the probability of a rare event. Define f(x) = I(x ∈ E), for some set E. Then it is better to sample from a proposal of the form q(x) ∝ f(x)p(x) than to sample from p(x) itself.

Importance sampling samples from any proposal, q(x). It then uses these samples to estimatethe integral as follows:

$$\mathbb{E}[f] = \int f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x} \approx \frac{1}{S}\sum_{s=1}^{S} w_s f(\mathbf{x}^s) = \hat{I}$$

where w_s are the importance weights. In Contrast to rejection sampling, we use all the samples.

How must we choose the proposal? A natural condition is to minimize the variance of the estimate $\hat{I}$.

When we don't have a particular target function f(x) in mind, we frequently just try to make q(x) as close as possible to p(x). In general, this is difficult, especially in high dimensions, but it is possible to amend the proposal distribution to improve the approximation. This is known as adaptive importance sampling.

Now we will describe a way to use importance sampling to generate samples from a distribution which can be represented as a directed graphical model(DGM) (discussed in later chapter). If we don't have proof, we can sample from the unconditional joint distribution of a DGM p(x) as follows:

- First sample the root nodes, then sample their children, then sample their children, etc. This is known as ancestral sampling.

- It works because, in a DAG, we can always topologically order the nodes so that parents preceed children.

  Now suppose we have some evidence, so some nodes are "clamped" to observed values, and we want to sample from the posterior p(x|D). If all the variables are discrete, we can use the following simple procedure:

- Perform ancestral sampling, but as soon as we sample a value that is inconsistent with an observed value, reject the whole sample and start again. This is known as logic sampling .

- Logic sampling is very inefficient, and it cannot be applied when we have real-valued evidence. But, it can be modified as follows:

- Sample unobserved variables as before, conditional on their parents. But don't sample observed variables; instead we just use their observed values.This procedure is known as likelihood weighting.

It is possible to draw unweighted samples from p(x) as follows :

- First by using importance sampling (with proposal q) to generate a distribution of the form

$$p(\mathbf{x}) \approx \sum_s w_s \delta_{\mathbf{x}^s}(\mathbf{x})$$

Where $w_s$are the normalized importance weights.

- Then sample with replacement from the above equation in first step, where the probability that we choosex_s is w_s.

This processinduce a distribution denoted by $\hat{p}$ .This is known as sampling importance resampling (SIR).

5.5 PARTICLE FILTERING

The idea of the particle filter is based on *Monte Carlo methods*, which utilize particle sets to represent probabilities and can be used in any form of state space model. The core idea is to express its distribution by extracting random state particles from the posterior probability. It is a sequential importance sampling method (Sequential Importance Sampling).

It is an algorithm for recursive Bayesian inference. That is, it approximates the predict-update cycle described in earlier Section of filtering. It is extensivelyapplied in many areas, including tracking, time-series forecasting, online parameter learning, etc.

In simple terms, the particle filtering method refers to the process of obtaining the state minimum variance distribution by finding a set of random samples propagating in the state space to approximate the probability density function and replacing the integral operation with the sample mean.This method also handles nonlinear dynamic models and can tackle nonnormally distributed random instability to the state and measurement.

Sequential importance sampling The basic idea is to appproximate the belief state (of the entire state trajectory) using a weighted set of particles :

$$p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_{1:t}^s}(\mathbf{z}_{1:t})$$

where $\hat{w}_t^s$ is the normalized weight of sample s at time t.

We can easily compute the marginal distribution over the most recent state from this representation,$p(z_t|y_{1:t})$ and by simply disregarding the previous parts of the trajectory, $z_{1:t-1}$.We update this belief state using importance sampling.The basic algorithm is as follows :

- for each old sample s, propose an extension using $z_t^s \sim q(z_t|z_{t-1}^s, y_t)$, and give this new particle weight w_t^s using Equation

$$w_t^s \propto w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)}$$

- This basic algorithm unfortunatelydoes not work very well, as we discuss further down.

The basic sequential importance sampling algorithm stop working after a few steps because most of the particles will have negligible weight. This is called the degeneracy problem, and occurs because we are sampling in a high-dimensional space (in fact, the space is growing in size over time), using a biased proposal distribution.There are two principal solutions to the degeneracy problem:

1. Adding a resampling step, and
2. Applying a good proposal distribution.

1. The resampling step :In the resampling step, the particles with negligible weights are replaced by new particles in the vicinity of the particles with higher weights. From the statistical and probabilistic point of view, particle filters may be interpreted as mean-field particle interpretations of Feynman-Kac probability measures.There are a variety of algorithms for peforming the resampling step. The simplest is multinomial resampling, which computes
(K1,...,KS) ~ Mu(S,(w_t^1,...,w_t^s))

We then make K_s copies of z_t^s . Various enhancements exist, such as systematic resampling residual resampling, and stratified sampling, which can reduce the variance of the weights.

Although the resampling step helps with the degeneracy problem, it introduces difficulties of its own. In particular, since the particles with high weight will be selected many times, there is a loss of diversity amongst the population. This is known as sample impoverishment.

In extreme case of no process noise (for example, if we have static but unknown parameters as part of the state space), then all the particles will collapse to a single point within a few iterations. To moderate this problem, several solutions have been proposed.

(1) Only resample when necessary, not at every time step. (The original bootstrap filter (Gordon 1993) resampled at every step, but this is suboptimal.)

(2) After replicating old particles, sample new values using an MCMC step which leaves the posterior distribution invariant.

(3) Create a kernel density estimate on top of the particles, We then sample from this smoothed distribution. This is known as a regularized particle filter .

(4) When performing inference on static parameters, add some artificial process noise.

2. The proposal distribution: The simplest and most widely used proposal distribution is to sample from the prior:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = p(\mathbf{z}_t|\mathbf{z}_{t-1}^s)$$

In this case, the weight update simplifies to

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{z}_t^s)$$

we sample values from the dynamic model, and then evaluate how good they are after we see the data. This is the approach used in the condensation algorithm (which stands for "conditional density propagation") used for visual tracking.However, if the likelihood is narrower than the dynamical prior that is the sensor is more informative than the motion model, (which is often the case), this is a very inefficient approach, since most particles will be assigned very low weight.

5.6 APPLICATIONS

In this section we will see some applications of particle filtering.

1. Robot localization : Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental capabilities required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions.For example,consider a mobile robot wandering around an office environment. We will assume that it already has a map of the world, represented in the form of an occupancy grid, which just specifies whether each grid cell is empty space or occupied by an something solid like a wall. The goal is for the robot to estimate its location. This can be solved optimally using an HMM filter, since we are assuming the state space is discrete. However, since the number of states, K, is often very large, the $O(K^2)$ time complexity per update is prohibitive. We can use a particle filter as a sparse approximation to the belief state. This is known as Monte Carlo localization.

2. Visual object tracking: It is one of the principal challenges in Computer Vision, where the task is to locate a certain object in all frames of a video, given only its location in the first frame.It is concerned with

tracking an object (for example a remote-controlled helicopter) in a video sequence. The method uses a simple linear motion model for the centroid of the object, and a color histogram for the likelihood model, using Bhattacharya distance to compare histograms. The proposal distribution is obtained by sampling from the likelihood.

3. Time series forecasting : InearlierSection , we discussed how to use the Kalman filter to perform time series forecasting. This assumes that the model is a linear-Gaussian state-space model. There are many models which are either non-linear and/or non-Gaussian. For example, stochastic volatility models, which are widely used in finance, assume that the variance of the system and/or observation noise changes over time. Particle filtering is widely used in such settings.

5.7 RAO-BLACKWELLISED PARTICLE FILTERING (RBPF)

RBPFs are an extension to particle filters (PFs) which are applicable to conditionally linear-Gaussian state-space models.

In some models, we can partition the hidden variables into two kinds, q_t and z_t, such that we can analytically integrate out z_t provided we know the values of $q_{1:t}$. This means we only have sample $q_{1:t}$, and can represent $p(z_t|q_{1:t})$ parametrically. Thus each particle s represents a value for $q_{1:t}^s$ and a distribution of the form $p(z_t|y_{1:t}, q_{1:t}^s)$. These hybrid particles are are sometimes called distributional particles or collapsed particles.The advantage of this approach is that we reduce the dimensionality of the space in which we are sampling, which reduces the variance of our estimate. Hence this technique is known as Rao-Blackwellised particle filtering or RBPF for short.

Some of the applications of RBPF are :

- **Tracking a maneuvering target:** It is to track moving objects that have piecewise linear dynamics. For example, suppose we want to track an airplane or missile; q_t can specify if the object is flying normally or is taking evasive action. This is called maneuvering target tracking.

- **Fast SLAM:** It is an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations, yet scales logarithmically with the number of landmarks in the map.in earlier section we introduced the problem of simultaneous localization and mapping or SLAM for mobile robotics. The main problem with the Kalman filter implementation is that it is cubic in the number of landmarks.An additional advantage of this techniques is ,it is easy to use sampling to handle the data association ambiguity, and that it allows for other representations of the map, such as occupancy grids. This idea was first suggested in (Murphy 2000), and was subsequently extended and made practical in (Thrun et al. 2004), who christened the technique FastSLAM.

5.8 SUMMARY

In this chapter we consider approximate inference methods based on numerical sampling, also known as Monte Carlo techniques. Although for some applications the posterior distribution over unobserved variables will be of direct interest in itself, for most situations the posterior distribution is required primarily for the purpose of evaluating expectations, for example in order to make predictions. The fundamental problem that we therefore wish to address in this chapter involves finding the expectation of some function f(z) with respect to a probability distribution p(z).we considered some simple strategies for generating random samples from a given distribution.

5.9 EXERCISE

1 Show how to use inverse probability transform to sample from a standard Cauchy, T (x|0, 1, 1).

2 Explain sampling from standard distributions.

3 Explain various types of sampling.

4 What are applications of sampling.

5 Write a note on Partilcle filtering.

5.10 REFERENCES

- Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).
- Introducing Monte Carlo Methods with R, Christian P. Robert, George Casella, Springer, 2010
- Introduction to Machine Learning (Third Edition): EthemAlpaydın, The MIT Press (2015).
- Pattern Recognition and Machine Learning: Christopher M. Bishop, Springer (2006)

6

MARKOV CHAIN MONTE CARLO (MCMC) INFERENCE

Unit structure :

6.0 OBJECTIVES

After going through this chapter, students will able to learn

- To carry out sampling from high-dimensional distributions is Markov chain Monte Carlo or MCMC.
- Learn about some algorithms of MCMC
- Speed and accuracy of MCMC.

6.1 INTRODUCTION

In Chapter 5, we introduced some simple Monte Carlo methods, including rejection sampling and importance sampling. The difficulty with these methods is that they do not work well in high dimensional spaces. The most popular method for sampling from high-dimensional distributions is Markov chain Monte Carlo or MCMC. These methods encompass a class of algorithms for sampling from a probability distribution. By creating a Markov chain that has the anticipated distribution as its equilibrium distribution, one can acquire a sample of the desired distribution by recording states from the chain. The more steps that are included, the more closely the distribution of the sample matches the actual desired distribution. Various algorithms exist for constructing chains, including the Metropolis–Hastings algorithm. MCMC allow for parameter estimation such as means, variances, expected values, and exploration of the posterior distribution of Bayesian models. To review the properties of a "posterior", many representative random values should be sampled from that distribution.

The MCMC algorithm has an exciting history. It was discovered by physicists working on the atomic bomb at Los Alamos during World War II, and was first published in the open literature in a chemistry journal. An extension was published in the statistics literature in (Hastings 1970), but was largely unnoticed. A special case of Gibbs sampling was independently invented in 1984 in the context of Ising models and was published in Geman and Geman 1984. But it was not until Gelfand and Smith 1990 that the algorithm became well-known to the wider statistical community. Since then it has become wildly popular in Bayesian statistics, and is becoming increasingly popular in machine learning.

The basic idea behind MCMC is to construct a Markov chain on the state space X whose stationary distribution is the target density $p^*(x)$ of interest (this may be a prior or a posterior). That is, we perform a random walk on the state space, in such a way that the fraction of time we spend in each state x is proportional to $p^*(x)$. By drawing correlated samples $x_0, x_1, x_2,...$, from the chain, we can perform Monte Carlo integration with respect to p^*.

If we try comparing MCMC to variational inference, then it is as follows: The advantages of variational inference are

(1) for small to medium problems, it is usually faster;

(2) it is deterministic;

(3) is it easy to determine when to stop;

(4) it often provides a lower bound on the log likelihood.

Whereas ,the advantages of sampling are:

(1) it is often easier to implement;

(2) it is applicable to a broader range of models, such as models whose size or structure changes depending on the values of certain variables (e.g., as happens in matching problems), or models without nice conjugate priors;

(3) sampling can be faster than variational methods when applied to really huge models or datasets.

6.2 GIBBS SAMPLING

In this section, we exhibit one of the most popular MCMC algorithms, known as Gibbs sampling. In physics, this method is known as Glauber dynamics or the heat bath method. This is the MCMC analog of coordinate descent.

Like other MCMC methods, the Gibbs sampler constructs a Markov Chain whose values converge towards a target distribution. Gibbs Sampling is in fact a specific case of the Metropolis-Hastings algorithm wherein proposals are always accepted.

For instance suppose we wanted to sample a multivariate probability distribution(A **multivariate** probability distribution is a function of multiple variables (i.e. 2 dimensional normal distribution)). We don't know how to sample from the latter directly. Though, because of some mathematical convenience, we happen to know the conditional probabilities. This is where Gibbs sampling comes in. Gibbs Sampling is appropriate when the joint distribution is not known explicitly or is difficult to sample from directly, but the conditional distribution of each variable is known and is easier to sample from.

Gibbs sampling is commonly used as a means of statistical inference, especially Bayesian inference. It is a randomized algorithm (i.e. an algorithm that makes use of random numbers), and is an alternative to deterministic algorithms for statistical inference such as the expectation-maximization algorithm (EM).

The main idea of Gibbs sampling is that given a multivariate distribution, it's simpler to sample from a conditional distribution than from a joint distribution. For instance, instead of sampling directly from a joint distribution P(x,y),Gibbs sampling propose sampling from two conditional distribution P(x|y) and P(y|x).
For a joint distribution P(x,y) , we start with a random sample ($x^{(0)}$, $y^{(0)}$). Then we sample $x^{(1)}$ from the conditional distribution $P(x|x^{(0)})$ and $y^{(1)}$from the conditional distribution $P(y|y^{(0)})$.Analogously,we sample $x^{(k)}$ from the conditional distribution $P(x|x^{(k1)})$ and $y^{(k)}$from the conditional distribution $P(y|y^{(k-1)})$
Consider the distribution $p(z) = p(z_1,...,z_M)$ from which we wish to sample, and suppose that we have chosen some initial state for the Markov chain. Each step of the Gibbs sampling procedure involves replacing the value of one of the variables by a value drawn from the distribution of that variable conditioned on the values of the remaining variables. Thus we replace z_i by a value drawn from the distribution p(zi|z\i), where z_i denotes the ith component of z, and z\i denotes $z_1,...,z_M$ but with z_i omitted. This procedure is repeated either by cycling through the variable in some particular order or by choosing the variable to be updated at each step at random from some distribution.For example, if we have D = 3 variables, we use

- $x_1^{s+1} \sim p(x_1|x_2^s, x_3^s)$
- $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_3^s)$
- $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1})$

This readily generalizes to D variables. If x_i is a visible variable, we do not sample it, since its value is already known.

The expression $p(x_i|x_{-i})$ is called the **full conditional** for variable i.

