Module 4: Bayesian Learning: Introduction, Bayes theorem, Bayes theorem and concept learning, ML and LS error hypothesis, ML for predicting, MDL principle, Bayes optimal classifier, Gibbs algorithm, Navie Bayes classifier, BBN, EM Algorithm

4. BAYESIAN LEARNING

Bayesian reasoning provides a probabilistic approach to inference.

4.1 INTRODUCTION

Bayesian learning methods are relevant to our study of machine learning for two different reasons. First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems.

The second reason that Bayesian methods are important to our study of ma-chine learning is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.

Features of Bayesian learning methods include:

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct.
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions.
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.

4.2 BAYES THEOREM

In machine learning we are often interested in determining the best hypothesis from some space H, given the observed training data D. One way to specify what we mean by the **best** hypothesis is to say that we demand the **most probable** hypothesis, given the data **D** plus any initial knowledge about the prior probabilities of the various hypotheses in H.

Bayes theorem provides a direct method for calculating such probabilities. Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

We shall write **P (h)** to denote the initial probability that hypothesis **h** holds, before we have observed the training data. **P(h)** is often called the **priorprobability** of **h** . We will write **P(D)** to denote the prior probability that training data D will be observed. We will write **P(D/h)** to denote the probability of observing data **D** given some world in which hypothesis **h** holds. We are interested in the probability **P(h/D)** that **h** holds given the observed training data **D. P(h/D)** is called the **posteriorprobability** of **h .**

Bayes theorem provides a way to calculate the posterior probability **P(h/D),** from the prior probability **P(h),** together with **P(D)** and **P (D / h) .**

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis **h E** H given the observed data **D** (or at least one of the maximally probable if there are several). Any such maximally probable hypothesis is called a **maximum** a **posteriori** (MAP) hypothesis. We can determine the MAP hypotheses by using Bayes theorem to calculate the posterior probability of each candidate hypothesis. More precisely, we will say that **MAP** is a MAP hypothesis provided

$$\begin{aligned} h_{MAP} &\equiv \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h) \end{aligned} \qquad (6.2)$$

Notice in the final step above we dropped the term **P (D)** because it is a constant independent of **h .**

In some cases, we will assume that every hypothesis in H is equally probable a priori **($P(h_i)$** = **$P(h_j)$** for all **h_i** and **h_j** in H). In this case we can further simplify Equation (6.2) and need only to consider the term **P(D/h)** to find the most probable hypothesis. **P(D/h)** is often called the **likelihood** of the data **D** given **h,** and any hypothesis that maximizes **P(D/h)** is called a **maximum likelihood** (ML) hypothesis, **h_{ML} .**

$$h_{ML} \equiv \underset{h \in H}{\operatorname{argmax}} P(D|h) \qquad (6.3)$$

4.2.1An Example

To illustrate Bayes rule, consider a medical diagnosis problem in which there are two alternative hypothesis: (1) that the patient have a particular form of cancer and (2) that the patient does not.

We have prior knowledge that over the entire population of people only .008 have this disease. The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result. The above situation can be summarized by the following probabilities:

$$\begin{aligned} P(cancer) &= .008, & P(\neg cancer) &= .992 \\ P(\oplus|cancer) &= .98, & P(\ominus|cancer) &= .02 \\ P(\oplus|\neg cancer) &= .03, & P(\ominus|\neg cancer) &= .97 \end{aligned}$$

Suppose we now observe a new patient for whom the lab test returns a positive result. The maximum a posterior hypothesis can be found using Equation (6.2):

$$P(\oplus|cancer)\,P(cancer) = (.98).008 = .0078$$
$$P(\oplus|\neg cancer)\,P(\neg cancer) = (.03).992 = .0298$$

Thus, $h_{MAP} = \neg cancer.$

4.3 BAYES THEOREM AND CONCEPT LEARNING

Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data.

Brute-Force Bayes Concept Learning

Assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept **c** : X → **{0,1}.** As usual, we assume that the learner is given some sequence of training examples **((x_1, d_1). . . (x_m, d_m))** where x_i is some instance from X and where d_i is the target value of x_i (i.e., d_i = **c(x_i).** To simplify the discussion, we assume the sequence of instances **(x_1 . . .x_m)** is held fixed, so that the training data D can be written simply as the sequence of target values D = **(d_1 . . .d_m)**

We can design a straightforward concept learning algorithm to output the maximum a posterior hypothesis, based on Bayes theorem, as follows:

4.3.1 Brute-Force Map Learning Algorithm

1. For each hypothesis **h** in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

In order to specify a learning problem for the **BRUTE-FORCE MAP LEARNING** algorithm we must specify what values are to be used for **P(h)** and for **P(D/h).** Let us choose them to be consistent with the following assumptions:

- The training data **D** is noise free (i.e., d_i = **c(x_i)) .**
- The target concept **c** is contained in the hypothesis space H
- We have no a priori reason to believe that any hypothesis is more probable than any other.

Then we should choose,

$$P(h) = \frac{1}{|H|} \quad \text{for all } h \text{ in } H$$

P(D\h) is the probability of observing the target values D = **(d_1. . . d_m)** for the fixed set of instances **(x_1. . . x_m),** given a world in which hypothesis **h** holds. Since we assume noise-free training data, the probability of observing classification **d_i** given **h** is just **1** if **d_i = h(xi)** and 0 if **di ≠ h(x_i) .**

Therefore,

$$P(D|h) = \begin{cases} 1 \text{ if } d_i = h(x_i) \text{ for all } d_i \text{ in } D \\ 0 \text{ otherwise} \end{cases} \qquad (6.4)$$

In other words, the probability of data **D** given hypothesis **h** is 1 if **D** is consistent with **h,** and 0 otherwise.

