CS601PC: MACHINE LEARNING

III Year B.Tech. CSE II-Sem.

UNIT – III

Bayesian learning – Introduction, Bayes theorem, Bayes theorem and concept learning, Maximum Likelihood and least squared error hypotheses, maximum likelihood hypotheses for predicting probabilities, minimum description length principle, Bayes optimal classifier, Gibs algorithm, Naïve Bayes classifier, an example: learning to classify text, Bayesian belief networks, the EM algorithm.

Computational learning theory – Introduction, probably learning an approximately correct hypothesis, sample complexity for finite hypothesis space, sample complexity for infinite hypothesis spaces, the mistake bound model of learning.

Instance-Based Learning - Introduction, k-nearest neighbour algorithm, locally weighted regression, radial basis functions, case-based reasoning, remarks on lazy and eager learning.

TEXT BOOKS:

1. Machine Learning – Tom M. Mitchell, - MGH (Page Nos. 154 to 248)

BAYESIAN LEARNING

Introduction:

- Bayesian reasoning provides a **probabilistic approach** to inference.
- Bayesian learning algorithms that calculate probabilities for hypotheses, such as the **naive Bayes classifier**. Its application is to classify text documents such as electronic news articles.
- Bayesian analysis to justify a key **design choice in neural network** learning algorithms: choosing to **minimize the sum of squared errors** when searching the space of possible neural networks.
- Bayesian perspective is to analyze the inductive bias of **decision tree learning algorithms** that favor **short decision trees** and examine the closely related **Minimum Description** Length principle.

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** (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Bayesian methods prove computationally intractable; they can provide a standard of **optimal decision making** against which other practical methods can be measured.

BAYES THEOREM

Introduction : Bayes theorem gives the probability of an event based on prior knowledge of conditions.

Proof: Let us take 2 events: A and B

Conditional Probability $P(A|B) = \frac{P(A \cap B)}{P(B)}$ ie., $P(A|B) \times P(B) = P(A \cap B)$

Conditional Probability $P(B|A) = \frac{P(B \cap A)}{P(A)}$ ie. $P(B|A) \times P(A) = P(B \cap A)$

We know that $P(A \cap B) = P(B \cap A)$

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

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$ where A- hypothesis, B- given Data

Bayes theorem is the basis of Bayesian learning methods because it 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).

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

P(h|D) increases with P(h) and with P(D| h) according to Bayes theorem. P(h|D) decreases as P(D) increases, because the more probable it is that D will be observed independent of h.

Maximally probable hypothesis is called a **maximum a posteriori (MAP) hypothesis**. MAP hypotheses (h_{MAP}) can be determined by using Bayes theorem to calculate the posterior probability of each candidate hypothesis.

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

P(D) dropped because it is a constant independent of h.

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**, $\mathbf{h_{ML}}$**.**

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

For example,

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

The available data is from a particular laboratory test with two possible outcomes: $\oplus$ (positive) and $\ominus$ (negative).

We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the 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:

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

The maximum a posteriori hypothesis (h_{MAP}) can be found using Equation :

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

The exact posterior probabilities can also be determined by **normalizing** the above quantities so that they sum to 1

(e.g., P(cancer($\oplus$))= $\frac{0.0078}{0.0078+0.0298}$ = 0.21, P(cancer($\ominus$))= $\frac{0.0298}{0.0078+0.0298}$ = 0.79)

Thus, h_{MAP}= ¬cancer.

BAYES THEOREM AND CONCEPT LEARNING

Bayes theorem calculates the probability of each possible hypothesis and outputs the most probable one.

Brute-Force Bayes Concept Learning:

- Consider 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}.**
- Sequence of instances (x_1 . . . x_m) and sequence of target values D = (d_1, . . . d_m). The 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)).

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

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

- Assume

  1. The training data D is noise free (i.e., d_i = c(x_i)).
  2. The target concept c is contained in the hypothesis space H
  3. We have no a priori reason to believe that any hypothesis is more probable than any other.
  4. Sum of prior probabilities is 1.