In general, x_i may only depend on some of the other variables. If we represent p(x) as a graphical model, we can infer the dependencies by looking at i's Markov blanket, which are its neighbors in the graph. Thus to sample x_i, we only need to know the values of i's neighbors.
In this sense, Gibbs samplingis a distributed algorithm. However, it is not a parallel algorithm, since the samples must be generated sequentially.For reasons that will be explain in later sections , it is necessary to discard some of the initial samples until the Markov chain has **burned in**, or entered its stationary distribution. We will discuss how to estimate when burnin has occured next section . In the examples below, we just discard the initial 25% of the samples, for simplicity.

To better explain Gibbs Sampling , let us consider a simple example. Suppose we assume that we have two events A and B. Assume that the joint probability that the event will happen is given as:

- P(A and B) = 0.2 (both events will happen)
- P(A and not B) = 0.4 (only event A will happen)
- P(not A and B) = 0.3 (only event B will happen)
- P(not A and not B) = 0.1 (neither A nor B will happen)

Suppose now we want to sample fro the joint distribution above. We need to generate a sequence of pairs (x,y) where $x,y \in \{0,1\}$. For example, (1,0) indicates that only event A has happened.

Obviously there is a simple way of sampling from the joint distribution above by taking a unit interval and dividing it into four parts, where the area of each part is equal to the probability above.

For example, we can generate a random number n between 0 and n. If n is between 0 and 0.2 , then our sample is (1,1), if n is between 0.2 and 0.6, our sample is (1,0) and analogously for others. For this simple example, the method with unit interval is more convenient than Gibbs sampling but for some more complicated examples, especially with continuous distributions, we can't use this method

Gibbs sampling can be applied to the following examples :

1. **Isingmodel :**

In many cases where we are interested in doing inference, we can't do it exactly. With such cases, we can approximate the true distribution using samples from it. Let's look at a model where we need to use techniques like the Ising model .
The Ising model isn't the only one where sampling techniques like the ones we'll discuss are useful, and these techniques aren't the only way to do approximate inference here, but they provide a convenient story for illustrating both ideas.

Gibbs sampling in pairwise MRF/CRF takes the form:

$$p(x_t|\mathbf{x}_{-t}, \boldsymbol{\theta}) \propto \prod_{s \in \text{nbr}(t)} \psi_{st}(x_s, x_t)$$

In the case of an Ising model with edge potentials $\psi(x_s, x_t) = \exp(Jx_sx_t)$, where $x_t \in \{-1, +1\}$

2. **For inferring the parameters of a GMM :**

   It is straightforward to derive a Gibbs sampling algorithm to "fit" a mixture model, specifically if we use conjugate priors. We will emphasis on the case of mixture of Gaussians, although the results are easily extended to other kinds of mixture models.

   Suppose we use a semi-conjugate prior. Then the full joint distribution is given by:

$$\begin{aligned} p(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\mathbf{z}|\boldsymbol{\pi}) p(\boldsymbol{\pi}) \prod_{k=1}^{K} p(\boldsymbol{\mu}_k) p(\boldsymbol{\Sigma}_k) \\ &= \left(\prod_{i=1}^{N} \prod_{k=1}^{K} (\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times \\ &\quad \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_0, \mathbf{V}_0) \text{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_0, \nu_0) \end{aligned}$$

   The same prior is used for each mixture component.

   Even though it is easy to implement, Gibbs sampling for mixture models has a major weakness. The problem is that the parameters of the model θ, and the indicator functions z, are unidentifiable, since we can arbitrarily permute the hidden labels without affecting the likelihood. Accordingly, we cannot just take a Monte Carlo average of the samples to compute posterior means, since what one sample considers the parameters for cluster 1 may be what another sample considers the parameters for cluster 2. Indeed, if we could average over all modes, we would find E [μk|D] is the same for all k (assuming a symmetric prior). This is called the **label switching problem**.

   In certain cases, we can analytically integrate out some of the unknown quantities, and just sample the rest. This is called a **collapsed Gibbs sampler**, and it tends to be much more efficient, since it is sampling in a lower dimensional space.

   Suppose we sample z and integrate out θ. Thus the θ parameters do not participate in the Markov chain; therefore we can draw conditionally independent samples which will have much lower variance than samples drawn from the joint state space .This process is called **Rao-Blackwellisation.**This process can reduce statistical variance, it is only worth doing if the integrating out can be done quickly, otherwise we

will not be able to produce as many samples per second as the naive method.

Gibbs sampling for hierarchical GLMs: Frequently we have data from multiple related sources. If some sources are more reliable and/or data-rich than others, it makes sense to model all the data simultaneously, so as to enable the borrowing of statistical strength. One of the most natural way to solve such problems is to use hierarchical Bayesian modeling, also called multi-level modeling. In earlier section wedeliberated a way to perform approximate inference in such models using variational methods. It can also be done using Gibbs sampling.

Example : Suppose we have data on studentsin diferent schools. Such data is naturally modeled in a two-level hierarchy:

- we let y_{ij} be the response variable we want to predict for student i in school j.
- This prediction can be based on school and student specific covariates, x_{ij} .

Since the quality of schools varies, we want to use a separate parameter for each school. So our model becomes yij = $x_{ij}^T w_j$ + □$_{ij}$

We might fit each w_j separately, but this can give poor results if the sample size of a given school is small. We can get better results if we construct a hierarchical Bayesian model, in which the w_j are assumed to come from a common prior: $w_j \sim N(\mu_w, \Sigma_w)$

In this model, the schools with small sample size borrow statistical strength from the schools with larger sample size, because the wj 's are correlated via the latent common parents (μ_w, Σ_w).

Gibbs sampling is so popular since it is possible to design general purpose software that will work for almost any model which is one of the reason. This software just needs a model specification, usually in the form a directed graphical model (specified in a file, or created with a graphical user interface), and a library of methods for sampling from diferent kinds of full conditionals. BUGS is an example of an packagewhich stands for "Bayesian updating using Gibbs Sampling". BUGS is very broadly used in biostatistics and social science. Another more recent, but very similar, package is JAGS which stands for "Just Another Gibbs Sampler". This uses a similar model specification language to BUGS.

The Imputation Posterior or IP algorithm is a special case of Gibbs sampling in which we group the variables into two classes:

- hidden variables z and
- parameters θ.

It is basically an MCMC version of EM, where the E step gets replaced by the I step, and the M step gets replaced the P step. This is an example of a more general strategy called data augmentation, whereby we introduce auxiliary variables in order to simplify the posterior computations (here the computation of $p(\theta|D)$).

Gibbs sampling can be quite slow, since it only updates one variable at a time (so-called single site updating). If the variables are highly correlated, it will take a long time to move away from the current state. If the variables are highly correlated, the algorithm will move very slowly through the state space. In particular, the size of the moves is controlled by the variance of the conditional distributions. If this is ℓ in the x_1 direction, and the support of the distribution is L along this dimension, then we need $O((L/\ell)^2)$ steps to obtain an independent sample. In some cases we can efficiently sample groups of variables at a time. This is called blocking Gibbs sampling or blocked Gibbs sampling.

6.3 METROPLOIS HASTINGS ALGORITHM

Even though Gibbs sampling is simple, it has some drawbacks :

- It is somewhat restricted in the set of models to which it can be applied. For example, it is not much help in computing $p(w|D)$ for a logistic regression model, since the corresponding graphical model has no useful Markov structure.
- Gibbs sampling can be quite slow.

Fortunately, there is a more general algorithm that can be used, known as the Metropolis Hastings or MH algorithm.

The Metropolis-Hastings algorithm is one of the most popular Markov Chain Monte Carlo (MCMC) algorithms. Like other MCMC methods, the Metropolis-Hastings algorithm is used to generate serially correlated draws from a sequence of probability distributions. The sequence converges to a given target distribution.

The Metropolis-Hastings algorithm requires only two things:
1. The ability to compute unnormalized probabilities of samples $p_X(x)$: here, unnormalized is okay because we'll only be interested in ratios
2. A proposal distribution $V(x'|x)$, which tells us how to generate the next sample x' given the current sample x.

The elementary idea in MH is that at each step, we propose to move from the current state x to a new state x' with probability $q(x'|x)$, where q is called the proposal distribution (also called the kernel). The user is free to use any kind of proposal they want, subject to some conditions which we explain below. This makes MH quite a flexible method. A commonly used proposal is a symmetric Gaussian distribution centered on the current state, $q(x'|x) = N(x'|x, \Sigma)$; this is called a random walk Metropolis

algorithm. If we use a proposal of the form $q(x' |x) = q(x')$, where the new state is independent of the old state, we get a method known as the independence sampler, which is similar to importance sampling. Having proposed a move to x' , we then decide whether to accept this proposal or not according to some formula, which ensures that the fraction of time spent in each state is proportional to $p^*(x)$. If the proposal is accepted, the new state is x' , otherwise the new stateis the same as the current state, x (i.e., we repeat the sample). If the proposal is symmetric, so $q(x' |x) = q(x|x')$, the acceptance probability is given by the following formula: $r = \min(1, p^*(x')/ p^*(x)$.

We see that if x' is more probable than x, we definitely move there (since $p^*(x')/ p^*(x) > 1$), but if x' is less probable, we may still move there anyway, depending on the relative probabilities. So instead of greedily moving to only more probable states, we occasionally allow "downhill" moves to less probable states.

If the proposal is asymmetric, so $q(x' |x) \neq q(x|x')$, we need the Hastings correction, given by the following:

$$r = \min(1, \alpha)$$
$$\alpha = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})}{p^*(\mathbf{x})/q(\mathbf{x}|\mathbf{x}')}$$

This correction is needed to compensate for the fact that the proposal distribution itself (rather than just the target distribution) might favor certain states.

It turns out that Gibbs sampling, is a special case of MH. In particular, it is equivalent to using MH with a sequence of proposals of the form $q(x' |x) = p(x_i' |x_{-i})I(x_{-i}' = x_{-i})$. That is, we move to a new state where x_i is sampled from its full conditional, but x_{-i} is left unchanged.

Some important points to note for MH:

(a) Choosing a good proposal distribution can be tricky, and is usually problem-dependent.

(b) Notice that we initialized completely arbitrarily: in many cases, this initialization could be in a particularly low-probability location. A common practice is to wait K iterations before collecting any samples, to avoid any artifacts from initialization. In this case, K is known as the burn-in time.

(c) Another somewhat common practice is to not save every single sample, but rather to wait a _xed number of iterations between each save. This prevents samples from beingdependent on each other, but is not necessarily a problem for a well-chosen proposal distribution with enough samples.

Proposal distributions: For a given target distribution p^*, a proposal distribution q is valid or admissible if it gives a non-zero probability of

moving to the states that have non-zero probability in the target. Formally, we can write this as $\text{supp}(p^*) \subseteq \cup_x \text{supp}(q(\cdot|x))$

Following types of proposal distribuitons can be used :

1. **Gaussian proposals:** If we have a continuous state space, the Hessian H at a local mode $\hat{\mathbf{w}}$ can be used to define the covariance of a Gaussian proposal distribution. This approach has the advantage that the Hessian models the local curvature and length scales of each dimension; this approach therefore avoids some of the slow mixing behavior of Gibbs sampling.

2. **Mixture proposals:** If one doesn't know what kind of proposal to use, one can try a mixture proposal, which is a convex combination of base proposals.

$$q(\mathbf{x}'|\mathbf{x}) = \sum_{k=1}^{K} w_k q_k(\mathbf{x}'|\mathbf{x})$$

   where w_k are the mixing weights. As long as each q_k is individually valid, the overall proposal will also be valid.

3. **Data-driven MCMC:** The most efficient proposals depend not just on the previous hidden state, but also the visible data, i.e., they have the form $q(x' |x, D)$. This is called data-driven MCMC. To create such proposals, one can sample (x, D) pairs from the forwards model and then train a discriminative classifier to predict $p(x|f(D))$, where f(D) are some features extracted from the visible data.

Adaptive MCMC: One can change the parameters of the proposal as the algorithm is running to increase efficiency. This is called adaptive MCMC. This allows one to start with a broad covariance , allowing large moves through the space until a mode is found, followed by a narrowing of the covariance to ensure careful exploration of the region around the mode.

It is essential to start MCMC in an initial state that has non-zero probability. If the model has deterministic constraints, finding such a legal configuration may be a hard problem in itself. It is therefore common to initialize MCMC methods at a local mode, found using an optimizer.

Reversible jump (trans-dimensional) MCMC: Suppose we have a set of models with diferent numbers of parameters, e.g., mixture models in which the number of mixture components is unknown. Sampling in spaces of differing dimensionality is called transdimensional MCMC.

The difficulty with this approach arises when we move between models of diferent dimensionality. The problem is that when we compute the MH acceptance ratio, we are comparing densities defined in diferent dimensionality spaces, which is meaningless. It is like trying to compare a sphere with a circle. The solution, proposed by (Green 1998) and known

as reversible jump MCMC or RJMCMC, is to enhance the low dimensional space with extra random variables so that the two spaces have a common measure.

6.4 SPEED AND ACCURACY OF MCMC

In this section, we discuss a number of important theoretical and practical issues to do with MCMC.

1. **The burn-in phase:** We begin MCMC from an arbitrary initial state. Only when the chain has "forgotten" where it started from will the samples be coming from the chain's stationary distribution. Samples collected before the chain has reached its stationary distribution do not come from p^*, and are usually thrown away. The initial period, whose samples will be ignored, is called the burn-in phase.

2. **Mixing rates of Markov chains:** The amount of time it takes for a Markov chain to converge to the stationary distribution, and forget its initial state, is called the mixing time.

3. **Practical convergence diagnostics:** Computing the mixing time of a chain is in general quite difcult, since the transition matrix is usually very hard to compute. In practice various heuristics have been proposed to diagnoseconvergence. Firmly speaking, these methods do not diagnose convergence, but rather non-convergence. That is, the method may claim the chain has converged when in fact it has not. This is a flaw common to all convergence diagnostics, since diagnosing convergence is computationally intractable in general One of the simplest methods to evaluating when the method has converged is to run multiple chains from very diferentoverdispersed starting points, and to plot the samples of some variables of interest. This is called a trace plot. If the chain has mixed, it should have "forgotten" where it started from, so the trace plots should converge to the same distribution, and thus overlap with each other

Accuracy of MCMC:The samples produced by MCMC are auto-correlated, and this reduces their information content relative to independent or "perfect" samples.

A natural question to ask is: how many chains should we run? We could either run one long chain to ensure convergence, and then collect samples spaced far apart, or we could run many short chains, but that wastes the burnin time. In practice it is common to run a medium number of chains (say 3) of medium length (say 100,000 steps), and to take samples from each after discarding the first half of the samples. If we initialize at a local mode, we may be able to use all the samples, and not wait for burn-in.

6.5 SUMMARY

In this chapter we saw Markov **chain Monte Carlo** (**MCMC**) methods which encompass a class of algorithms for sampling from a probability

distribution. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, one can obtain a sample of the desired distribution by recording states from the chain. The more steps that are included, the more closely the distribution of the sample matches the actual desired distribution. Various algorithms exist for constructing chains, like Gibbs sample, Metropolis–Hastings algorithm were discussed.

6.6 EXERCISE

1. What is Markov chain Monte Carlo (MCMC) inference? Why it is needed?
2. What is Gibbs sampling? Give an example
3. Explain working of Gibbs sampling with an example.
4. Explain briefly Metropolis Hastings algorithm.
5. What is proposal distribution ? what are its types?
6. What is reversible jump MCMC?
7. Discuss Speed and accuracy of MCMC.