Given these choices for **P(h)** and for **P(D\h)** we now have a fully-defined problem for the above **BRUTE-FORCEMAP LEARNING** algorithm. **Let** us consider the first step of this algorithm, which uses Bayes theorem to compute the posterior probability **P(h/D)** of each hypothesis **h** given the observed training data **D .**

Recalling Bayes theorem, we have

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

First consider the case where h is inconsistent with the training data D. Since Equation (6.4) defines P(D/h) to be 0 when h is inconsistent with D, we have

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0 \text{ if } h \text{ is inconsistent with } D$$

The posterior probability of a hypothesis inconsistent with D is zero.

Now consider the case where h is consistent with D. Since Equation (6.4) defines P(D\h) to be 1 when h is consistent with D, we have

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)}$$

$$= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}}$$

$$= \frac{1}{|VS_{H,D}|} \text{ if } h \text{ is consistent with } D$$

where $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D.

be one and because the number of hypotheses from H consistent with D is by definition $|VS_{H,D}|$.

$$P(D) = \sum_{h_i \in H} P(D|h_i) P(h_i)$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|}$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|}$$

$$= \frac{|VS_{H,D}|}{|H|}$$

To summarize, Bayes theorem implies that the posterior probability P(h/D) under our assumed P(h) and P(D/h) is

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases} \qquad (6.5)$$

Every consistent hypothesis is, therefore a MAP hypothesis.

4.3.2 MAP Hypotheses and Consistent Learners

A learning algorithm is a consistent learner provided it outputs a hypothesis that commits zero errors over the training examples. We can conclude that every consistent learner outputs a MAP hypothesis, if **we** assume a uniform prior probability distribution over H and if we assume deterministic, noise free training data.

For example, the concept learning algorithm FIND-S outputs a consistent hypothesis, we know that it will output a MAP hypothesis under the probability distributions P(h) and P(D/h).

Because FIND-S outputs a **maximally specific** hypothesis from the version space, its output hypothesis will be a MAP hypothesis. Suppose ***H*** is any probability distribution **P(h)**over that assigns **$P(h_1)$** >=**$P(h_2)$** if **h_1** is more specific than **h_2.** Then it can be shown that FIND-S outputs a MAP hypothesis assuming the prior distribution ***H*** and the same distribution **P(D/h)**.

The Bayesian framework allows one way to characterize the behaviour of learning algorithms (e.g., FIND-S), even when the learning algorithm does not explicitly manipulate probabilities. By identifying probability distributions **P(h)** and **P(D/h)** under which the algorithm outputs optimal (i.e., MAP) hypotheses, we can characterize the implicit assumptions under which this algorithm behaves optimally.

4.4 MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

Consider the problem of learning a continuous-valued target function. A straight forward Bayesian analysis will show that **under certain assumptions any learning algorithm that minimizes the**

squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis.

Consider the following problem setting. Learner L considers instance spaces X and a hypothesis space H consisting of some class of real-valued functions defined over X. The problem faced by L is to learn an unknown target function f : X → R drawn from H. A set of **m** training examples is provided, where the target value of each example is corrupted by random noise. Each training example is a pair of the form **(x_i, d_i)** where **d_i** = f **(x_i)**+ **e_i.** Here f **(x_i)** is the noise-free value of the target function and **e_i** is a random variable representing the noise. The task of the learner is to output a maximum likelihood hypothesis, or, equivalently, a MAP hypothesis assuming all hypotheses are equally probable a priori.

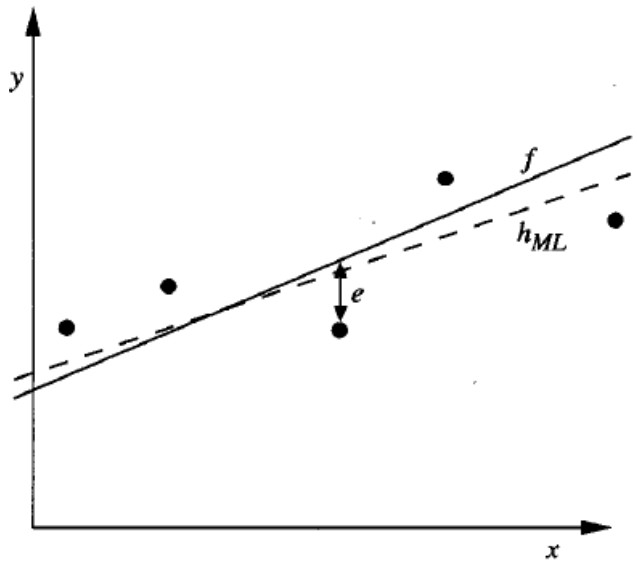

FIGURE 6.2
Learning a real-valued function. The target function f corresponds to the solid line. The training examples $\langle x_i, d_i \rangle$ are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$. The dashed line corresponds to the linear function that minimizes the sum of squared errors. Therefore, it is the maximum likelihood hypothesis h_{ML}, given these five training examples.

A simple example of such a problem is learning a linear function, though our analysis applies to learning arbitrary real-valued functions. Figure 6.2 illustrates a linear target function f depicted by the solid line, and a set of noisy training examples of this target function. The dashed line corresponds to the hypothesis **h_{ML}** with least-squared training error, hence the maximum likelihood hypothesis.