From the above assumptions, we can choose

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

The probability of data D given hypothesis h is 1 if D is consistent with h, and 0 otherwise. $d_i \neq h(x_i)$

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

If h is consistent with D,

$$\begin{aligned} P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H.D}|} \text{ if } h \text{ is consistent with } D \end{aligned}$$

where $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D (i.e., $VS_{H,D}$ is the version space of H with respect to D).

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

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

where $|VS_{H,D}|$ is the number of hypotheses from H consistent with D.

The above analysis implies that under our choice for P(h) and P(D|h), every consistent hypothesis has posterior probability $(1 / |VS_{H,D}|)$, and every inconsistent hypothesis has posterior probability 0. Every consistent hypothesis is, therefore, a MAP hypothesis.

MAXIMUM LIKELIHOOD AND LEAST SQUARED ERROR HYPOTHESES

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.

In this section, we have to show that the least-squared error hypothesis is the maximum likelihood hypothesis.

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

Where lower case p to refer to the probability density.

Assume a fixed set of training instances ($x_1 \ldots x_m$) and the data D to be the corresponding sequence of target values $D = (d_1 \ldots d_m)$.

Assume the training examples are mutually independent given h, 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 $\mu = f(x_i)$. The normal distribution formula is

$$\mathbf{p(x|\mu)} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Apply the above formula into h_{ML}.

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

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

Rather than maximizing the above complicated expression we shall choose to maximize its (less complicated) logarithm. Therefore maximizing In 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.

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

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, the above equation shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values d_i and the hypothesis predictions $h(x_i)$. Minimizing the sum of squared errors is a common approach in many neural networks.

MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

Predicting Probability:

Consider the function f: X-> {0,1}, which has two discrete output values.

Ex. If target function f(x) might be 1 if the patient survives the disease and 0 if not.

If f^1: X-> {0,1}, $f^1(x) = 0.92$ whereas the probabilistic function will be equal to 1 in 92% of cases.

To find a maximum likelihood hypothesis for f^1,

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

For example, the probability that our training set contains a particular patient x_i is independent of our hypothesis When x is independent of h, we can rewrite the above expression as

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

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

Re-express the above as,

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

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

The expression for the maximum likelihood hypothesis is

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

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

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

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

The above equation describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis.

MINIMUM DESCRIPTION LENGTH(MDL) PRINCIPLE

- Representing a concept in minimum possible way – Then it is said to be good one. That means shorter hypothesis/shorter decision tree.
- Choosing the shortest explanation for the observed data

Consider the definition of h_{MAP}

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

which can be equivalently expressed in terms of maximizing the log,

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

or alternatively, minimizing the negative of this quantity. The below statement interpreted that short hypotheses are preferred.

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

Where,

- $\log_2$ P(h) is is the size of the description of hypothesis h using this optimal representation. L_{CH} (h) = - $\log_2$P(h), where CH is the optimal code for hypothesis space H.

-log2 P(D|h) is the description length of the training data D given hypothesis h, under its optimal encoding. L_{CH} (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 h_{MAP} equation as

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

We can state the Minimum Description Length (MDL) Principle as,

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

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths.

BAYES OPTIMAL CLASSIFIER

Bayes optimal classifier is a probabilistic model that makes the most probable classification for the new instance. (Finding optimal solution). It can be answered by applying MAP hypothesis to the new instance.

Consider,

A hypothesis space containing three hypotheses, h_1, h_2, and h_3 and the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus h_1 is 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

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

The optimal classification of the new instance is the value v_j, for which $P(v_j \mid D)$ 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

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

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

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

Any system that classifies new instances according to Bayes optimal classification Equation is called a Bayes optimal classifier, or Bayes optimal learner.

This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

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 P(h|D) 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. 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.

E(error gibs algorithm) ≤ 2 E(error Bayesoptimal algorithm)

NAIVE BAYES CLASSIFIER

It is one of classification technique based on Bayes theorem with an assumption of independence among features.