6.7 REFERENCES

- Machine Learning: A Probabilistic Perspective: Kevin P Murphy, The MIT Press Cambridge (2012).
- Introducing Monte Carlo Methods with R, Christian P. Robert, George Casella, Springer, 2010
- Introduction to Machine Learning (Third Edition): EthemAlpaydın, The MIT Press (2015).
- Pattern Recognition and Machine Learning: Christopher M. Bishop, Springer (2006)

❖❖❖❖

7

GRAPHICAL MODEL STRUCTURE LEARNING

Unit Structure :

7.0 OBJECTIVES

At end of the course the students will be able to:

- Describe the Structured learning and DAGs Learning
- Explain the Gaussian and discrete undirected graph models
- Analyse the different types of graph models to the applications

7.1 INTRODUCTION

A graphical model is a way to represent the dependencies between random variables using a graph. In a graphical model, each node represents a random variable, and each edge represents a conditional dependence between variables.

Graphical models are commonly used in machine learning, statistics, and artificial intelligence to model complex systems and make predictions. They can be used to solve problems such as classification, regression, and clustering.

There are two main types of graphical models: (i) Bayesian networks (ii) Markov networks.

(i)Bayesian networks, also known as belief networks, represent the joint probability distribution of a set of random variables as a directed acyclic graph.

(ii)Markov networks, also known as undirected graphical models or Markov random fields, represent the joint probability distribution as an undirected graph.

Graphical models provide a compact and intuitive way to represent complex probabilistic relationships between variables, and they have been widely used in a variety of fields including computer vision, natural language processing, and bioinformatics.

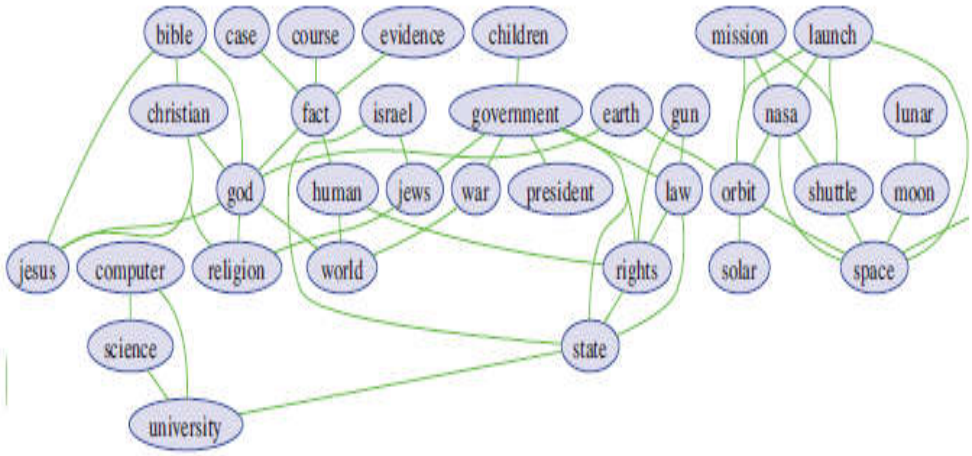


Figure 7.1Part of a relevance network constructed from the 20-news

7.2 STRUCTURE LEARNING FOR KNOWLEDGE DISCOVERY

Since computing the MAP graph or the exact posterior edge marginal is in general computationallyintractable.

Methodsfor learning graph structures used to visualize one's data are:

(i) Relevance networks **(ii) Dependency networks**

Limitations:

The resulting models donot constitute consistent joint probability distributions, so they cannot be used for prediction,and they cannot even be formally evaluated in terms of goodness of fit.

Benefit:

Thesemethods are a useful ad hoc tool based on theirspeed and simplicity.

7.2.1 Relevance networks

A **relevance network** is a way of visualizing the pairwise mutual information between multiple random variables:

Choose a threshold and draw an edge from node i to node j if‖ (Xi;Xj) is above this threshold.

In the Gaussian case

$$\mathbb{I}(X_i; X_j) = -\frac{1}{2}\log(1 - \rho_{ij}^2)$$

Where ρ_{ij} is the correlation coefficient,

Illustrate the idea using natural languagetext. Figure 7.1 gives an example, where MI (Mutual Information)is visualizedbetween words in the newsgroupdataset from Figure 7.2.

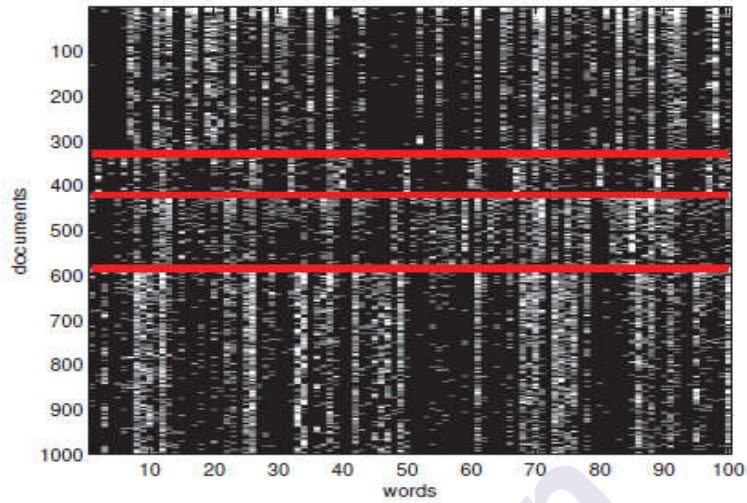


Figure 7.2 Newsgroup dataset

Subset of size 16242 x 100 of the 20-newsgroups data. Each row is a document (represented as a bag-of-words bit vector), each column is a word. The red lines separate the 4 classes, which are (in descending order) comp, rec, sci, talk,there are subsets of words whose presence or absence is indicative of the class.

Drawback:

The graphs are usually very dense,since most variables are dependent on most other variables, even after thresholding the MIs.

For example, suppose X_1 directly influences X_2 which directly influences X_3. Then X_1 has non-zero MI with X_3, so there will be a $1 - 3$ edge in the relevance network.

Indeed, most pairs will beconnected.

A better approach is to use graphical models, which represent conditional independence, rather than dependence. In the above example, X_1 is conditionally independent of X_3 given X_2, so there will not be a $1 - 3$ edge. Consequently graphical models are usually much sparser than relevance networks, and hence are a more useful way of visualizing interactions between multiple variables

7.2.2 Dependency networks

Dependency network :A simple and efficient way to learn a graphical model structure is to independently fit D sparsefull-conditional distributions $p(x_t| X-t)$.

The chosen variables constitute the inputs to the node, i.e., its Markov blanket. Then the resulting sparse graph can be visualized.

Advantage over relevance networks: Redundant variables will not be selected as inputs.

Any kind of sparse regression or classification method is used to fit each CPD (Conditional Probability Distribution) uses classification/ regression trees, use ℓ_1-regularized linear regression, use ℓ_1-regularized logistic regression,uses Bayesian variable selection, etc.

Figure 7.3 shows a dependency network that was learned from the 20-newsgroup data using ℓ_1 regularized logistic regression, where the penalty parameter λ was chosen by Bayesian Information Criterion (BIC). Many of the words present in these estimated Markov blankets represent fairly natural associations However, some of the estimated statistical dependencies seem less intuitive, such as baseball:windows and bmw:christian. We can gain more insight if we look not only at the sparsity pattern, but also the values of the regression weights. For example, here are the incoming weights for the first 5 words shown in Table 7.1.

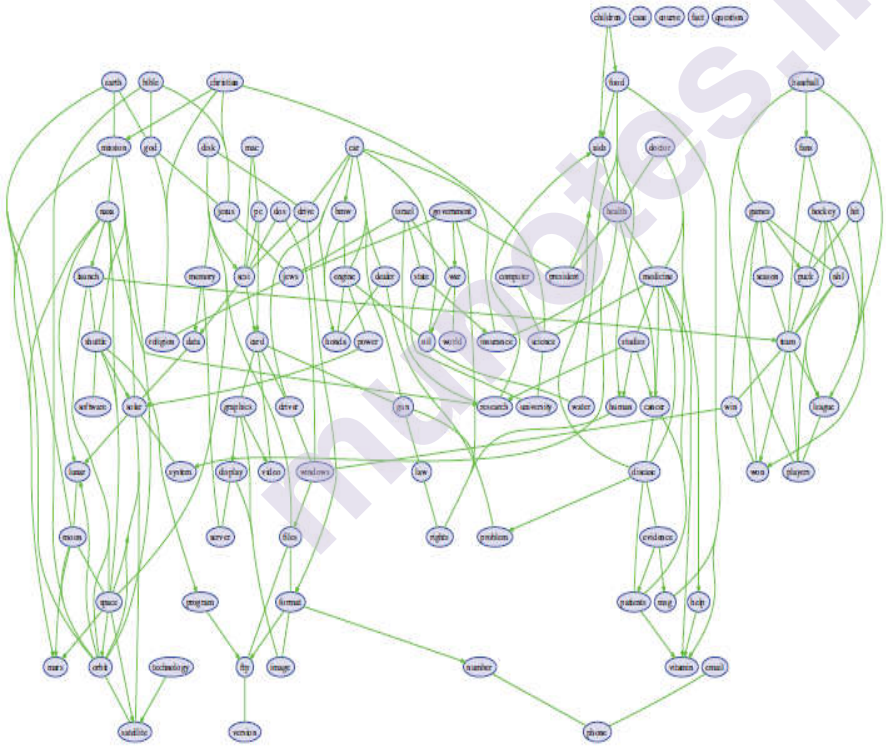


Figure 7.3 Dependency networkfrom the 20-

Words in italic red have negative weights, which represents a dissociative relationship. Forexample, the model reflects that baseball:windows is an unlikely combination. It shows thatmost of the weights are negative (1173 negative, 286 positive, 8541 zero) in this model.

7.3 LEARNING TREE STRUCTURES

The fully specified joint probability models can be used for density estimation, prediction and knowledge discovery.The problem of structure learning for general graphs is NP-hard, so start by considering the special case of trees.

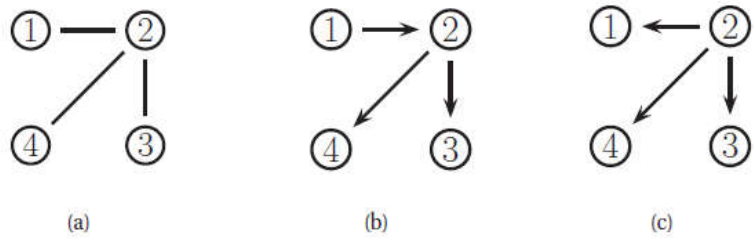


Figure 7.4 An undirected tree and two equivalent directed trees

Table 7.1

7.3.1 Directed or undirected tree

A directed tree, with a single root node r, defines a joint distribution as follows:

Words	Relationship and Weights										
aids	children (0.53)	disease (0.84)	Fact (0.47)	health (0.77)	president (0.50)	research (0.53)					
Base ball	christian (-0.98)	drive (-0.49)	games (0.81)	God (-0.46)	government (-0.69)	hit (0.62)	memory (-1.29)	players (1.16)	season (0.31)	Software (-0.68)	Windows (-1.45)
bible	Card (-0.88)	christian (0.49)	Fact (0.21)	god (1.01)	Jesus (0.68)	orbit (0.83)	Car (-0.72)	program (-0.56)	religion (0.24)	version (0.49)	
bmw	car (0.60)	christian (-11.54)	engine (0.69)	god (-0.74)	government (-1.01)	help (-0.50)	windows (-1.43)				
cancer	Disease (0.62)	medicine (0.58)	patients (0.90)	research (0.49)	studies (0.70)						

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t|x_{\mathrm{pa}(t)})$$

where we define pa(r) = ∅. For example, in Figure 7.4(b-c),

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) \end{aligned}$$

The choice of root does not matter: both of these models are equivalent.

To make the model more symmetric, it is preferable to use an undirected tree. This can berepresented as follows:

$$p(\mathbf{x}|T) = \prod_{t \in V} p(x_t) \prod_{(s,t) \in E} \frac{p(x_s, x_t)}{p(x_s)p(x_t)}$$

where $p(x_s, x_t)$ is an edge marginal and $p(x_t)$ is a node marginal. For example, in Figure 7.4(a)

$$p(x_1, x_2, x_3, x_4|T) = p(x_1)p(x_2)p(x_3)p(x_4)\frac{p(x_1, x_2)p(x_2, x_3)p(x_2, x_4)}{p(x_1)p(x_2)p(x_2)p(x_3)p(x_2)p(x_4)}$$

$$\begin{aligned} p(x_1, x_2, x_3, x_4|T) &= p(x_1, x_2)\frac{p(x_2, x_3)}{p(x_2)}\frac{p(x_2, x_4)}{p(x_2)} \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2) \\ &= p(x_2)p(x_1|x_2)p(x_3|x_2)p(x_4|x_2) \end{aligned}$$

where $p(x_t|x_s) = p(x_s, x_t)/p(x_s)$.

A tree can be represented as either an undirected or directed graph: the number ofparameters is the same, and hence the complexity of learning is the same. Theinference is the same in both representations.

The undirected representation, which issymmetric, is useful for structure learning, but the directed representation is more convenientfor parameter learning.

7.3.2 Chow-Liu algorithm for finding the ML tree structure

The log-likelihood for a tree is as follows:

$$\begin{aligned} \log p(\mathcal{D}|\boldsymbol{\theta}, T) &= \sum_t \sum_k N_{tk} \log p(x_t = k|\boldsymbol{\theta}) \\ &+ \sum_{s,t} \sum_{j,k} N_{stjk} \log \frac{p(x_s = j, x_t = k|\boldsymbol{\theta})}{p(x_s = j|\boldsymbol{\theta})p(x_t = k|\boldsymbol{\theta})} \end{aligned}$$

N_{stjk} : the number of times;

s : node

j,I,k : state in node s

N_{tk} :Number of times node t in state k

Rewriting these counts in terms of the empiricaldistribution:

$N_{stjk} = Np_{emp}$ (x_s = j, x_t= k) and $N_{tk} = Np_{emp}$(x_t= k).

Setting θ to theMLEs, becomes

$$\frac{\log p(\mathcal{D}|\boldsymbol{\theta}, T)}{N} = \sum_{t \in \mathcal{V}} \sum_{k} p_{\text{emp}}(x_t = k) \log p_{\text{emp}}(x_t = k) + \sum_{(s,t) \in \mathcal{E}(T)} \mathbb{I}(x_s, x_t | \hat{\boldsymbol{\theta}}_{st})$$

where $\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) \geq 0$ is the mutual information between x_s and x_t given the empirical distribution:

$$\mathbb{I}(x_s, x_t|\hat{\boldsymbol{\theta}}_{st}) = \sum_{j} \sum_{k} p_{\text{emp}}(x_s = j, x_t = k) \log \frac{p_{\text{emp}}(x_s = j, x_t = k)}{p_{\text{emp}}(x_s = j) p_{\text{emp}}(x_t = k)} \quad (26.12)$$

Chow-Liu algorithm : The tree topology that maximizes the likelihood can be found bycomputing the maximum weight spanning tree, where the edge weights are the pairwise mutualinformations,$\mathbb{I}(y_s, y_t|\hat{}\theta_{st})$.

There are several algorithms for finding a max spanning tree (MST).

Prim's algorithm and Kruskal's algorithm are the best algorithms to run in O(E log V) time,

where $E = V^2$ is the number of edges and V is the number of nodes.

So, overall running time is $O(NV^2 + V^2 \log V)$, where thefirst term is the cost of computing the sufficient statistics.