Before showing why a hypothesis that minimizes the sum of squared errors is a maximum likelihood hypothesis, consider two basic concepts from probability theory: probability densities and Normal distributions. The probability density for continuous variables such as e and require that the integral of this probability density over all possible values be one. We will use lower case p to refer to the probability density function. The probability density **p(x_0)** is the limit as ε goes to zero, of 1/ε times the probability that **x** will take on a value in the interval **[x_o, x_o** + ε**].**

Probability density function:

$$p(x_0) \equiv \lim_{\epsilon \to 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

Second, we stated that the random noise variable e is generated by a Normal probability distribution.

The maximum likelihood hypothesis but using lower case p to refer to the probability density

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\, p(D|h)$$

we assume a fixed set of training instances $(x_1 \ldots x_m)$ and therefore consider the data D to be the corresponding sequence of target values $D=(d_1 \ldots d_m)$. Here $d_i = f(x_i) + e_i$. We can write **P(D/h)** as the product of the various $p(d_i/h)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} p(d_i|h)$$

$p(d_i/h)$ can **be** written as a Normal distribution with variance σ^2 and mean $\mu = f(x_i)$. Because we are writing the expression for the probability of d_i given that **h** is the correct description of the target function f, we will also substitute $p = f(x_i) = h(x_i)$, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-\mu)^2}$$

$$= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-h(x_i))^2}$$

Rather than maximizing the above complicated expression we shall choose to maximize its (less complicated) logarithm. Therefore maximizing **ln p** also maximizes **p.**

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of **h,** and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of **h.**

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Thus, Equation (6.6) shows that the maximum likelihood hypothesis $\mathbf{h_{ML}}$ **is** the one that minimizes the sum of the squared errors between the observed training values $\mathbf{d_i}$ and the hypothesis predictions $\mathbf{h(x_i)}$ **.**

4.5 Maximum Likelihood Hypotheses for Predicting Probabilities

Consider the setting in which we wish to learn a probabilistic function **f** : X →{0, 1}, which has two discrete output values. For example, the instance space X might represent medical patients in terms of their symptoms, and the target function **f** (x) might be 1 if the patient survives the disease and 0 if not.

Given this problem setting, we might wish to learn a neural network whose output is the **probability** that **f** (x) = 1. In other words, we seek to learn the target function, **f** ': X → [0, 1], such that **f** '(x) = P(**f** (x) = 1). In the above medical patient example, if x is one of those indistinguishable patients of which 92% survive, then f'(x) = 0.92 whereas the probabilistic function **f** (x) will be equal to 1 in 92% of cases and equal to 0 in the remaining 8%.

The criterion that we should optimize in order to find a maximum likelihood hypothesis for **f'** is first obtain an expression for P(D/h). Let us assume the training data D is of the form D = {(x_1, d_1) . . . (x_m, d_m)},where d_i is the observed 0 or 1 value for f (x_i).

we can write P(D/h) as

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) \qquad (6.7)$$

It is reasonable to assume that the probability of encountering any particular instance x_i is independent of the hypothesis h. When **x** is independent of **h** we can rewrite the above expression as,

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) = \prod_{i=1}^{m} P(d_i|h, x_i)P(x_i) \qquad (6.8)$$

Now what is the probability $\mathbf{P(d_i/h, xi)}$ of observing $\mathbf{d_i = 1}$ for a single instance **xi,**

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \qquad (6.9)$$

In order to substitute this into the Equation **(6.8)** for **P(D/h),** let us first re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} \qquad (6.10)$$

We can use Equation **(6.10)** to substitute for $\mathbf{P(d_i/h, x_i)}$ in Equation **(6.8)** to obtain

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i) \qquad (6.11)$$

Now we write an expression for the maximum likelihood hypothesis

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of **h,** so it can be dropped

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \qquad (6.12)$$

we will find it easier to work with the log of the likelihood, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \qquad (6.13)$$

And negation of this term is referred as cross entropy.

4.5.1Gradient Search to Maximize Likelihood in a Neural Net

Let us use G(h, D) to denote the above(6.13) quantity. We derive a weight-training rule for neural network learning that seeks to maximize G(h, D) using gradient ascent.

The gradient of G(h, D) is given by the vector of partial derivatives of G(h, D) with respect to the various network weights that define the hypothesis h represented by the learned network . In this case, the partial derivative of G(h, D) with respect to weight w_{jk} from input k to unit j is

$$\begin{aligned}
\frac{\partial G(h, D)}{\partial w_{jk}} &= \sum_{i=1}^{m} \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{h(x_i)(1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}} \qquad (6.14)
\end{aligned}$$

To keep our analysis simple, suppose our neural network is constructed from a single layer of sigmoid units. In this case we have.

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i)x_{ijk} = h(x_i)(1 - h(x_i))x_{ijk}$$

where x_{ijk} is the k_{th} input to unit j for the i_{th} training example, and $\sigma'(x)$ is the derivative of the sigmoid squashing function. Finally, substituting this expression into Equation (6.14), we obtain a simple expression for the derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk}$$

Because we seek to maximize rather than minimize P(D/h), we perform gradient ascent rather than gradient descent search. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight-update rule and where **η** is a small positive constant that determines the step size of the gradient ascent search.