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 x.

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

We can use Bayes theorem to rewrite this expression as

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

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

The probability of observing the conjunction a_1, a_2.. .a_n is the product of the probabilities for the individual attributes:

Naive Bayes classifier:

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

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

For example,

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

TABLE 3.2
Training examples for the target concept *PlayTennis*.

Use the naive Bayes classifier and the training data from this table 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 Play Tennis for this new instance. The target value v_{NB} is given by

$$\begin{aligned} v_{NB} &= \underset{v_j \in \{yes,no\}}{\operatorname{argmax}} P(v_j) \prod_i P(a_i|v_j) \\ &= \underset{v_j \in \{yes,no\}}{\operatorname{argmax}} P(v_j) \quad P(Outlook = sunny|v_j) P(Temperature = cool|v_j) \\ &\qquad P(Humidity = high|v_j) P(Wind = strong|v_j) \end{aligned} \tag{6.21}$$

To calculate v_{NB} we now require 10 probabilities that can be estimated from the training data. First, the probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples.

$$P(PlayTennis = yes) = 9/14 = .64$$

$$P(PlayTennis = no) = 5/14 = .36$$

Similarly, we can estimate the conditional probabilities. For example, those for Wind = strong are calculated by the equation n_c/n.

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

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

Using these probability estimates and similar estimates for the remaining attribute values, we calculate v_{NB} according to Equation

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

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

Thus, the naive Bayes classifier assigns the target value Play Tennis = no to this new instance, based on the probability estimates learned from the training data. Furthermore, by normalizing the above quantities to sum to one we can calculate the conditional probability that the target value is no, given the observed attribute values. For the current example, this probability is

$$\frac{0.0206}{0.0206+0.0053} = 0.795$$

If n_c is very low value like 0.08 or zero, we can adopt a Bayesian approach to estimating the probability, using the m-estimate defined as follows.

m-estimate of probability:

$$\frac{n_c + mp}{n + m}$$

Here,
p is prior estimate of the probability
m is a constant called the equivalent sample size, which determines how heavily to weight p relative to the observed data

LEARN_NAIVE_BAYES_TEXT($Examples$, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - $Vocabulary \leftarrow$ the set of all distinct words and other tokens occurring in any text document from $Examples$
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*
   - For each target value v_j in V do
     - $docs_j \leftarrow$ the subset of documents from $Examples$ for which the target value is v_j
     - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
     - $n \leftarrow$ total number of distinct word positions in $Text_j$
     - for each word w_k in $Vocabulary$
       - $n_k \leftarrow$ number of times word w_k occurs in $Text_j$
       - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT(Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the ith position within Doc.

- $positions \leftarrow$ all word positions in Doc that contain tokens found in $Vocabulary$
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\text{argmax}}\, P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

TABLE 6.2
Naive Bayes algorithms for learning and classifying text. In addition to the usual naive Bayes assumptions, these algorithms assume the probability of a word occurring is independent of its position within the text.

Instances are text documents. For example, we might wish to learn the target concept

"electronic news articles that I find interesting,"

or

"pages on the World Wide Web that discuss machine learning topics."

In both cases, if a computer could learn the target concept accurately, it could automatically filter the large volume of online text documents to present only the most relevant documents to the user.

- Consider an instance space X consisting of all possible text documents (i.e., all possible strings of words and punctuation of all possible lengths).

> Our approach to representing arbitrary text documents is disturbingly simple: Given a text document, such as this paragraph, we define an attribute for each word position in the document and define the value of that attribute to be the English word found in that position. Thus, the current paragraph would be described by 11 1 attribute values, corresponding to the 11 1 word positions. The value of the first attribute is the word "our," the value of the second attribute is the word "approach," and so on. Notice that long text documents will require a larger number of attributes than short documents. As we shall see, this will not cause us any trouble.