Figure 7.5 gives an example of the method in action, applied to the binary 20 newsgroupsdata. The tree has been arbitrarily rooted at the node representing "email".

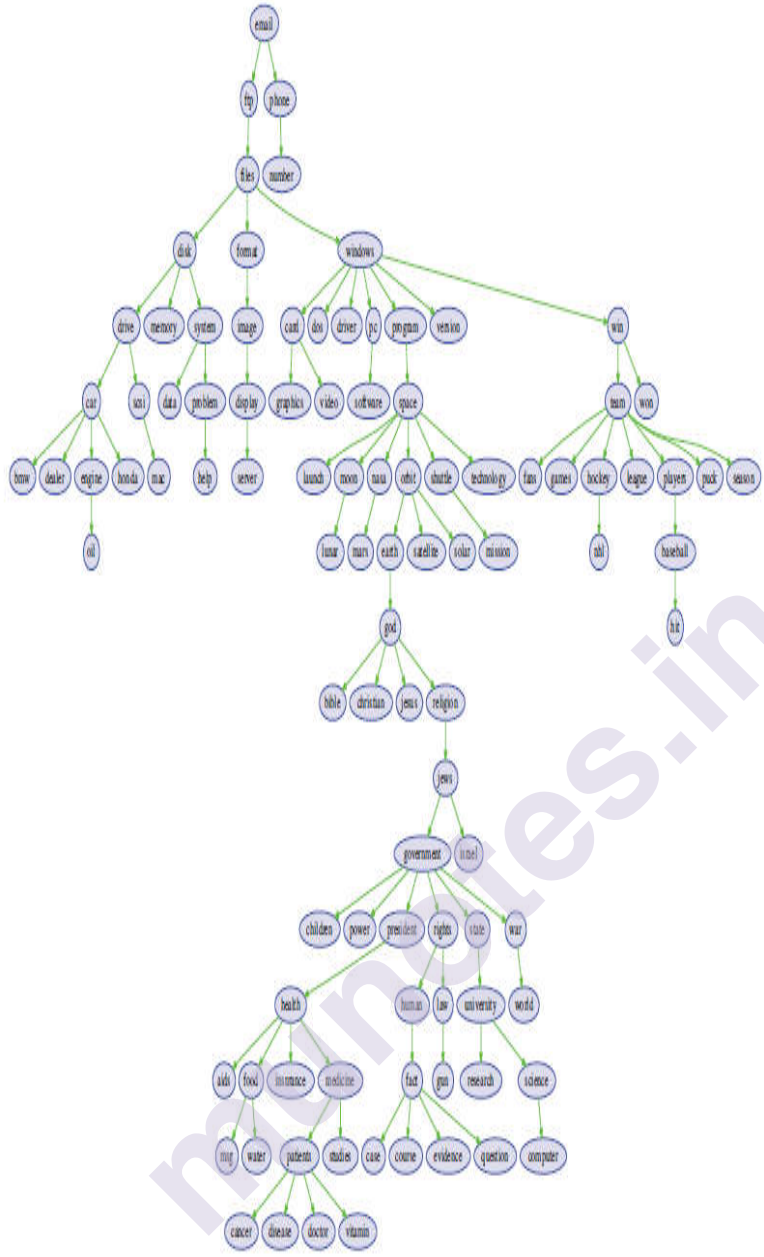


Figure 7.5 The MLE tree on

7.3.3 Finding the MAP forest

Since all trees have the same number of parameters, the maximum likelihoodscore is used as a model selection criterion

Since inference in a forest is much faster thanin a tree a **forest is** fitted rather than a single tree,The MLE criterionwill never choose to omit an edge. However, if the marginal likelihood or a penalizedlikelihood (such as BIC), are used the optimal solution may be a forest. The details for themarginal likelihood case resulting expression:

$$\log p(\mathcal{D}|T) = \sum_{t \in \mathcal{V}} \log \int \prod_{i=1}^{N} p(x_{it}|\mathbf{x}_{i,\mathrm{pa}(t)}|\boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t = \sum_t \mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)})$$

where $N_{t,pa(t)}$ are the counts for node t and its parents, and score is

$$\mathrm{score}(\mathbf{N}_{t,\mathrm{pa}(t)}) \triangleq \prod_{c=1}^{C_t} \frac{B(\mathbf{N}_{tc} + \boldsymbol{\alpha}_{tc})}{B(\boldsymbol{\alpha}_{tc})}$$

DAGs with at most one parent.

Associate a weight with each s → t edge, ws,t ! score(t|s) − score(t|0), wherescore(t|0) is the score when t has no parents.

Then the objective as follows:

$$\log p(\mathcal{D}|T) = \sum_t \mathrm{score}(t|\mathrm{pa}(t)) = \sum_t w_{\mathrm{pa}(t),t} + \sum_t \mathrm{score}(t|0)$$

7.3.4 Mixtures of trees

- A single tree is rather limited in its expressive power.
- A**mixture of trees is** mixture of component may have a different tree topology. This is an unsupervised version ofthe TAN classifier.
- It can be fitted a mixture of trees by using EM: in theE step, compute the responsibilities of each cluster for each data point, and in the M step,use a weighted version of the Chow-Liu algorithm.
- It is possible to create an "infinite mixture of trees", by integrating out over all possibletrees. This can be done in V^3 time using the matrix tree theorem.

7.4 LEARNING DAG STRUCTURE WITH LATENT VARIABLES

Sometimes the complete data assumption does not hold, either because of missing data,and/ or because of hidden variables. In this case, the marginal likelihood is given by

$$p(\mathcal{D}|G) = \int \sum_{\mathbf{h}} p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta} = \sum_{\mathbf{h}} \int p(\mathcal{D}, \mathbf{h}|\boldsymbol{\theta}, G) p(\boldsymbol{\theta}|G) d\boldsymbol{\theta}$$

where h represents the hidden or missing data.

In general this is intractable to compute. For example, consider a mixture model, wherethe cluster label is not observed. In this case, there are KN possible completions of thedata;

The inner integral can be evaluated for each one of theseassignments to h.

7.4.1 Approximating the marginal likelihood for missing data

The simplest approach is to use standard structure learning methods for fully visible DAGs,but to approximate the marginal likelihood. some faster deterministic approximations.

(i) BIC approximation

(ii) Cheeseman-Stutz approximation

(iii) Variational Bayes EM

(i) BIC approximation

A simple approximation is to use the BIC score, which is given by

$$\mathrm{BIC}(G) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{\log N}{2}\dim(G)$$

where dim(G) is the number of degrees of freedom in the model

ˆθ is the MAP(Maximum A Posteriori) or MLestimate.

(ii) Cheeseman-Stutz approximation (CS)

Compute a MAP estimate of the parameters ˆθ.

Denote the expected sufficient statistics of the data by D = D(ˆθ);

In the case of discretevariables, “fill in” the hidden variables with their expectation.

Then use the exactmarginal likelihood equation on this filled-in data:

$$p(\mathcal{D}|G) \approx p(\overline{\mathcal{D}}|G) = \int p(\overline{\mathcal{D}}|\boldsymbol{\theta}, G)p(\boldsymbol{\theta}|G)d\boldsymbol{\theta}$$

To sum over all values of h

$$\log p(\mathcal{D}|G) = \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G)$$

and then apply a BIC approximation to the last two terms:

$$\begin{aligned}\log p(\mathcal{D}|G) - \log p(\overline{\mathcal{D}}|G) &\approx \left[\log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] \\ &\quad - \left[\log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G) - \frac{N}{2}\dim(\hat{\boldsymbol{\theta}})\right] \\ &= \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)\end{aligned}$$

$$\log p(\mathcal{D}|G) \approx \log p(\overline{\mathcal{D}}|G) + \log p(\mathcal{D}|\hat{\boldsymbol{\theta}}, G) - \log p(\overline{\mathcal{D}}|\hat{\boldsymbol{\theta}}, G)$$

p(D|G) - Computed by plugging in the filled-in data into the exact marginallikelihood.

p(D|ˆθ ,G) - Computed using an inference algorithm.

The finalterm p(D|ˆθ,G) can be computed by plugging in the filled-in data into the regular likelihood.

(iii) Variational Bayes EM

An even more accurate approach is to use the variational Bayes EM algorithm.

key idea is to make the following factorization assumption:

$$p(\theta, \mathbf{z}_{1:N}|\mathcal{D}) \approx q(\theta)q(\mathbf{z}) = q(\theta)\prod_i q(\mathbf{z}_i)$$

where z_i are the hidden variables in case i. In the E step, update the q(zi), and in theM step, update q(θ). The corresponding variational free energy provides a lower bound onthe log marginal likelihood.

Example: college plans revisited

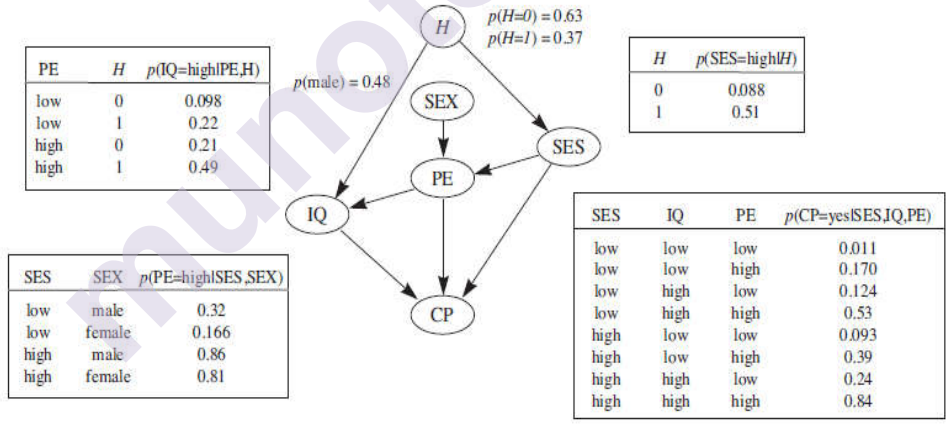


Figure 7.6 The most probable DAG with a single binary hidden variable learned from the Sewell-Shah data. MAP estimates of the CPT entries are

Let us revisit the college plans dataset.

- Sex Male or female
- SES Socio economic status: low, lower middle, upper middle or high.
- IQ Intelligence quotient: discretized into low, lower middle, upper middle or high.
- PE Parental encouragment: low or high
- CP College plans: yes or no.

If ignore thepossibility of hidden variables are ignored there was a direct link from socio economic status to IQ in theMAP DAG.

Alternate:

- Introduce a hiddenvariable H, which is considered as a parent of both SES and IQ, representing a hidden common cause
- Consider a variant in which H points to SES, IQ and PE.
- Consider dropping none, one, or both of the SES-PE and PE-IQ edges.
- Vary the numberof states for the hidden node from 2 to 6.
- Compute the approximate posterior over8 × 5 = 40 different models, using the CS approximation.
- Most probable model found is shown in Figure 7.6.
- This is 2 · 1010 timesmore likely than the best model containing no hidden variable.
- It is also 5 · 109 times morelikely than the second most probable model with a hidden variable.
- So again the posterior isvery peaked.

Results suggest that there is a hidden common cause underlying both thesocio-economic status of the parents and the IQ of the children.

By examining the CPT entries,that both SES and IQ are more likely to be high when H takes on the value 1.

7.4.2 Structural EM

Structural EM algorithm: Instead of fitting each candidate neighboring graph and then filling in its data, fill in the data once, and use this filled-in data to evaluate the score of all the neighbors.

It is a bad approximation to the marginal likelihood,

It is a good enough approximation of the difference in marginal likelihoods between different models, in order to pick the best neighbor.

More precisely, define $D(G_0,\hat{\theta}_0)$ to be the data filled in using model G_0 with MAP parameters

$$\text{score}_{\text{BIC}}(G,\mathcal{D}) \triangleq \log p(\mathcal{D}|\hat{\boldsymbol{\theta}},G) - \frac{\log N}{2}\dim(G) + \log p(G) + \log p(\hat{\boldsymbol{\theta}}|G)$$

Which includes the log prior for the graph and parameters.

A graph G which increases the BIC score relative to G_0 on the expecteddata, it will also increase the score on the actual data, i.e.,

$$\text{score}_{\text{BIC}}(G,\overline{\mathcal{D}}(G_0,\hat{\theta}_0)) - \text{score}_{\text{BIC}}(G_0,\overline{\mathcal{D}}(G_0,\hat{\theta}_0) \leq \text{score}_{\text{BIC}}(G,\mathcal{D}) - \text{score}_{\text{BIC}}(G_0,\mathcal{D}($$

Algorithm:

- Initialize with some graphG_0 and some set of parameters θ_0.
- Fill-in the data using the current parameters—the expected counts for any particular family, Performinference using current model.

- Evaluate the BIC score of all of the neighbors usingthe filled-in data, and pick the best neighbor.
- Refit the model parameters, fill-in thedata again, and repeat.
- For increased speed, choose only refit the model every fewsteps, since small changes to the structure hopefully won't invalidate the parameter estimatesand the filled-in data too much.

Applications: Learn a phylogenetic tree structure., to learn sparse mixture models

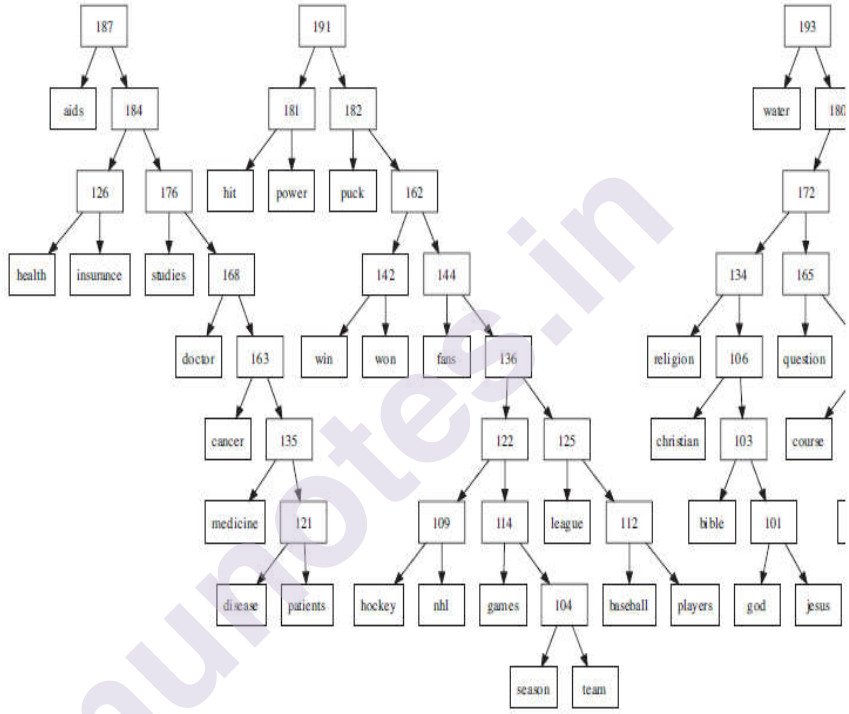


Figure 7.7 Part of a hierarchical latent tree learned from the 20-

7.4.3 Discovering hidden variables

Performs structure learning in the visible domain, and then look for **structuralsignatures**, such as sets of densely connected nodes (near-cliques); introduce a hidden variableand connect it to all nodes in this near-clique; and then let structural EM sort out the details.

Limitation:

This technique does not work too well, since structure learning algorithms arebiased against fitting models with densely connected cliques.