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk} \qquad (6.15)$$

4.6 MINIMUM DESCRIPTION LENGTH PRINCIPLE

The Minimum Description Length principle is motivated by interpreting the definition of $\mathbf{h_{MAp}}$ the light of basic concepts from information theory. Consider again the familiar definition of h_{MAP}.

$$h_{MAP} = \underset{h \in H}{\text{argmax}}\ P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$

$$h_{MAP} = \underset{h \in H}{\text{argmax}}\ \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{\text{argmin}}\ -\log_2 P(D|h) - \log_2 P(h) \qquad (6.16)$$

Equation (6.16) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data.

[**Consider the problem of designing a code to transmit messages drawn at random, where the probability of encountering message **i** is $\mathbf{p_i}$. We are interested in the code that minimizes the expected number of bits we must transmit in order to encode a message drawn at random. To minimize the expected code length we should assign shorter codes to messages that are more probable. Shannon and Weaver (1949) showed that the optimal code (i.e., the code that minimizes the expected message length) assigns $-\log_2 p_i$ bits to encode message **i.****]

We will refer to the number of bits required to encode message **i** using code **C** as the **description length of message i with respect to C,** which we denote by $L_C(i)$.

Let us interpret Equation (6.16) in light of the above result from coding theory.

- $-\log_2 P(h)$ is the description length of **h** under the optimal encoding for the hypothesis space H. $L_{C_H}(h) = \log_2 P(h)$, where C_H is the optimal code for hypothesis space H.
- $-\log_2 P(D/h)$ is the description length of the training data **D** given hypothesis **h,** under its optimal encoding. $L_{C(D/h)} = -\log_2 P(D/h)$, where $C_{D/h}$ is the optimal code for describing data **D** assuming that both the sender and receiver know the hypothesis **h .**

Therefore we can rewrite Equation (6.16) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

$$h_{MAP} = \underset{h}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

where C_H and $C_{D\backslash h}$ are the optimal encodings for H and for **D** given h, respectively.

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths. Assuming we use the codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis, we can state the MDL principle as

Minimum Description Length principle: Choose $\mathbf{h_{MDL}}$ where

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h) \qquad (6.17)$$

4.7 BAYES OPTIMAL CLASSIFIER

Consider a hypothesis space containing three hypotheses, hl, h2, and h3. Suppose that the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus, hl is the MAP hypothesis. Suppose a new instance x is encountered, which is classified positive by hl, but negative by h2 and h3. Taking all hypotheses into account, the probability that x is positive is .4 (the probability associated with hi), and the probability that it is negative is therefore .6. The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis.

In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities. If the possible classification of the new example can take on any value v_j from some set V, then the probability $P(v_j|D)$ that the correct classification for the new instance is v_j, is just

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

The optimal classification of the new instance is the value v,, for which P (v; 1 D) is maximum.

Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

To illustrate in terms of the above example, the set of possible classifications of the new instance is $V = \{\oplus, \ominus\}$, and

$$P(h_1|D) = .4, \quad P(\ominus|h_1) = 0, \quad P(\oplus|h_1) = 1$$
$$P(h_2|D) = .3, \quad P(\ominus|h_2) = 1, \quad P(\oplus|h_2) = 0$$
$$P(h_3|D) = .3, \quad P(\ominus|h_3) = 1, \quad P(\oplus|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = .6$$

and

$$\underset{v_j \in \{\oplus, \ominus\}}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Any system that classifies new instances according to above Equation is called a Bayes optimal classifier, or Bayes optimal learner. No other classification method using the same hypothesis space and same prior knowledge can outperform this method on average. This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

For example, in learning Boolean concepts using version spaces as in the earlier section, the Bayes optimal classification of a new instance is obtained by taking a weighted vote among all members of the version space, with each candidate hypothesis weighted by its posterior probability.

4.8 GIBBS ALGORITHM

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply. The expense is due to the fact that it computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance.

An alternative, less optimal method is the Gibbs algorithm defined as follows:

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H.
2. Use h to predict the classification of the next instance x.

Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution. Surprisingly, it can be shown that under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier. More precisely, the expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner. Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier.

This result has an interesting implication for the concept learning problem described earlier. In particular, it implies that if the learner assumes a uniform prior over H, and if target concepts are in fact drawn from such a distribution when presented to the learner, then classifying the next instance according to a hypothesis drawn at random from the current version space (according to a uniform distribution), will have expected error at most twice that of the Bayes optimal classifier. Again, we have an example where a Bayesian analysis of a non-Bayesian algorithm yields insight into the performance of that algorithm.

4.9 NAIVE BAYES CLASSIFIER

One highly practical Bayesian learning method is the naive Bayes learner, often called the **naive Bayes classifier.**

The naive Bayes classifier applies to learning tasks where each instance **x** is described by a conjunction of attribute values and where the target function f **(x)** can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values **(a_1 ,a_2 . . . a_n).** The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, V_{MAP}, given the attribute values **(a_1,a_2 . . . a_n)** that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \qquad (6.19)$$

It is easy to estimate each of the **$P(v_j)$** simply by counting the frequency with which each target value **v_j** occurs in the training data. However, estimating the different **$P(a_1 , a_2 . . . a_n|v_j)$** terms in this fashion is not feasible

The assumption is that given the target value of the instance, the probability of observing the conjunction **$a_1 , a_2 . . . a_n$** is just the product of the probabilities for the individual attributes:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j).$$

Substituting this into Equation **(6.19),** we have the approach used by the naive Bayes classifier.

Naive Bayes classifier:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j) \qquad (6.20)$$

where **V_{NB}** denotes the target value output by the naive Bayes classifier.