- The task is to learn from these training examples to predict the target value for subsequent text documents.
- For illustration, we will consider the target function classifying documents as interesting or uninteresting to a particular person, using the target values like and dislike to indicate these two classes.
- we define an attribute for each word position in the document and define the value of that attribute

$$v_{NB} = \underset{v_j \in \{like, dislike\}}{\text{argmax}} P(v_j) \prod_{i=1}^{111} P(a_i | v_j)$$

$$= \underset{v_j \in \{like, dislike\}}{\text{argmax}} P(v_j)\ P(a_1 = \text{“}our\text{”} | v_j) P(a_2 = \text{“}approach\text{”} | v_j)$$

$$\ldots\ P(a_{111} = \text{“}trouble\text{”} | v_j)$$

- let us assume we are given a set of 700 training documents that a friend has classified as **dislike** and another 300 she has classified as **like**. That is P(1ike) = .3 and P(dis1ike) = .7
- As usual, estimating the class conditional probabilities (e.g., P(a_l = "our"|dis1ike)) is more problematic because we must estimate one such probability term for each combination of text position, English word, and target value. Unfortunately, there are approximately 50,000 distinct words in the English vocabulary, 2 possible target values, and 111 text positions in the current example, so we must estimate 2.x111x50,000 = 10 million such terms from the training data.

EXPERIMENT ON ALGORITHM:

comp.graphics	misc.forsale	soc.religion.christian	sci.space
comp.os.ms-windows.misc	rec.autos	talk.politics.guns	sci.crypt
comp.sys.ibm.pc.hardware	rec.motorcycles	talk.politics.mideast	sci.electronics
comp.sys.mac.hardware	rec.sport.baseball	talk.politics.misc	sci.med
comp.windows.x	rec.sport.hockey	talk.religion.misc	
		alt.atheism	

TABLE 6.3
Twenty usenet newsgroups used in the text classification experiment. After training on 667 articles from each newsgroup, a naive Bayes classifier achieved an accuracy of 89% predicting to which newsgroup subsequent articles belonged. Random guessing would produce an accuracy of only 5%.

- 20 electronic newsgroups were considered
- 1,000 articles were collected from each newsgroup, forming a data set of 20,000 documents
- The naive Bayes algorithm was then applied using two-thirds of these 20,000 documents as training examples, and performance was measured over the remaining third.
- The accuracy achieved by the program was 89%

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.

Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier.

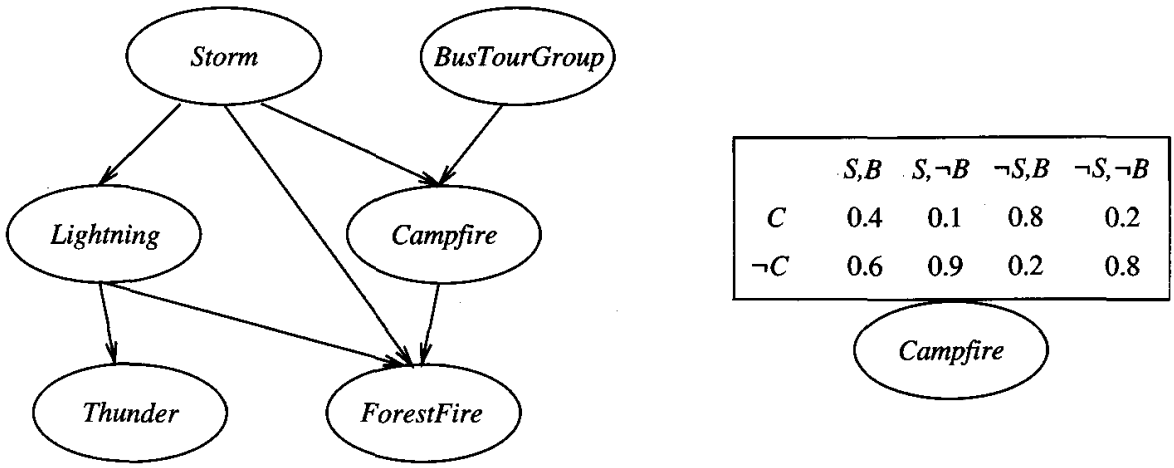

FIGURE 6.3
A Bayesian belief network. The network on the left represents a set of conditional independence assumptions. In particular, each node is asserted to be conditionally independent of its nondescendants, given its immediate parents. Associated with each node is a conditional probability table, which specifies the conditional distribution for the variable given its immediate parents in the graph. The conditional probability table for the *Campfire* node is shown at the right, where *Campfire* is abbreviated to C, *Storm* abbreviated to S, and *BusTourGroup* abbreviated to B.