Latentclass model: It is aflat mixture model; the discrete latent variable provides a compressed representation of its children.Create hidden variables with high mutual information with their children.

Hierarchical latent class model: Creating a tree-structured hierarchy of

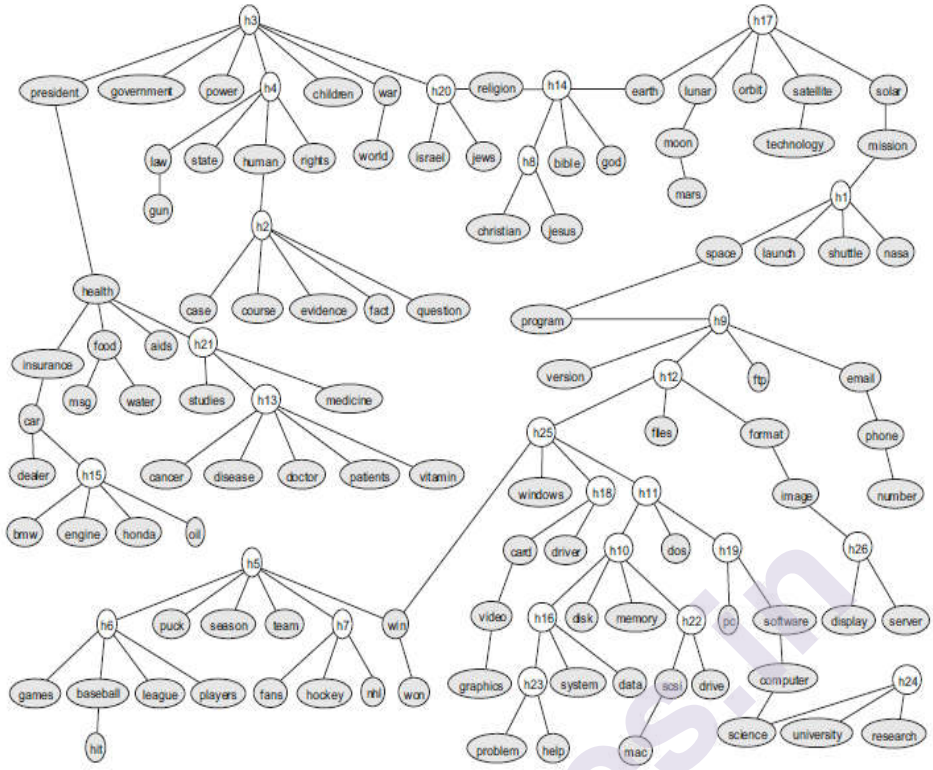


Figure 7.8 A partially latent tree learned from the 20-newsgroup data. Note that some words can have multiple meanings, and get connected to different latent variables, representing different "topics".

latent variables, each of whichonly has to explain a small set of children. A greedy local search algorithm is used to learn such structures, based on addingor deleting hidden nodes, adding or deleting edges, etc.

Figure 7.7 shows separate clusters concerning medicine, sports and religion. Thisprovides an alternative to LDA and other topic models with the added advantagethat inference in latent trees is exact and takes time linear in the number of nodes.

In an alternative approach the observed data is notconstrained to be at the leaves.

This method starts with the Chow-Liu tree on the observeddata, and then adds hidden variables to capture higher-order dependencies between internalnodes.

This results in much more compact models, as shown in Figure 7.8.

Advantage:

Has better predictive accuracy than other approaches, such as mixture models, or trees whereall the observed data is forced to be at the leaves.

7.4.4 Structural equation models (SEM)

Structural equation model (Path Diagrams) : Special kind of directed mixed graph possibly cyclic, in which all CPDs are linear Gaussian, and in which all bidirectededges represent correlated Gaussian noise.

SEMsare widely used in economics and social science.

SEM - a series of full conditionals as follows:

$$x_i = \mu_i + \sum_{j \neq i} w_{ij} x_j + \epsilon_i$$

where □ ~ N(0,Ψ).

The model is rewritten in matrix form as follows:

$x = Wx + \mu + \square \Rightarrow x = (I - W)^{-1}(\square + \mu)$

Hence the joint distribution is given by $p(x) = N(\mu, \Sigma)$ where

$\Sigma = (I - W)^{-1} \Psi (I - W)^{-T}$

Draw an arc $X_i \leftarrow X_j$ if $|w_{ij}| > 0$.

- If W is lower triangular then the graph is acyclic.
- If,in addition, Ψ is diagonal, then the model is equivalent to a Gaussian DGM,; such models are called **recursive**.
- If Ψ is not diagonal, then draw a bidirected arc Xi ↔ Xj for each non-zero off-diagonal term. Such edges represent correlation,
- When using structural equation models, it is common to partition the variables into latentvariables, Z_t, and observed or **manifest** variables Y_t.

For example, Figure 7.9 illustrates thefollowing model:

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} = \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & w_{13} & 0 & 0 & 0 \\ w_{21} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{32} & 0 & 0 & 0 & 0 \\ w_{41} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{52} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{63} & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{pmatrix},$$

where

$$\Psi = \begin{pmatrix} \Psi_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \Psi_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \Psi_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Psi_{44} & \Psi_{45} & 0 \\ 0 & 0 & 0 & \Psi_{54} & \Psi_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \Psi_{66} \end{pmatrix}$$

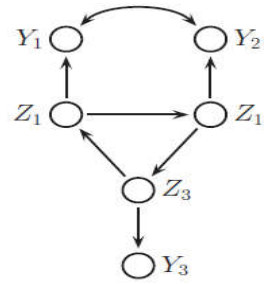


Figure 7.9A cyclic directed mixed graphical model (non-recursive SEM). Z1 → Z2 → Z3 → Z1 is feedback loop.

The presence of a feedback loop Z1 → Z2 → Z3 is evident from the fact that W is not lowertriangular. Also the presence of confounding between Y1 and Y2 is evident in the off-diagonalterms in Ψ.

7.5 LEARNING CAUSAL DAGS

Causal models are models which can predict the effects of interventions to, or manipulationsof, a system.

Example :

(i)An electronic circuit diagram implicitly provides a compact encodingof what will happen if one removes any given component, or cuts any wire.

(ii) A causal medicalmodel might predict that if I continue to smoke, I am likely to get lung cancer.

Causal claims are inherently stronger,yet more useful, than purely **associative** claims, such as "people who smoke often have lungcancer".Causal models are often represented by DAGs

7.5.1 Causal interpretation of DAGs

Causal Markov Assumption: Directed edge A → B in a DAG to mean that "A directly causes B",

so if A is manipulated, then B will change.

Causal sufficiency assumption: Assuming that all relevant variables are included in the model, i.e., there are nounknown **confounders**, reflecting hidden common causes.

Perfect intervention: Assuming that the causal Markov and causal sufficiency assumptions, use DAGs to answer causal questions.

This represents the act of setting a variable to some known value, say setting X_i to x_i.

A realworld example: A gene knockout experiment, in which a geneis "silenced".

Notational conventions used to distinguish this from observing that X_i happens to have value x_i

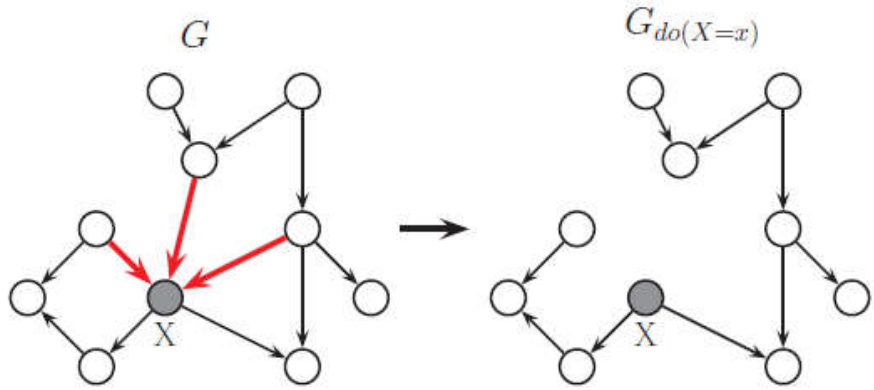


Figure 7.10Surgical interventions on X. Based on (Pe'er 2005).

Use Pearl's **do calculus** notation to denote the event that we set X_i to $x_i$as:

$do(X_i = xi)$

A causal model can be used to makeinferences of the form $p(x|do(X_i = x_i))$, which is different from making inferences of the form$p(x|X_i = x_i)$

Difference between conditioning on interventions and conditioning onobservations:

Consider a 2 node DGM $S \rightarrow Y$,

S = 1 if you smoke and S = 0 otherwise,

Y = 1 if you have yellow-stained fingers,and Y = 0 otherwise.

If you are observed to having yellow fingers, I am licensed to infer that you areprobably a smoker

$p(S = 1|Y = 1) > p(S = 1)$ (26.49)

However, if I intervene and paint your fingers yellow, I am no longer licensed to infer this, sinceI have disrupted the normal causal mechanism. Thus

$p(S = 1|do(Y = 1)) = p(S = 1)$ (26.50)

graph surgery: (Method to model perfect interventions) represent the joint distributionby a DGM, and then cut the arcs coming into any nodes that were set by intervention.

Example :Figure 7.10. This prevents any information flow from the nodes that wereintervened on from being sent back up to their parents.

Theorem. To compute p(Xi|do(Xj))for sets of nodes i, j, we can perform surgical intervention on the Xj nodes and then use standardprobabilistic inference in the mutilated graph.

Augmented DAG: Generalize the notion of a perfect intervention by adding interventions as explicitaction nodes to the graph. The result is like an influence diagram, except there are no utilitynodes. Define the CPD p(Xi|do(Xi)) .

Fat hand intervention: Allowing an action toaffect multiple nodes; a reference to someone trying tochange a single component of some system (e.g., an electronic circuit), but accidently touchingmultiple components and thereby causing various side effects.

7.5.2 Using causal DAGs to resolve Simpson's paradox

In this section, causal reasoning is performed byapplying d-separation to the mutilated graph.

Simpon's paradox.: Any statistical relationship between two variables can be reversedby including additional factors in the analysis. For example, suppose some cause C (say, takinga drug) makes some effect E (say getting better) more likely

P (E|C) >P (E|¬C)

and yet, when condition on the gender of the patient is provoked, it has been found that taking the drug makesthe effect less likely in both females (F) and males (¬F):

P(E|C, F) < P(E|¬C, F)

P(E|C, ¬F) < P(E|¬C, ¬F)

This seems impossible, but by the rules of probability, this is perfectly possible, because theevent space where the condition triggered on (¬C, F) or (¬C, ¬F) can be completely different to theevent space when a condition activated on ¬C. The table of numbers below shows a concrete example.

	Combined				Male				Female			
	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate	E	$\neg E$	Total	Rate
C	20	20	40	50%	18	12	30	60%	2	8	10	20%
$\neg C$	16	24	40	40%	7	3	10	70%	9	21	30	30%
Total	36	44	80		25	15	40		11	29	40	

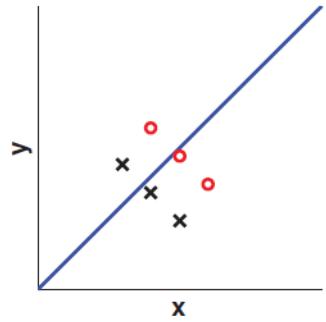
From this table of numbers,

p(E|C) = 20/40 = 0.5 > p(E|¬C) = 16/40 = 0.4 (26.51)

p(E|C, F) = 2/10 = 0.2 < p(E|¬C, F) = 9/30 = 0.3 (26.52)

p(E|C, ¬F) = 18/30 = 0.6 < p(E|¬, ¬F) = 7/10 = 0.7 (26.53)

A visual representation of the paradox is given in in Figure 7.11.



Line- goes up andto the right: Effect (y-axis) increases as the cause (x-axis) increases.

Dots - Data for females,

Crosses-Data for males.

In eachsubgroup,the effect decreases as the cause increase.

Figure 7.11 Illustration of Simpson's paradox

Effect is real, but it is still very counter-intuitive. The reason the paradoxarises is that the statements are interpreted causally, but not using proper causalreasoning when performing the calculations. The statement that the drug C causes recovery Eis

(i) $P(E|do(C)) > P(E|do(\neg C))$

whereas the data shows

(ii) $P(E|C) > P(E|\neg C)$

This is not a contradiction. Observing C is positive evidence for E, since more males thanfemales take the drug, and the male recovery rate is higher (regardless of the drug). SoEquation (i) does not imply Equation (ii).

7.5.3 Learning causal DAG structures

There are two ways to learn causal DAG structures.

(i) Learning from observational data

(ii) Learning from interventional data

(i) Learning from observational data

Even for infinite data, an optimal method can onlyidentify the DAG up to Markov equivalence, it can identify the PDAG(partially directed acylic graph), but not the complete DAG structure, because all DAGs which areMarkov equivalent have the same likelihood.

Algorithms such as **greedy equivalence search** method are

Consistent estimators of PDAG structure,

Identify thetrue Markov equivalence class as the sample size goes to infinity,

(Assumptions: Observe all thevariables. Assume that the generating distribution p is **faithful** tothe generating DAG G).

All the conditional indepence (CI) properties of p areexactly captured by the graphical structure, so I(p) = I(G);

Stable distribution: A faithful distribution that cannot be any CIproperties in p that are due to particular settings of the parameters (such as zeros in a regressionmatrix) that are not graphically explicit.

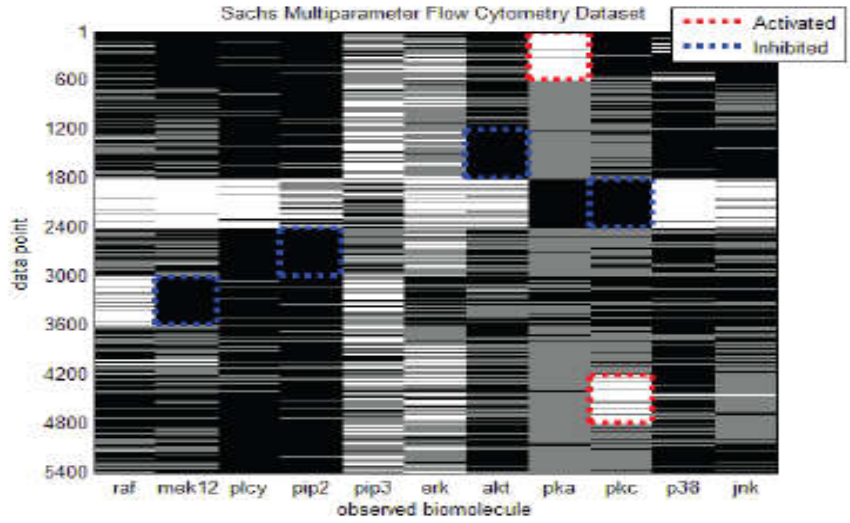
(ii) Learning from interventional data

Interventional data: Used to distinguish between DAGs within the equivalence class, which have certain variables have been set, and the consequences have been measured.