4.9.1 An Illustrative Example

We use the naive Bayes classifier and the PlayTennis training data (from table 3.2) to classify the

following novel instance:

(Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong)

Our task is to predict the target value (yes or no) of the target concept PlayTennis for this new instance. Instantiating Equation (6.20) to fit the current task, the target value V_{NB} is given by

$$v_{NB} = \underset{v_j \in \{yes,no\}}{\text{argmax}} P(v_j) \prod_i P(a_i|v_j)$$

$$= \underset{v_j \in \{yes,no\}}{\text{argmax}} P(v_j) \quad P(Outlook = sunny|v_j)P(Temperature = cool|v_j)$$

$$P(Humidity = high|v_j)P(Wind = strong|v_j) \quad (6.21)$$

First, the probabilities of the different target values can easily be estimated based on their frequencies over the **14** training examples

P(P1ayTennis = yes) = 9/14 = .64

P(P1ayTennis = no) = 5/14 = .36

Similarly, we can estimate the conditional probabilities. For example, those for Wind = strong are

P(Wind = strong/PlayTennis = yes) = 3/9 = .33

P(Wind = strong/PlayTennis = no) = 3/5 = .60

Using these probability estimates and similar estimates for the remaining attribute values, we calculate $\mathbf{V_{NB}}$ according to Equation **(6 .21)** as follows,

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$

$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Thus, the naive Bayes classifier assigns the target value **PlayTennis** = **no** to this new instance, based on the probability estimates learned from the training data.

4.9.1.1 Estimating Probabilities

We estimated **P(Wind** = **strong/Play Tennis** = **no)** by the fraction n_c/n where n=5 is the total number of training examples for which **PlayTennis** = **no,** and $\mathbf{n_c}$ = 3 is the number of these for which **Wind** = **strong.**

While this observed fraction provides a good estimate of the probability in many cases, it provides poor estimates when $\mathbf{n_c}$ is very small. The value of **P(Wind** = **strong/PlayTennis** = **no)** is **.08** and that we have a sample containing only 5 examples for which **PlayTennis=no.**

Then the most probable value for $\mathbf{n_c}$ is **0.** This raises two difficulties. First, n_c/n produces a biased underestimate of the probability. Second, when this probability estimate is zero, this probability term

will dominate the Bayes classifier if the future query contains **Wind** = **strong.**

To avoid this difficulty we can adopt a Bayesian approach to estimating the probability, using the m-estimate defined as follows.

m-estimate of probability:

$$\frac{n_c + mp}{n + m} \qquad (6.22)$$

Here, $\mathbf{n_c}$ and **n** are defined as before, p is our prior estimate of the probability we wish to determine, and m is a constant called the **equivalent sample size,** which determines how heavily to weight p relative to the observed data. A typical method for choosing p in the absence of other information is to assume uniform priors, that is, if an attribute has **k** possible values we set p=1/k.

If m is zero, the m-estimate is equivalent to the simple fraction n_c/n. If both n and m are nonzero, then the observed fraction n_c/n and prior p will be combined according to the weight m.

4.10 BAYESIAN BELIEF NETWORKS

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities.

A Bayesian belief network describes the probability distribution over a set of variables. Consider an arbitrary set of random variables $Y_1 \ldots Y_n$ where each variable Y_i can take on the set of possible values $V(Y_i)$. We define the joint space of the set of variables Y to be the cross product $V(Y_1) * V(Y_2) * \ldots V(Y_n)$. In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables ($Y_1 \quad Y_n$). The probability distribution over this joint space is called the joint probability distribution. The joint probability distribution specifies the probability for each of the possible variable bindings for the tuple ($Y_1 \ldots Y_n$). A Bayesian belief network describes the joint probability distribution for a set of variables.

4.10.1 Conditional Independence

Let X,Y, and Z be three discrete-valued random variables. We say that X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given a value for Z; that is, if

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

where $\mathbf{x_i} \in V(X)$, $\mathbf{y_j} \in V(Y)$, and $\mathbf{z_k} \in V(Z)$. We commonly write the above expression in abbreviated form as P(X/Y, Z) = P(X/Z). We say that the set of variables $X_1 \ldots X_l$ is conditionally independent of the set of variables $\mathbf{Y_1} \ldots Y_m$ given the set of variables $Z_1 \quad Z_n$ if

$$P(X_1 \ldots X_l | Y_1 \ldots Y_m, Z_1 \ldots Z_n) = P(X_1 \ldots X_l | Z_1 \ldots Z_n)$$

The naive Bayes classifier assumes that the instance attribute A_1 is conditionally independent of instance attribute A_2 given the target value V. This allows the naive Bayes classifier to calculate P (A_1, A_2/V) in Equation (6.20) as follows

$$P(A_1, A_2|V) = P(A_1|A_2, V)P(A_2|V) \quad (6.23)$$

$$= P(A_1|V)P(A_2|V) \quad (6.24)$$

Equation (6.23) is the product rule of probability. Equation (6.24) follows because if A_1 is conditionally independent of A_2 given V, then by our definition of conditional independence $P(A_1 /A_2, V) = P(A_1/V)$.

4.10.2 Representation

A Bayesian belief network (Bayesian network for short) represents the joint probability distribution for a set of variables. For example, the Bayesian network in Figure 6.3 represents the joint probability distribution over the Boolean variables **Storm, Lightning, Thunder, ForestFire, Campfire,** and **BusTourGroup.** In general, a Bayesian network represents the joint probability distribution by specifying a set of conditional independence assumptions (represented by a directed acyclic graph), together with sets of local conditional probabilities.