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. Each variable in the joint space is represented by a node in the Bayesian network.

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

where $Parents(Y_i)$ denotes the set of immediate predecessors of Y_i in the network. Note the values of $P(y_i|Parents(Y_i))$ are precisely the values stored in the conditional probability table associated with node Y_i.

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

Note this table provides only the conditional probabilities of Campfire given its parent variables Storm and BusTourGroup.

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. In the terminology of conditional independence, we express this by stating that Thunder is conditionally independent of other variables in the network, given the value of Lightning.

THE EM ALGORITHM (Expectation – Maximization)

- The EM algorithm can be used even for variables whose value is never directly observed
- The EM algorithm has been used to train Bayesian belief networks
- The EM algorithm is also the basis for many unsupervised clustering algorithms

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.

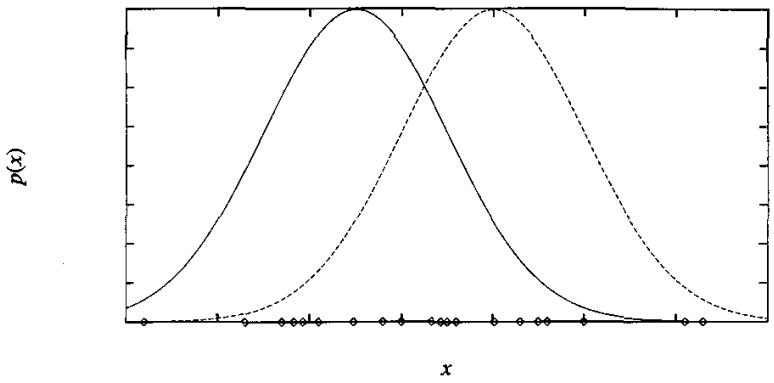

FIGURE 6.4
Instances generated by a mixture of two Normal distributions with identical variance σ. The instances are shown by the points along the x axis. If the means of the Normal distributions are unknown, the EM algorithm can be used to search for their maximum likelihood estimates.

Each instance is generated using a two-step process.

1. One of the k Normal distributions is selected at random. (k=2)
2. A single random instance x_i is generated according to this selected distribution.

This process is repeated to generate a set of data points as shown in the figure.

Note it is easy to calculate the **maximum likelihood hypothesis** for the mean of a single Normal distribution given the observed data instances $x_I, x_2, \ldots, x_m$ drawn from this single distribution.

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

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

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i$$

In the example of Figure 6.4, we can think of the full description of each instance as the triple (x_i,z_{il} , z_{i2}), where x_i is the observed value of the ith instance and where z_{il} and z_{i2} indicate which of the two Normal distributions was used to generate the value x_i.

EM algorithm first initializes the hypothesis to h = (μ1, μ2), where μ1 and μ2 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.

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

Steps involved in EM Algorithm:

1. Initially, a set of initial values are considered. A set of incomplete data is given to system
2. Next step – Expectation step – E step. Here, we use observed data to estimate or guess the value of missed / incomplete data.
3. Maximization step – M step. Here, we use the complete data generated in preceding E-step to update the values.
4. We check if values are converging/ not. If converging – stop. Otherwise, Repeat step 2 and 3 till the convergence occurred.

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). The algorithm thus converges to a local maximum likelihood hypothesis for (μl, μ2).