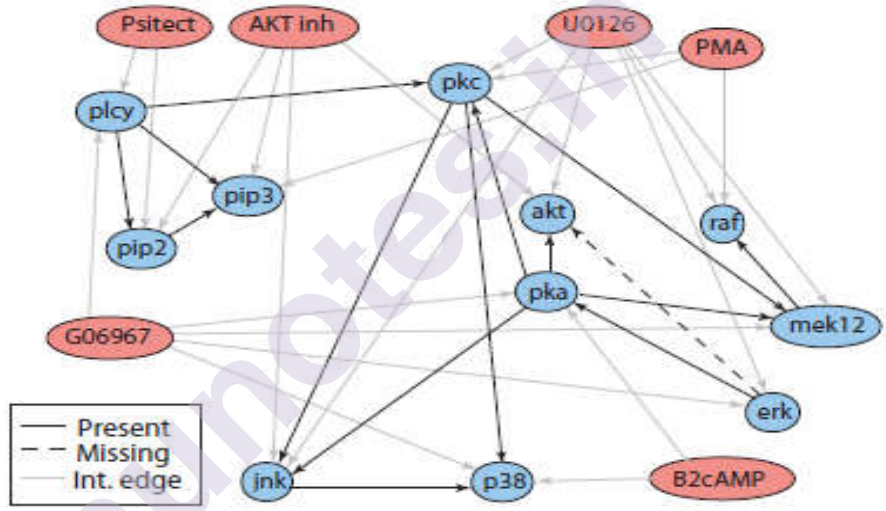
Example of this is the dataset in Figure 7.12(a)

In the example proteins in a signalling pathwaywere agitated, and their phosphorylation status was measured using a technique called flowcytometry.

It is straightforward to modify the standard Bayesian scoring criteria, such as the marginallikelihood or BIC score, to handle learning from mixed observational and experimental data:



(a)



(b)

Figure 7.12 (a) A design matrix consisting of 5400 data points (rows) measuring the status (using flow cytometry) of 11 proteins (columns) under different experimental conditions. The data has been we discretized into 3 states: low (black), medium (grey) and high (white). Some proteins were explicitly controlled using activating or inhibiting chemicals.

(b) A directed graphical model representing dependencies between various proteins (blue circles) and various experimental interventions (pink ovals), which was inferred from this data. All edges for which p(Gst = 1|D) >0.5are plotted.
Dotted edges are believed to exist in nature but were not discovered by the algorithm (1 false negative). Solid edges are true positives. The light colored edges represent the effects of intervention.

Compute the sufficient statistics for a CPD's parameter by skipping over the cases where thatnode was set by intervention For example, when using tabular CPDs, the counts are modified as follows:

$$N_{tck} \triangleq \sum_{i:x_{it} \text{ not set}} \mathbb{I}(x_{i,t} = k, \mathbf{x}_{i,\text{pa}(t)} = c)$$

Figure 7.12(b) shows the augmented DAG that was learned from the interventional flowcytometry data depicted in Figure 7.12(a).

In particular, plot the median graph, whichincludes all edges for which p(Gij = 1|D) > 0.5. It turns out that, in this example, the median model has exactlythe same structure as the optimal MAP model, argmaxG p(G|D).

7.6 LEARNING UNDIRECTED GAUSSIAN GRAPHICAL MODELS

Learning the structure of undirected graphical models is easier than learning DAG structure. This precludes thekind of local search methods (both greedy search and MCMC sampling) are used to learn DAGstructures, because the cost of evaluating each neighboring graph is too high, since refitting each model from scratch to be done.

Solutions to this problem, is arrived in the context of Gaussianrandom fieldsor undirected Gaussian graphical models (GGM)s.

7.6.1 MLE for a GGM

The task ofcomputing the MLE for a (non-decomposable) GGM is called **covariance selection**

The log likelihood can be written as

$\ell(\Omega) = \log\det\Omega - \text{tr}(S\Omega)$

where $\Omega = \Sigma - 1$ is the precision matrix, and

$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$ is the empiricalcovariance matrix.

The gradient of this is given by

$\nabla\ell(\Omega) = \Omega - 1 - S$

However, enforce the constraints that $\Omega_{st} = 0$ if $G_{st} = 0$ (structural zeros), andthat Ω is positive definite.

Show that the MLE must satisfy the following property:

$\Sigma_{st} = S_{st}$ if $G_{st} = 1$ or s = t, i.e., the covariance of a pair that are connected by an edge must match theempirical covariance. In addition, $\Omega_{st} = 0$ if

G_{st} = 0, by definition of a GGM, i.e.,the precision of a pair that are not connected must be 0.

∑is a positive definite**matrix completion** of S, since it retains as many of the entries in S as possible, correspondingto the edges in the graph, subject to the required sparsity pattern on ∑−1, corresponding to theabsent edges; the remaining entries in ∑ are filled in so as to maximize the likelihood.

Example: Use the followingadjacency matrix, representing the cyclic structure, X1−X2−X3−X4−X1, and the followingempirical covariance matrix:

$$G = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad S = \begin{pmatrix} 10 & 1 & 5 & 4 \\ 1 & 10 & 2 & 6 \\ 5 & 2 & 10 & 3 \\ 4 & 6 & 3 & 10 \end{pmatrix}$$

The MLE is given by

$$\Sigma = \begin{pmatrix} 10.00 & 1.00 & 1.31 & 4.00 \\ 1.00 & 10.00 & 2.00 & 0.87 \\ 1.31 & 2.00 & 10.00 & 3.00 \\ 4.00 & 0.87 & 3.00 & 10.00 \end{pmatrix}, \quad \Omega = \begin{pmatrix} 0.12 & -0.01 & 0 & -0.05 \\ -0.01 & 0.11 & -0.02 & 0 \\ 0 & -0.02 & 0.11 & -0.03 \\ -0.05 & 0 & -0.03 & 0.13 \end{pmatrix}$$

The constrained elements in Ω,and the free elements in Σ, both of which correspond to absent edges, have been highlighted.

7.6.2 Graphical lasso

Learn a sparse graph structure by using an objective that encourages zerosin the precision matrix. By analogy to lasso one can define the following $l_1$penalized NLL:

$J(\Omega) = -\log \det\Omega + \text{tr}(S\Omega) + \lambda||\Omega||_1$

where $||\Omega||_1 = \sum_{i,j} |w_{j,k}|$ is the 1-norm of the matrix. This is called the **graphical lasso** or **Glasso**.

The objective is convex, but it is non-smooth and is constrained.

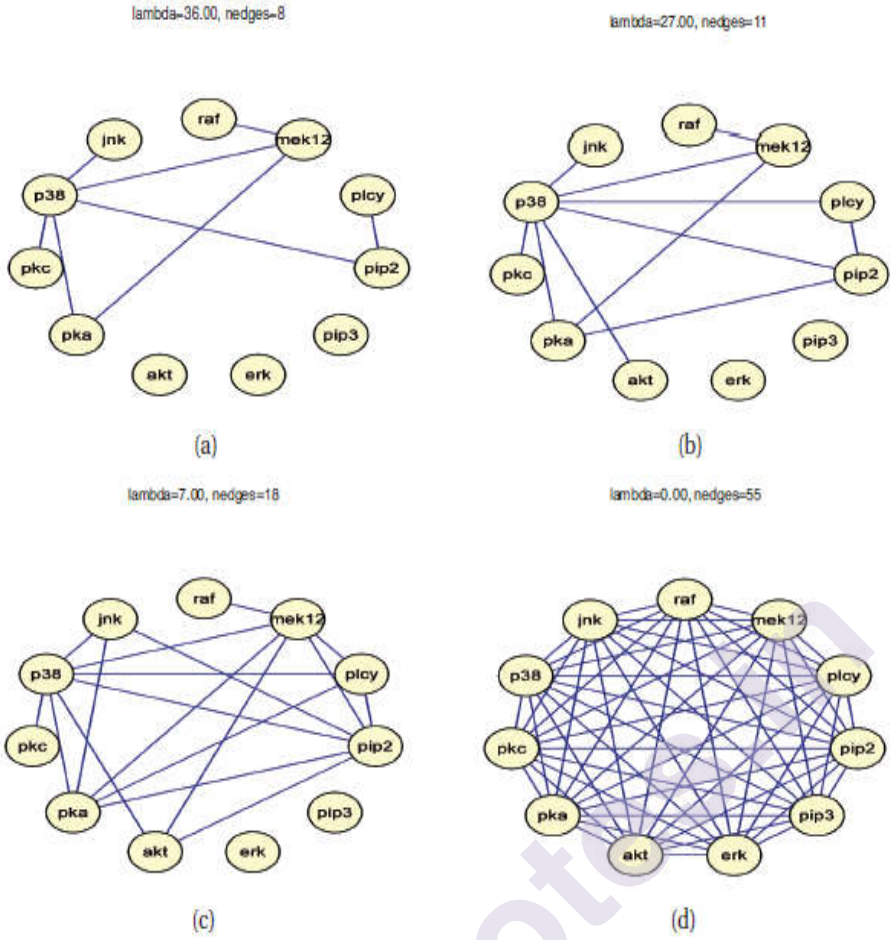


Figure 7.13 Sparse GGMs learned using graphical lasso applied to the flow cytometry data.

As an example, let us apply the method to the flow cytometry dataset. A discretized version of the data is shown in Figure 7.12(a) which is using original continuousdata. However, ignore the fact that the data was sampled under intervention.

InFigure 7.13, the graph structures sweep λ from 0 to a largevalue. These represent a range of plausible hypotheses about the connectivity of these proteins.

Advantage of the DAG : Easily model the interventional nature of the data,

Disadvantage: Cannot model the feedback loops that are known to exist in this biological pathway

7.6.3 Bayesian inference for GGM structure

Graphical lasso is reasonably fast, it only gives a point estimate of the structure.It is not model-selection consistent,meaning it cannot recoverthe true graph even as $N \rightarrow \infty$.

It would be preferable to integrate out the parameters, andperform posterior inference in the space of graphs, i.e., to compute p(G|D).

Extractsummaries of the posterior, such as posterior edge marginal, $p(G_{ij} = 1|D)$,

If the graph is decomposable, and if conjugate priors are used, compute the marginallikelihood in closed form.

The decomposable neighbors of a graph can be identifiable efficiently i.e., the set of legal edgeadditions and removals. However, the restriction to decomposable graphs is rather limiting if one's goal is knowledge discovery, since the number of decomposable graphs is much less than the number of generalundirected graphs

7.6.4 Handling non-Gaussian data using copulas

The graphical lasso and variants is inhertently limited to data that is jointly Gaussian, which isa rather severe restriction.

The method can be generalized to handle non-Gaussian,but still continuous, data in a fairly simple fashion. The basic idea is to estimate a set of Dunivariate monotonic transformations f_j, one per variable j, such that the resulting transformeddata is jointly Gaussian.

If this is possible, then data belongs to the nonparametricNormal distribution, or **nonparanormal** distribution. This is equivalent to thefamily of **Gaussian copulas**

7.7 LEARNING UNDIRECTED DISCRETE GRAPHICAL MODELS

The problem of learning the structure for UGMs with discrete variables is harder than theGaussian case, because computing the partition function $Z(\theta)$, which is needed for parameterestimation, has complexity comparable to computing the permanent of a matrix, which ingeneral is intractable.

But in the Gaussian case, computing Z onlyrequires computing a matrix determinant, which is at most $O(V^3)$.

Since stochastic local search is not tractable for general discrete UGMs, the possible alternative approaches can be tried.

(i) Graphical lasso for MRFs/CRFs

(ii) Thin junction trees

7.7.1 Graphical lasso for MRFs/CRFs

the graphical lasso idea can be extended to the discrete MRF and CRF case. However, nowthere is a set of parameters associated with each edge in the graph, so use the graphanalog of group lasso.

For example, consider a pairwise CRF with ternarynodes, and node and edge potentials given by

$$\psi_t(y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{v}_{t1}^T\mathbf{x} \\ \mathbf{v}_{t2}^T\mathbf{x} \\ \mathbf{v}_{t3}^T\mathbf{x} \end{pmatrix}, \quad \psi_{st}(y_s, y_t, \mathbf{x}) = \begin{pmatrix} \mathbf{w}_{t11}^T\mathbf{x} & \mathbf{w}_{st12}^T\mathbf{x} & \mathbf{w}_{st13}^T\mathbf{x} \\ \mathbf{w}_{st21}^T\mathbf{x} & \mathbf{w}_{st22}^T\mathbf{x} & \mathbf{w}_{st23}^T\mathbf{x} \\ \mathbf{w}_{st31}^T\mathbf{x} & \mathbf{w}_{st32}^T\mathbf{x} & \mathbf{w}_{st33}^T\mathbf{x} \end{pmatrix}$$

where assume x begins with a constant 1 term, to account for the offset. To learn sparse structure, minimize the following objective:

$$\begin{aligned} J &= -\sum_{i=1}^{N}\left[\sum_t \log\psi_t(y_{it}, \mathbf{x}_i, \mathbf{v}_t) + \sum_{s=1}^{V}\sum_{t=s+1}^{V}\log\psi_{st}(y_{is}, y_{it}, \mathbf{x}_i, \mathbf{w}_{st})\right] \\ &\quad + \lambda_1\sum_{s=1}^{V}\sum_{t=s+1}^{V}||\mathbf{w}_{st}||_p + \lambda_2\sum_{t=1}^{V}||\mathbf{v}_t||_2^2 \end{aligned}$$

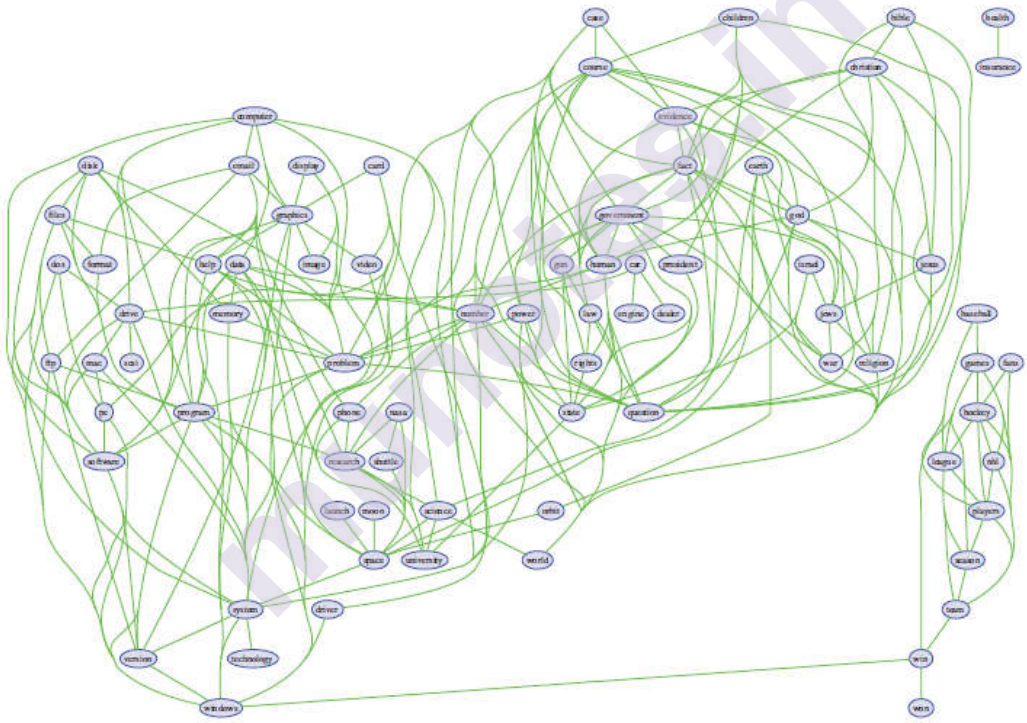


Figure 7.14 MRF estimated from the 20-newsgroup data using group ℓ1 regularization with λ = 256.

where ||wst||p is the p-norm; common choices are p = 2 or p = ∞,. This method is known as CRF structure learning

Although this objective is convex, it can be costly to evaluate (performinference to compute its gradient)

So use an optimizer that does not make too many calls to the objective functionor its gradient, such as the projected quasi-Newton method

In addition,we can use approximate inference, such as convex belief propagation to computean approximate objective and gradient more quickly. Another approach is to apply the grouplasso penalty to the pseudo-likelihood. This is much faster, sinceinference is no longer required Figure 7.14 shows the result ofapplying this procedure to the 20-newsgroup data, where y_{it} indicates the presence of word tin document i, and $x_i = 1$.