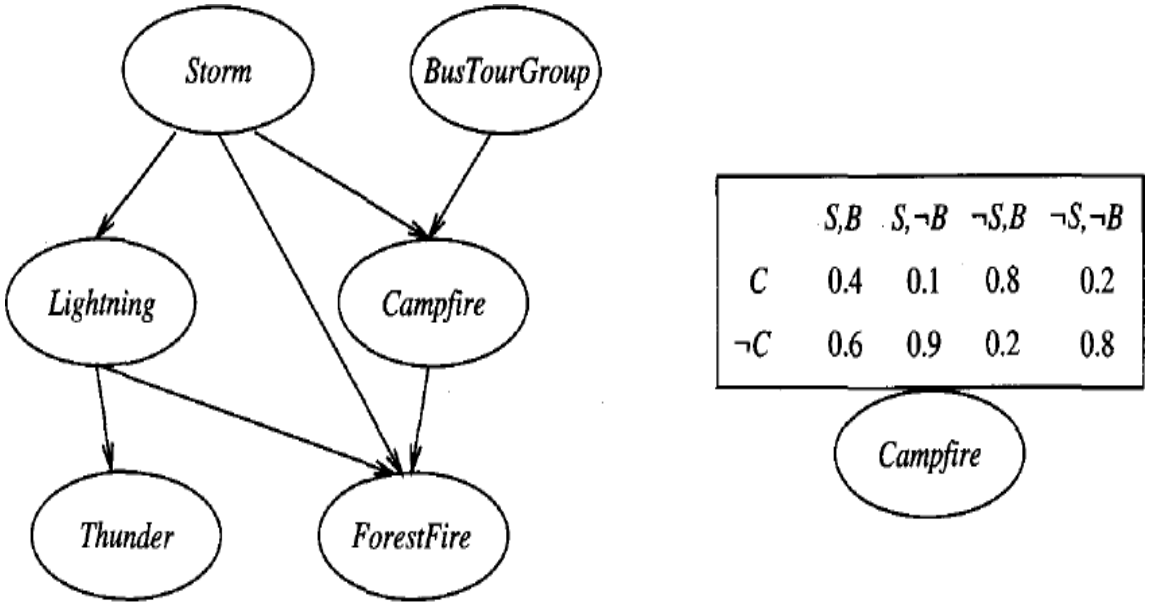

FIGURE 6.3

Each variable in the joint space is represented by a node in the Bayesian network. For each variable two types of information are specified. First, the network arcs represent the assertion that the variable is conditionally independent of its non descendants in the network given its immediate predecessors in the network. Second, a conditional probability table is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors. The joint probability for any de-sired assignment of values **(y_1 ,. . . , y_n)** to the tuple of network variables **(Y_1. . . Y_n)** can be computed by the formula.

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

where **Parents(Y_i)** denotes the set of immediate predecessors of Y_i in the net-work.

To illustrate, the Bayesian network in Figure 6.3 represents the joint probability distribution over the boolean variables **Storm, Lightning, Thunder, Forest-Fire, Campfire,** and **BusTourGroup.** Consider the node **Campfire.** The network nodes and arcs represent the assertion that **Campfire** is conditionally indepen-dent of its non descendants **Lightning** and **Thunder,** given its immediate parents **Storm** and **BusTourGroup.** This means that once we know the value of the variables **Storm** and **BusTourGroup,** the variables **Lightning** and **Thunder** provide no additional information about **Campfire.** The right side of the figure shows the conditional probability table associated with the variable **Campfire.** The top left entry in this table, for example, expresses the assertion that

$$P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4$$

Note this table provides only the conditional probabilities of **Campfire** given its parent variables **Storm** and **BusTourGroup.** The set of local conditional probability tables for all the variables, together with the set of conditional independence assumptions described by the network, describe the full joint probability distribution for the network.

One attractive feature of Bayesian belief networks is that they allow a convenient way to represent causal knowledge such as the fact that **Lightning** causes **Thunder.**

4.10.3 Inference

We might wish to use a Bayesian network to infer the value of some target variable (e.g., **ForestFire)** given the observed values of the other variables. We really wish to infer is the probability distribution for the target variable, which specifies the probability that it will take on each of its possible values given the observed values of the other variables.

This inference step can be straightforward if values for all of the other variables in the network are known exactly. In the more general case we may wish to infer the probability distribution for some variable (e.g., **ForestFire)** given observed values for only a subset of the other variables (e.g., **Thunder** and **BusTourGroup** may be the only observed values available). In general, a Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.

Exact inference of probabilities in general for an arbitrary Bayesian network is known to be NP-hard. Numerous methods have been proposed for probabilistic inference in Bayesian networks.

Gradient ascent procedure that learns the entries in the conditional probability tables is proposed. This gradient ascent procedure searches through a space of hypotheses that corresponds to the set of all possible entries for the conditional probability tables.

The objective function that is maximized during gradient ascent is the probability P(D/h) of the observed training data D given the hypothesis h. By definition, this corresponds to searching for the maximum likelihood hypothesis for the table entries.

4.10.4 Learning Bayesian Belief Networks

Can we devise effective algorithms for learning Bayesian belief networks from training data? Several different settings for this learning problem can be considered. First, the network structure might be given in advance, or it might have to be inferred from the training data. Second, all the network variables might be directly observable in each training example, or some might be unobservable.

In the case where the network structure is given in advance and the variables are fully observable in the training examples, learning the conditional probability tables is straightforward.

In the case where the network structure is given but only some of the variable values are observable in the training data, the learning problem is more difficult. This problem is somewhat analogous to learning the weights for the hidden units in an artificial neural network, where the input and output node values are given but the hidden unit values are left unspecified by the training examples.