Usage:

1. Used to fill missed data
2. Used for unsupervised clustering
3. Used to discover values of unobserved variable

Advantages:

1. With each iteration, likelihood increases
2. E-step and M-step are easy to implement.

Disadvantages:

1. Slow convergence
2. Makes convergence to local optimal only.

COMPUTATIONAL LEARNING THEORY

INTRODUCTION

The goal of this chapter is to answer questions such as:

1. **Sample complexity.** How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?
2. **Computational complexity.** How much computational effort is needed for a learner to converge (with high probability) to a successful hypothesis?
3. **Mistake bound**. How many training examples will the learner misclassify before converging to a successful hypothesis?

PROBABLY LEARNING AN APPROXIMATELY CORRECT HYPOTHESIS (PAC)

PAC model describes that how many training examples and how much computation are required.

First, we will not require that the learner output a zero error hypothesis-we will require only that its error be bounded by some constant, €, that can be made arbitrarily small. Second, we will not require that the learner succeed for every sequence of randomly drawn training examples-we will require only that its probability of failure be bounded by some constant, δ, that can be made arbitrarily small. In short, we require only that the learner probably learn a hypothesis that is approximately correct-**hence the term probably approximately correct learning, or PAC learning for short.**

Definition: Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is **PAC-learnable** by L using H if for all $c \in C$, distributions $\mathcal{D}$ over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n, and $size(c)$.

SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACE

PAC-learnability is largely determined by the number of training examples required. The growth in the number of required training examples with problem size, called the **sample complexity**.

A learner is consistent if it outputs hypotheses that perfectly fit the training data. To derive a bound on the number of training examples required by any consistent learner, we can use version space.

The significance of the version space here is that every consistent learner outputs a hypothesis belonging to the version space, regardless of the instance space X, hypothesis space H, or training data D. The reason is simply that by definition the version space $VS_{H,D}$ contains every consistent hypothesis in H. Therefore, to bound the number of examples needed by any consistent learner, we need only bound the number of examples needed to assure that the version space contains no unacceptable hypotheses.

Definition: Consider a hypothesis space H, target concept c, instance distribution $\mathcal{D}$, and set of training examples D of c. The version space $VS_{H,D}$ is said to be ϵ-**exhausted** with respect to c and $\mathcal{D}$, if every hypothesis h in $VS_{H,D}$ has error less than ϵ with respect to c and $\mathcal{D}$.

$$(\forall h \in VS_{H,D})\ error_{\mathcal{D}}(h) < \epsilon$$

The numbers of training examples are determined by

$$m \geq \frac{1}{\epsilon}(\ln |H| + \ln(1/\delta))$$

SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

In the above section we showed that sample complexity for PAC learning grows as the logarithm of the size of the hypothesis space.

There are two drawbacks to characterizing sample complexity in terms of |H|. First, it can lead to quite weak bounds. Second, in the case of infinite hypothesis spaces we cannot apply the above Equation at all.

Here we consider a second measure of the complexity of H, called the Vapnik-Chervonenkis dimension of H (VC dimension, or VC(H), for short). We can state bounds on sample complexity that use VC(H) rather than |H| to characterize the sample complexity of many infinite hypothesis spaces.

The VC dimension measures the complexity of the hypothesis space H, not by the number of distinct hypotheses |H|, but instead by the number of distinct instances from X that can be completely discriminated using H.

Definition: The **Vapnik-Chervonenkis dimension,** $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$.

We can now bound the number m of training examples sufficient to learn,

$$m \geq \frac{1}{\epsilon}(4\log(2/\delta) + 8VC(H)\log(13/\epsilon))$$

$$\geq \frac{1}{\epsilon}(4\log(2/\delta) + 16(r+1)s\log(es)\log(13/\epsilon))$$

THE MISTAKE BOUND MODEL OF LEARNING

The learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis. We are interested in minimizing the total number of mistakes it will make before converging to the correct target function. Here the total number of mistakes can be even more important than the total number of training examples.

To calculate the number of mistakes it will make in **Find-S algorithm**, we need only count the number of mistakes it will make misclassifying truly positive examples as negative.