7.7.2 Thin junction trees

Learning "sparse" graphs, do not necessarily havelow treewidth.

For example, a D × D grid is sparse, but has treewidth O(D). This means thatthe models may be intractable to use for inference purposes, which defeats the learn graph structure in the first place.

There have been various attempts to learn graphical models with bounded treewidthalso knownas **thin junction trees**, but the exact problem in general is hard.

An alternative approach is to learn a model with low **circuit complexity** Such models may have high treewidth, but they exploit contextspecificindependence and determinism to enable fast exact inference.

7.8 SUMMARY

This chapter covers the topics about Graphical model structure learning which discusses about Structure learning for knowledge discovery, Relevance networks and Dependency networks. The Learning tree structures discusses about Directed or undirected tree, Chow-Liu algorithm for finding the ML tree structure, finding the MAP forest, Mixtures of trees. Learning DAG structure with latent variables discusses about Approximating the marginal likelihood for missing data, Structural EM, Discovering hidden variables and Structural equation models.Learning causal DAGs discusses about Causal interpretation of DAGs, Using causal DAGs to resolve Simpson's paradox, Learning causal DAG structures . Learning undirected Gaussian graphical models discusses about MLE for a GGM, Graphical lasso, Bayesian inference for GGM structure and Handling non-Gaussian data using copulas. Finally Learning undirected discrete graphical models discusses about Graphical lasso for MRFs/CRFs and Thin junction trees.

7.9 REFERENCES

1. Machine Learning A Probabilistic Perspective Kevin P. Murphy The MIT Press Cambridge, Massachusetts London, England

2. http://cs.nyu.edu/~roweis/data.html

8

DEEP LEARNING

Unit Structure :

8.0 OBJECTIVES

At the end of Chapter students will be able to:

- Discuss about the concepts and implementation of deep directed, undirected graphical models
- Describe the role of various kinds of Deep neural networksin deep learning
- Explain the applications of deep networks in the text,audio and video domain

8.1 INTRODUCTION

The human brain engages in multiple layers of processing, with each layer acquiring features or representations at progressively higher levels of abstraction. For instance, according to the conventional model of the visual cortex, the brain initially identifies edges, then moves on to patches, surfaces, objects, and so forth.

This observation has served as a source of inspiration for a recent trend in machine learning known as deep learning, as discussed on deeplearning.net and the references provided therein. Deep learning aims to emulate this hierarchical architecture within computers. Furthermore, this concept can be extended beyond visual problems, including applications in areas like speech and language.

Many of the models presented in this context follow a basic architecture consisting of two layers, where it can be denoted as either "z → y" for unsupervised latent variable models or "x → y" for supervised models.

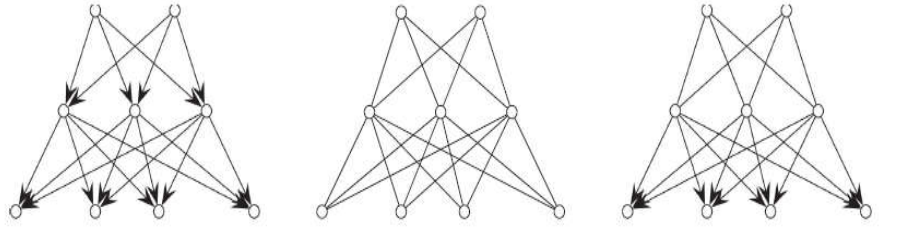
8.2 DEEP GENERATIVE MODELS

Issues in Deep Models:

- Having millions of parameters.
- Acquiring enough labeled data to train suchmodels is difficult
- This approach does not support scaling the complex scenes.For example, in simple settings, suchas hand-written character recognition, it is possible to generate lots of labeled data by makingmodified copies of a small manually labeled training set The unsupervisedlearning is used to overcome the problem of needing labeled training data.. The most natural way to perform this is to use generative models.

There are three different kinds of deep generative models such as

1. Directed
2. Undirected
3. Mixed.



(a) (b) (c)

Figure 8.1 Deep multi-layer graphical models.

(a) A directed model (b) An undirected model (c) A mixed directed-undirected model

8.2.1 Deep Directed Networks

A deep directedgraphical model is constructed as shown in Figure 8.1(a).

The bottom level contains the observed pixels (orwhatever the data is), and the remaining layers are hidden. 3 layers are taken into consideration fornotational simplicity.

The number and size of layers is usually chosen by hand, although onecan also use non-parametric Bayesian methods or boosting to infer the model structure.

These model forms are called as Deep Directed Networks or DDNs.

Sigmoid belief net: All the nodes arebinary, and all Conditional Probability Distributions (CPDs) are logistic functions.

Inthis case, the model defines the following joint distribution:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \prod_i \text{Ber}(v_i|\text{sigm}(\mathbf{h}_1^T \mathbf{w}_{0i})) \prod_j \text{Ber}(h_{1j}|\text{sigm}(\mathbf{h}_2^T \mathbf{w}_{1j})) \prod_k \text{Ber}(h_{2k}|\text{sigm}(\mathbf{h}_3^T \mathbf{w}_{2k})) \prod_l \text{Ber}(h_{3l}|w_{3l})$$

Disadvantage:

- Inference in these is intractable because the posterioron the hidden nodes is correlated due to explaining away.
  - To overcome this issue fast mean field approximations or MCMC inference can be used, but these may not be very accurate or can be quite slow respectively,
- .Slow inference also results in slow learning.

8.2.2 Deep Boltzmann machines

- Deep Boltzmann machine (DBM)Stacks a series of Restricted Boltzmann Machines(RBMs) on top of each other (Fig. 8.1 (b))
- It is a natural alternative to a directed model is to construct a deep undirected model.
- For 3 hiddenlayers, the model is defined as:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_{ij} v_i h_{1j} W_{1ij} + \sum_{jk} h_{1j} h_{2j} W_{2jk} + \sum_{kl} h_{2k} h_{3l} W_{3kl}\right)$$

(Ignore constant offset or bias terms)

Advantage

Performs efficient block Gibbs sampling, or block mean field compared to directed graph.

Disadvantage

- Training undirected models is more difficult, because of the partition function.
- Exact inference is intractable
- .Approximate inference can be slow

8.2.3 Deep Belief Networks (DBN)

- Uses a model that is partially directed and partially undirected.
- Construct a layered model which has directed arrows, except at thetop, where there is an undirected bipartite graph, as shown in Figure 28.1(c).
- Top two layers act as an associative memory and the remaining layers thengenerate the output.

For 3 hidden layers, the DBN model is defined as follows:

$$p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{v}|\boldsymbol{\theta}) = \prod_i \text{Ber}(v_i|\text{sigm}(\mathbf{h}_1^T \mathbf{w}_{1i}) \prod_j \text{Ber}(h_{1j}|\text{sigm}(\mathbf{h}_2^T \mathbf{w}_{2j}) \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_{kl} h_{2k} h_{3l} W_{3kl}\right)$$

Advantage:

The hidden states can be found in afast, bottom-up fashion.

To see why, suppose we only have two hidden layers, and that

$$\mathbf{W}_2 = \mathbf{W}_1^T,$$

so the second level weights are tied to the first level weights (Figure 8.2(a)).

This defines a model of the form p(h1, h2, v|W1).

One can show that the distribution

$p(\mathbf{h}_1, \mathbf{v}|\mathbf{W}_1) = \sum_{\mathbf{h}_2} p(\mathbf{h}_1, \mathbf{h}_2, \mathbf{v}|\mathbf{W}_1)$ has the form

$$p(\mathbf{h}_1, \mathbf{v}|\mathbf{W}_1) = \frac{1}{Z(\mathbf{W}_1)} \exp(\mathbf{v}^T \mathbf{W}_1 \mathbf{h}_1),$$

which is equivalent to an Restricted Boltzmann Machines(RBM).

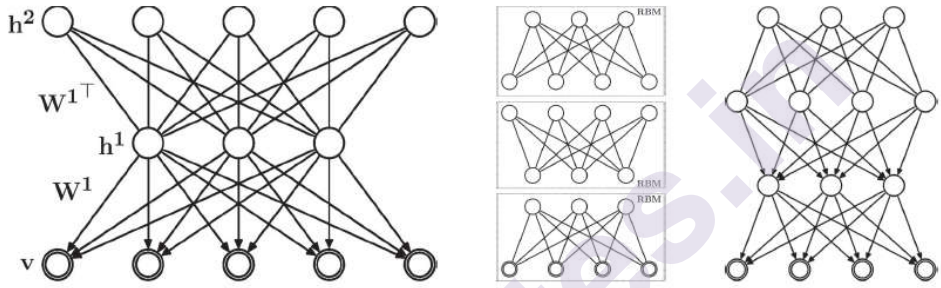
Since the DBN is equivalent to the RBM as far as p(h1, v|W1)is concerned, the posterior p(h1|v,W1) in the DBN exactly as in the RBM. Thisposterior is exact, even though it is fully factorized.

To get a factored posterior is if the prior p(h1|W1) is a complementaryprior.

This is a prior which, when multiplied by the likelihood p(v|h1), results in a perfectlyfactored posterior.

The top level RBM in a DBN acts as a complementary priorfor the bottom level directed sigmoidal likelihood function.

That top-down inference in a DBN is not tractable,so DBNs are usually only used in a feedforward manner.



(a) (b) (c)

Figure 8.2 (a) A DBN with two hidden layers and tied weights that is equivalent to an RBM(b) A stack of RBMs trained greedily (c) The corresponding DBN.

8.2.4 Greedy layer-wise learning of DBNs

Strategy for learning a DBN (The equivalence between DBNs and RBMs)

- FitanRBMtolearnW_1 using methods likeDeriving the gradient,Approximating the expectations,Contrastive divergence
- Unroll the RBM into a DBN with 2 hidden layers, as in Figure 8.2(a).
- "Freeze" thedirected weights W_1 and let W_2 be "untied" so it is no longer forced to be equal to WT_1 .
- There is a better prior for p(h1|W_1) by fitting a second RBM.
- The input data tothis new RBM is the activation of the hidden units E[h_1|v,W_1]
- Continue to addmore hidden layers until some stopping criterion is satisfied
- Construct the DBN fromthese RBMs, as illustrated in Figure 8.2(c).

This procedure always increases a lower bound theobserved data likelihood. This procedure might result in overfitting,

. The method can alsobe extended to train DBMs in a greedy way

After using the greedy layer-wise training strategy, it is standard to "fine tune" the weights,using a technique called backfitting.

Backfitting.(up-down procedure)

Used to "fine tune" the weights, after using the greedy layer-wise training strategy

- Perform brief Gibbs sampling in the top level RBM.
- Perform a CD updateof the RBM parameters.
- Finally, perform a downwards ancestral sampling pass (which is anapproximate sample from the posterior), and update the logistic CPD parameters using a smallgradient step.

8.3 DEEP NEURAL NETWORKS

DBNs are often only used in a feed-forward, or bottom-up, mode, they are effectivelyacting like neural networks. It is natural to dispense with the generative storyand try to fit deep neural networks directly using Deep multi-layer perceptrons, Deep auto-encoders and Stacked denoising auto-encoders.

Merits:

The resulting training methodsare often simpler to implement

Can be faster.

Limitation:

Performance with deep neural nets is sometimes not as good as withprobabilistic models One reason for this is that probabilistic models support top-down inference as well as bottom-up inference.

8.3.1 Deep Multi-layer Perceptrons

Many decision problems can be reduced to classification, e.g., predict which object (if any) is

present in an image patch, or predict which phoneme is present in a given acoustic featurevector. Such problems can be solved by creating a deep feedforward neural network (Figure 8.3.1) or multilayerperceptron (MLP), and then fitting the parameters using gradient descent (aka back-propagation).

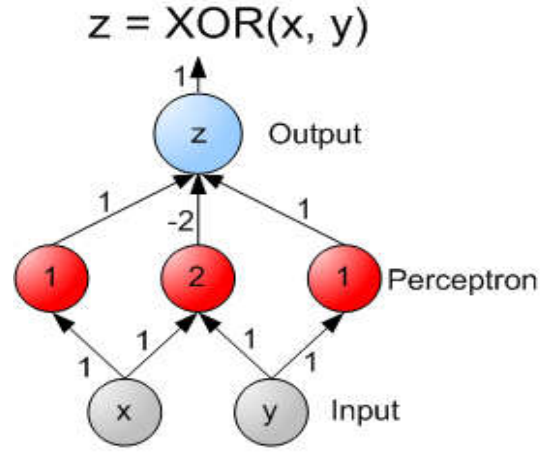


Figure 8.3.1 Feedforward Network

Limitations:

V**anishing gradient problem**: The gradient becomesweaker the further the process move away from the data;.

There can be large plateaus inthe error surface, which cause simple first-order gadient-based methods to get stuck

Generative pre-training

A way to initialize the parameters using unsupervised learning; The advantage of performing unsupervised learning first is that themodel is forced to model a high-dimensional response, namely the input feature vector, ratherthan just predicting a scalar response. This acts like a data-induced regularizer, and helpsbackpropagation find local minima with good generalization properties

8.3.2 Deep auto-encoders

An auto-encoder is a kind of unsupervised neural network that is used for dimensionalityreduction and feature discovery.

An auto-encoder is a feedforward neural networkthat is trained to predict the input itself.

To prevent the system from learning the trivial identity mapping, the hidden layer in the middle is usually constrained to be a narrow bottleneck.

The system can minimize the reconstruction error by ensuring the hidden units capture the most relevant aspects of the data.

Suppose the system has one hidden layer, so the model has the form v → h → v. Further, suppose all the functions are linear. In this case, the weights to the Khidden units will span the same subspace as the first K principal components of the data

More powerful representations can be learned by using deep auto-encoders. But training such models using back-propagation does not work well, because the gradient signalbecomes too small as it passes back through multiple layers and the learning algorithm oftengets stuck in poor local minima.

One solution to this problem is to greedily train a series of RBMs and to use these to initialize

an auto-encoder, as illustrated in Figure 8.3.2. The whole system can then be fine-tuned usingbackprop in the usual fashion. This works much better than trying to fit the deep auto-encoder directly starting with random weights.

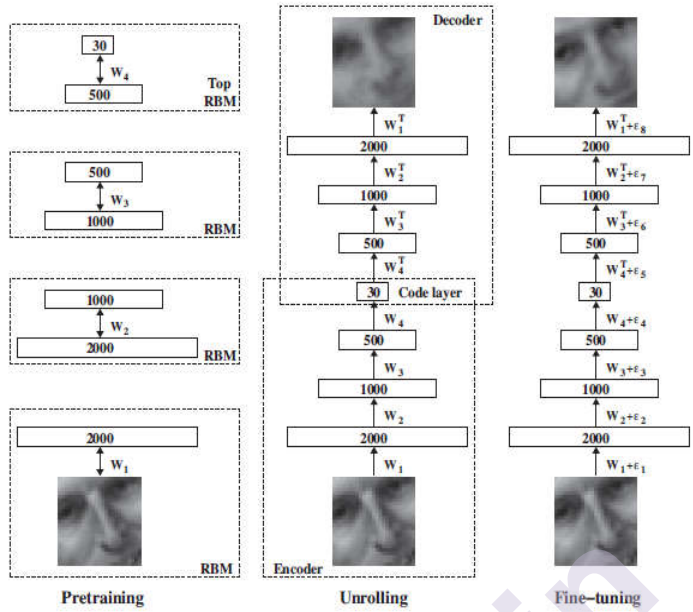


Figure 8.3.2 Training a deep autoencoder. (a) Greedily training some RBMs.