Gradient ascent procedure that learns the entries in the conditional probability tables is proposed. This gradient ascent procedure searches through a space of hypotheses that corresponds to the set of all possible entries for the conditional probability tables. The objective function that is maximized during gradient ascent is the probability P(D1h) of the observed training data D given the hypothesis h. By definition, this corresponds to searching for the maximum likelihood hypothesis for the table entries.

4.10.5 Gradient Ascent Training of Bayesian Networks

The gradient ascent rule maximizes P(D/h) by following the gradient of ln P(D/h) with respect to the parameters that define the conditional probability tables of the Bayesian network. Let w_{ijk} denote a single entry in one of the conditional probability tables.

In particular, let w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_i, given that its immediate parents U_i take on the values given by u_{ik}.

For example, if w_{ijk} is the top right entry in the conditional probability table in Figure 6.3, then Y_i is the variable **Campfire,** U_i is the tuple of its parents

(Storm,BusTourGroup), y_{ij} = **True,** and u_{ik} = **(False,False).**

The gradient of ln P(D/h) is given by the derivatives $\frac{\partial Ph(D)}{\partial w_{ijk}}$ for each of the w_{ijk}. As we show

below, each of these derivatives can be calculated as

$$\frac{\partial \ln P(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \qquad (6.25)$$

For example, to calculate the derivative of ln **P(D/h)** with respect to the upper-rightmost entry in the table of Figure 6.3 we will have to calculate the quantity **P(Campfire** = **True , Storm** = **False, BusTourGroup** = **False/d)** for each training example **d** in **D .** When these variables are unobservable for the training example **d,** this required probability can be calculated from the observed variables in **d** using standard Bayesian network inference.

4.10.6 Learning the Structure of Bayesian Networks

Learning Bayesian networks when the network structure is not known in advance is also difficult. Cooper and Herskovits (1992) present a Bayesian scoring metric for choosing among alternative networks. They also present a heuristic search algorithm called **K_2** for learning network structure when the data is fully observable. K_2 performs a greedy search that trades off network complexity for accuracy over the training data.

In one experiment **K_2** was given a set of 3,000 training examples generated at random from a manually constructed Bayesian network containing 37 nodes and 46 arcs. This particular network described potential anesthesia problems in a hospital operating room. In addition to the data, the program was also given an initial ordering over the 37 variables that was consistent with the partial ordering of variable dependencies in the actual network. The program succeeded in reconstructing the correct Bayesian network structure almost exactly, with the exception of one incorrectly deleted arc and one incorrectly added arc.

Constraint-based approaches to learning Bayesian network structure have also been developed. These approaches infer independence and dependence relationships from the data, and then use these relationships to construct Bayesian networks.

4.11 THE EM ALGORITHM

In many practical learning settings, only a subset of the relevant instance features might be observable. For example, in training or using the Bayesian belief network of Figure 6.3, we might have data where only a subset of the network variables **Storm, Lightning, Thunder, ForestFire, Campfire,** and **BusTourGroup** have been observed. Many approaches have been proposed to handle the problem of learning in the presence of unobserved variables. The EM algorithm is a widely used approach to learning in the presence of unobserved variables. The EM algorithm can be used even for variables whose value is never directly observed, provided the general form of the probability distribution governing these variables is known.

4.11.1 Estimating Means of k Gaussians

Consider a problem in which the data D is a set of instances generated by a probability distribution that is a mixture of k distinct Normal distributions. Each instance is generated using a two-step process. First, one of the k Normal distributions is selected at random. Second, a single random instance $\mathbf{x_i}$ is generated according to this selected distribution. We consider the special case where the selection of the single Normal distribution at each step is based on choosing each with uniform probability, where each of the k Normal distributions has the same variance σ^2, ∂ and where σ^2 is

known. The learning task is to output a hypothesis h = (μ_1, . . . **μ_k)** that describes the means of each of the k distributions. We would like to find a maximum likelihood hypothesis for these means; that is, a hypothesis h that maximizes **p(D**/h).

It is easy to calculate the maximum likelihood hypothesis for the mean of a single Normal distribution given the observed data instances **x_1,x_2,** . . . , **x_m** drawn from this single distribution. Restating Equation (6.6) using our current notation, we have

$$\mu_{ML} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} (x_i - \mu)^2 \qquad (6.27)$$

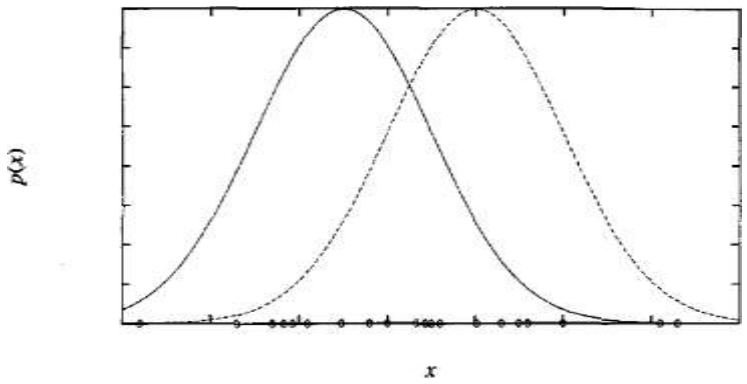

FIGURE 6.4: Instances generated by a mixture of two Normal distributions with identical variance a. The instances are shown by the points along the x axis. If the means of the Normal distributions are unknown, the EM algorithm can be used to search for their maximum likelihood estimates.