8.3.3 Stacked denoising auto-encoders

A common method for training an auto-encoder involves ensuring that the hidden layer has fewer neurons than the visible layer.

This precautionary measure prevents the model from merely learning to replicate its input data, effectively serving as an identity function.

The alternative strategies to prevent this trivial solution,

i. Enforce sparsity constraints on the activation of the hidden units, ensuring that only a limited number of neurons are active at any given time.

ii. Introducing noise to the input data, resulting in what is known as a denoisingautoencoder. For instance, some of the input values can be intentionally corrupted, such as setting them to zero, forcing the model to learn how to predict the missing or perturbed entries. This method can be demonstrated to be akin to a specific form of maximum likelihood training, referred to as score matching, when applied to a Restricted Boltzmann Machine (RBM).

iii. Stack these autoencoder models atop one another to create a deep stacked denoising auto-encoder. Such a deep architecture can be discriminatively fine-tuned, similar to a standard feedforward neural network, if desired.

iv.

8.4 APPLICATIONS OF DEEP NETWORKS

In this section the following applications of the models are discussed..

- Handwritten digit classification using DBNs
- Data visualization and feature discovery using deep auto-encoders
- Information retrieval using deep auto-encoders (semantic hashing)
- Learning audio features using 1d convolutional DBNs
- Learning image features using 2d convolutional DBNs

8.4.1 Handwritten digit classification using DBNs

DBN consisting of 3 hidden layers is shown in Figure 8.4.1(a).

The visible layer corresponds to binary images of handwritten digits from the MNIST data set.The top RBM is connected to a softmax layer with 10 units, representing the class label.

The first 2 hidden layers were trained in a greedy unsupervised fashion from 50,000 MNIST digits, using 30 epochs (passes over the data) and stochastic gradient descent, with the CD heuristic.

This process took “a few hours per layer”

The top layer was trained using as input the activations of the lower hidden layer, as well as the class labels. The corresponding generative model had a test error of about 2.5%. The network weights were then carefully fine-tuned on all 60,000 training images using the up-down procedure.

This process took “about a week”. The model can be used to classify by performing a deterministic bottom-up pass, and then computing the free energy for the top-level RBM for each possible class label.

The final error on the test set was about 1.25%. The misclassified examples are shown in Figure 8.4.1(b).This was the best error rate of any method on the permutation-invariant version of MNIST at that time.

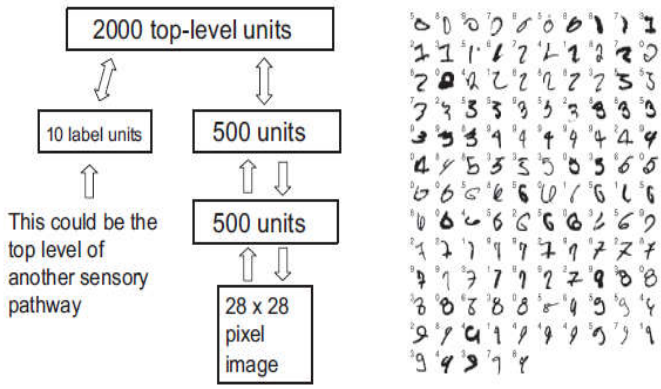


Figure 8.4.1 (a) A DBN architecture for classifying MNIST digits.
(b) Errors made by the DBN on the 10,000 test cases of MNIST.

8.4.2 Data visualization and feature discovery using deep auto-encoders

Deep autoencoders can learn informative features from raw data. Such features are often used as input to standard supervised learning methods.

Consider fitting a deep auto-encoder with a 2D hidden bottleneck to sometext data. The results are shown in Figure 8.4.2. On the left Figure 8.4.2 (a) the 2D embedding produced by LSA is depicted and on the right Figure 8.4.2 (b), the 2D embedding produced by the auto-encoder is shown.

The results show that the low-dimensional representation created by the auto-encoder has captured a lot of the meaning of the documents, even though class labels were not used.

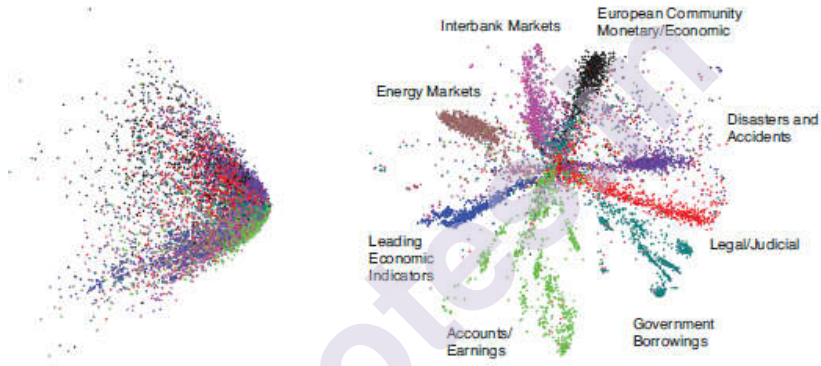


Figure 8.4.2 Results : 2D visualization of some bag of words data from the Reuters RCV1-v2 corpus. (a) Using LSA. (b) Using a deep auto-encoder.

8.4.3 Information retrieval using deep auto-encoders (semantic hashing)

Though the success of RBMs for information retrieval is achieved, the deep models perform even better. The performance of deep model is shown in Figure 8.4.3.

Use a binary low-dimensional representation in the middle layerof the deep auto-encoder.

Thisenables very fast retrieval of related documents.

Semantic hashing: The binary representation of

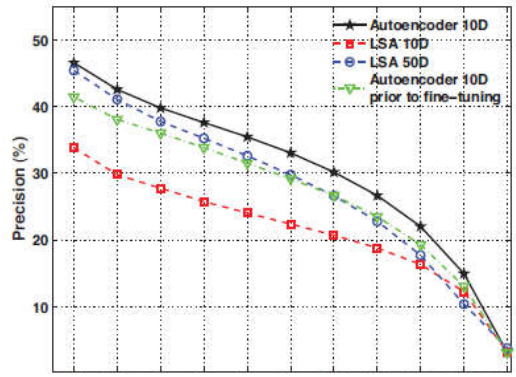


Figure 8.4.3. Precision-recall curves for document retrieval in the Reuters RCV1-v2

semantically similar documents will be close in Hamming distance. For a 20-bit code, precompute the binary representation for all the documents, and then create a hash-tablemapping codewords to documents.

For the 402,207 test documents in Reuters RCV1-v2, this translates to roughly 0.4 documents per entry listed in the table.

During the testing phase, the procedure involves calculating the codeword associated with the query and subsequently retrieving the relevant documents with constant-time efficiency by referencing the corresponding memory address.

To identify additional related documents, the approach entails computing all codewords that are within a Hamming distance, such as 4 from the original query. This process results in retrieving approximately $6196 \times 0.4 \approx 2500$ documents. The key point to emphasize is that the total time required for this operation remains unaffected by the size of the corpus.

In contrast, other methods for rapid document retrieval, like inverted indices, rely on the notion that individual words carry significant information, allowing for the straightforward intersection of documents containing each specific word. It's worth noting that applying inverted indexing techniques to real-valued data is a challenging endeavor.

8.4.4 Learning audio features using 1D convolutional DBNs

To employ Deep Belief Networks (DBNs) for time series data of infinite duration, it becomes imperative to implement a mechanism for parameter sharing. One approach to achieve this is by employing convolutional DBNs, which employ convolutional Restricted Boltzmann Machines (RBMs) as their fundamental building blocks. These models represent a generative counterpart of convolutional neural networks.The basic idea is illustrated in Figure 8.4..

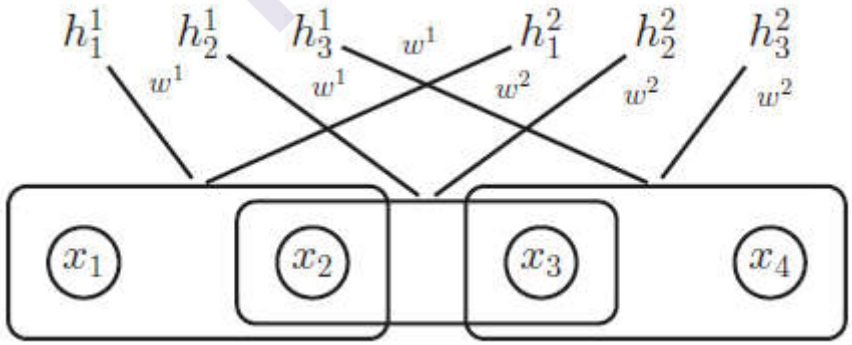


Figure 8.4.4 A small 1d convolutional RBM with two groups of hidden units, each associated with a filter of size 2.

h_1^1andh_1^2 are two diferent “views” of the data in the first window, (x_1, x_2).

The first view is computed using the filter W^1, the second view using filter W^2.

Similarly$h_1^2$and h_2^2 are the views of the data in the second window (x_2, x_3)., computed using W^1 and W^2 respectively

- The hidden activation vector for each group is computed by convolving the input vector withthat group's filter (weight vector or matrix).(Each node within a hidden groupis a weighted combination of a subset of the inputs.)
- Compute the activation of all thehidden nodes by "sliding" this weight vector over the input.
- Each group has its own filter, corresponding to its own pattern detector.
- One task's "signal" becomesother task's "noise", so donot "throw away" any irrelevant information
- For binary 1Dsignal, the full conditionals in a convolutionalRBM can be defined as:

$$p(h_t^k = 1|\mathbf{v}) = \text{sigm}((\mathbf{w}^k \otimes \mathbf{v})_t + b_t)$$
$$p(v_s = 1|\mathbf{h}) = \text{sigm}(\sum_k (\mathbf{w}^k \otimes \mathbf{h}^k)_s + c_s)$$

whereW^k is the weight vector for group k,

b_t and c_s are bias terms,

a⊗ b represents theconvolution of vectors **a** and **b**.

Integrate both a convolutional layer and a max pooling layer into the architecture, where theycalculate the local maximum within the filtered response. This introduces a degree of translation invariance and, concurrently, reduces the dimensions of the higher layers, leading to a significant acceleration in computation.

Defining this for a neural network is simple, but defining this in a way which allows forinformation flow backwards as well as forwards is a bit more involved.

The basic idea is similarto a noisy-OR CPD where a probabilistic relationship between the maxnode and the parts it is maxing over can be defined.

When the inputconsists of speech signals, the method recovers a representation that is similar to phonemes.To get a good performance result, standard features such as MFCCtechniques are used to apply music classification and speaker identification.

8.4.5 Learning image features using 2D convolutional DBNs

A convolutional DBN is extended from 1D to 2D in a straightforward way. The extended RBM is illustrated in Figure 8.4.5. The results of a 3 layer system trained on cars, motorbikes, faces and airplanesvisual objects having four classes to represent every ones properties, actions and behaviours from the Caltech 101 dataset. The same is shown in Figure 8.4.6.

The result is shown only for layers 2 and 3, because layer 1 learns Gabor-like filters that are very similar to those learned by sparse coding.

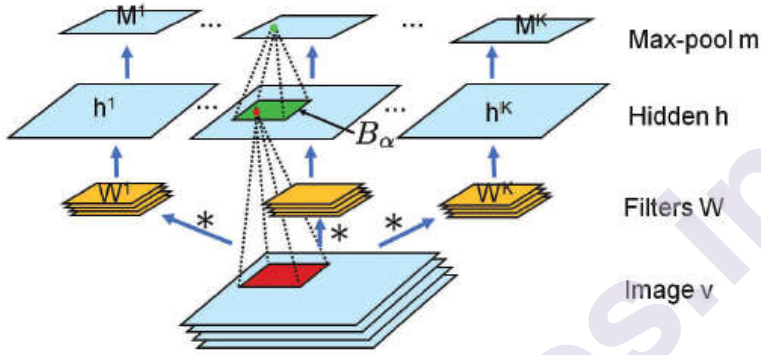


Figure 8.4.5 A 2d convolutional RBM with max-pooling layers.

In the figure 8.4.5, the input signal consist of a stack of 2D images (e.g., color planes). Each input layer is passed through a different set of filters. Each hidden unit is obtained by convolving with the appropriate filter, and then summing over the input planes. The final layer is obtained by computing the local maximum within a small window. As a result,

- Layer 2 haslearned some generic visual parts, shared amongst object classes,
- Layer 3 have learned filters like grandmother cells, that are specific to individual object classes,and in some cases, to individual objects.

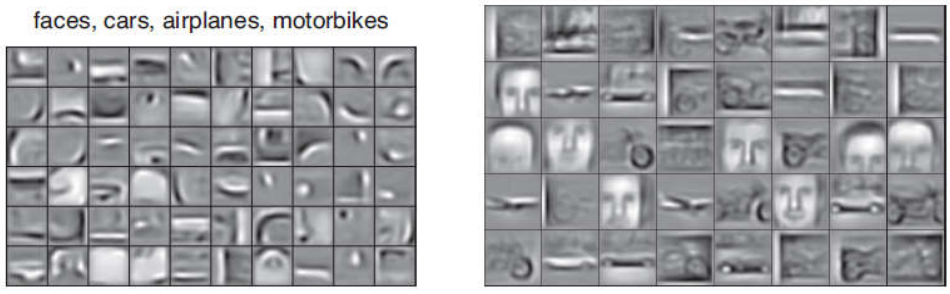


Figure 8.4.6 Visualization of the filters learned by a convolutional DBN in layers two and three

7. Source: http://research.microsoft.com/en-us/news/features/speechrecognition-082911.aspx.

8.5 SUMMARY

Until now, our discussions have centered on models inspired by the brain's low-level processing mechanisms. These models have proven effective in generating valuable features for straightforward classification tasks. However, can this purely bottom-up approach effectively tackle more complex challenges, such as scene interpretation or natural language comprehension?

To provide some context, let's consider the Deep Belief Network (DBN) designed for handwritten digit classification in Figure 28.4(a). This network comprises roughly 1.6 million free parameters (calculated as 28×28×500+500×500+510×2000 = 1,662,000). While this may seem substantial, it pales in comparison to the number of neurons present in the human brain.

The Section 8.2 described generative deep learning models in which the concepts under Deep directed networks, Deep Boltzmann machines, Deep belief networks and Greedy layer-wise learning of DBNs are explained clearly. In addition the concept of RBM is compared with remaining concepts.

Next to these the Deep multi-layer perceptrons, Deep auto-encoders and Stacked denoising auto-encoders are explained to show how the layers are performing their learning part.

Finally the applications of Applications of deep networks such as Handwritten digit classification using DBNs, Data visualization and feature discovery using deep auto-encoders, Information retrieval using deep auto-encoders, Learning audio features using 1D convolutional DBNs andLearning image features using 2D convolutional DBNs are described to explain how the deep learning concept is applied in the particular domain/scenario.

8.6 REFERENCES

1. Machine Learning A Probabilistic Perspective, Kevin P. Murphy, The MIT Press Cambridge, Massachusetts London, England

2. https://www.geeksforgeeks.org/

3. https://en.wikipedia.org/wiki/Deep_belief_network

❖❖❖❖