In this case, the sum of squared errors is minimized by the sample mean

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i \qquad (6.28)$$

Our problem here involves a mixture of k different Normal distributions, and we cannot observe which instances were generated by which distribution.

Applied to our k-means problem the EM algorithm searches for a maximum likelihood hypothesis by repeatedly re-estimating the expected values of the hid-den variables **z_{ij}** given its current hypothesis **(μ_1....μ_k),** then recalculating the maximum likelihood hypothesis using these expected values for the hidden variables.

Applied to the problem of estimating the two means the EM algorithm first initializes the hypothesis to h=(μ_1, μ_2), where μ_1 and μ_2 are arbitrary initial values. It then iteratively re-estimates h by repeating the following two steps until the procedure converges to a stationary value for h.

Step1: Calculate the expected value E[z_{ij}] of each hidden variable z_{ij}, assuming the current hypothesis h= (μ_1, μ_2) holds.
Step2: Calculate a new maximum likelihood hypothesis $h' = (\mu'_1, \mu'_2)$, assuming the value taken on by each hidden variable z_{ij} is its expected value E[z_{ij}] calculated in step1. Then replace the hypothesis h=(μ_1, μ_2) by the new hypothesis $h' = (\mu', \mu')_2$and iterate.

Let us examine how both of these steps can be implemented. Step1 must calculate $z_{ij.}$ This E[z_{ij}] is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

Thus the first step is implemented by substituting the current values the observed $\mathbf{x_i}$ into the above expression.

In the second step we use the $\mathbf{E[z_{ij}]}$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = (\mu'_1, \mu'_2)$. The maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{1}{m} \sum_{i=1}^{m} E[z_{ij}] x_i$$

The above algorithm for estimating the means of a mixture of k Normal distributions illustrates the essence of the EM approach: The current hypothesis is used to estimate the unobserved variables, and the expected values of these variables are then used to calculate an improved hypothesis. It can be proved that on each iteration through this loop, the EM algorithm increases the likelihood P(D/h) unless it is at a local maximum. The algorithm thus converges to a local maximum likelihood hypothesis for $\mathbf{(\mu_1, \mu_2)}$.

4.11.2 General Statement of EM Algorithm

The EM algorithm can be applied in many settings where we wish to estimate some set of parameters θ that describe an underlying probability distribution, given only the observed portion of the full data produced by this distribution.

In general let X = $\mathbf{\{x_1, \ldots x_m,\}}$ denote the observed data in a set of m independently drawn instances, let Z = $\mathbf{\{z_1, \ldots, z_m\}}$ denote the unobserved data in these same instances, and let **Y** = X ∪ Z denote the full data. Note the unobserved Z can be treated as a random variable whose probability distribution depends on the unknown parameters Θ and on the observed data X. Similarly, **Y** is a random variable because it is defined in terms of the random variable Z.

We use **h** to denote the current hypothesized values of the parameters θ, and **h'** to denote the revised hypothesis that is estimated on each iteration of the EM algorithm.

The EM algorithm searches for the maximum likelihood hypothesis **h'** by seeking the **h'** that maximizes E[ln **P(Y/h')] .**This expected value is taken over the probability distribution governing **Y ,** which is determined by the unknown parameters Θ.

First, **P(Y/h')** is the likelihood of the full data **Y** given hypothesis **h'.** It is reasonable that we wish to find a **h'** that maximizes some function of this quantity. Second, maximizing the logarithm of this quantity In **P(Y/h')** also maximizes **P(Y/h') ,** as we have discussed on several occasions already.

$$\ln P(Y|h') = \ln \prod_{i=1}^{m} p(y_i|h')$$

$$= \sum_{i=1}^{m} \ln p(y_i|h')$$

$$= \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2 \right)$$

Finally we must take the expected value of this In P(Y/h') over the probability distribution governing Y or, equivalently, over the distribution governing the un-observed components z_{ij} of Y. Note the above expression for In P(Y/ h') is a linear function of these z_{ij}. In general, for any function f (z) that is a linear function of z, the following equality holds

$$E[f(z)] = f(E[z])$$

This general fact about linear functions allows us to write

$$E[\ln P(Y|h')] = E\left[\sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2 \right) \right]$$

$$= \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2 \right)$$

To summarize, the function Q(h'/h) for the k means problem is

$$Q(h'|h) = \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2 \right)$$

where $h = (\mu_1 \ldots \mu_k)$ and where E[z_{ij}] is calculated based on the current hypothesis h and observed data X. As discussed earlier

$$E[z_{ij}] = \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}} \qquad (6.29)$$

Thus, the first (estimation) step of the EM algorithm defines the Q function based on the estimated E[z_{ij}] terms. The second (maximization) step then finds the values $\mu'_1 \ldots \mu'_k$ that maximize this Q function. In the current case

$$\underset{h'}{\text{argmax}}\, Q(h'|h) = \underset{h'}{\text{argmax}} \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2 \right)$$

$$= \underset{h'}{\text{argmin}} \sum_{i=1}^{m} \sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2 \qquad (6.30)$$

Thus, the maximum likelihood hypothesis here minimizes a weighted sum of squared errors, where the contribution of each instance x_i to the error that defines μ'_j is weighted by E[z_{ij}]. The quantity given by Equation (6.30) is minimized by setting each μ'_j to the weighted sample mean

$$\mu_j \leftarrow \frac{1}{m} \sum_{i=1}^{m} E[z_{ij}]\, x_i \qquad (6.31)$$