Definition: Let C be an arbitrary nonempty concept class. The **optimal mistake bound** for C, denoted $Opt(C)$, is the minimum over all possible learning algorithms A of $M_A(C)$.

$$Opt(C) \equiv \min_{A \in learning\ algorithms} M_A(C)$$

The **WEIGHTED-MAJORITlY algorithm** makes predictions by taking a weighted vote among a pool of prediction algorithms and learns by altering the weight associated with each prediction algorithm.

One interesting property of the WEIGHTED-MAJORITY algorithm is that it is able to accommodate inconsistent training data. This is **because it does not eliminate a hypothesis that is found to be inconsistent with some training example, but rather reduces its weight**. A second interesting property is that we can **bound the number of mistakes** made by WEIGHTED-MAJORITY in terms of the number of mistakes committed by the best of the pool of prediction algorithms.

The WEIGHTED-MAJORITY algorithm begins by assigning a weight of 1 to each prediction algorithm, then considers the training examples. **Whenever a prediction algorithm misclassifies a new training example its weight is decreased by multiplying it by some number β, where $0 \leq \beta < 1$.**

a_i denotes the i^{th} prediction algorithm in the pool A of algorithms. w_i denotes the weight associated with a_i.

- For all i initialize $w_i \leftarrow 1$
- For each training example $\langle x, c(x) \rangle$
  - Initialize q_0 and q_1 to 0
  - For each prediction algorithm a_i
    - If $a_i(x) = 0$ then $q_0 \leftarrow q_0 + w_i$
      If $a_i(x) = 1$ then $q_1 \leftarrow q_1 + w_i$
- If $q_1 > q_0$ then predict $c(x) = 1$
  If $q_0 > q_1$ then predict $c(x) = 0$
  If $q_1 = q_0$ then predict 0 or 1 at random for $c(x)$
- For each prediction algorithm a_i in A do
  If $a_i(x) \neq c(x)$ then $w_i \leftarrow \beta w_i$

TABLE 7.1
WEIGHTED-MAJORITY algorithm.

The number of mistakes committed by the WEIGHTED MAJORITY algorithm can be bounded in terms of the number of mistakes made by the best prediction algorithm in the voting pool.

Theorem 7.5. Relative mistake bound for WEIGHTED-MAJORITY. Let D be any sequence of training examples, let A be any set of n prediction algorithms, and let k be the minimum number of mistakes made by any algorithm in A for the training sequence D. Then the number of mistakes over D made by the WEIGHTED-MAJORITY algorithm using $\beta = \frac{1}{2}$ is at most

$$2.4(k + \log_2 n)$$

To summarize, the above theorem states that the number of mistakes made by the **WEIGHTED-MAJORITY algorithm will never be greater than a constant factor times the number of mistakes made by the best member of the pool**, plus a term that grows only logarithmically in the size of the pool.

INSTANCE-BASED LEARNING

INTRODUCTION

- Learning in these instance based learning algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance. ie. Memorise and then apply.
- Also called as memory based learning/ lazy learning
- Instance-based learning methods:
  - K-Nearest Neighbor algorithm(KNN)
  - Locally weighted regression
  - Radial based functions (RBF)
  - Case based Reasoning (CBR) - Case-based reasoning has been applied to tasks such as storing and reusing past experience at a help desk, reasoning about legal cases by referring to previous cases, and solving complex scheduling problems by reusing relevant portions of previously solved problems.

- *Regression* means approximating a real-valued target function.
- *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$.

K-NEAREST NEIGHBOR LEARNING ALGORITHM (KNN)

The most basic instance-based method is the k-NEAREST NEIGHBOR algorithm. Let an instance x be described by the vector { $a_1(x)$, $a_2(x)$, ….$a_n(x)$ } where a_r (x) denotes the value of the r^{th} attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

k-NEAREST NEIGHBOR algorithm computes the classification of each new query instance as needed. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

TABLE 8.1
The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

If we choose k = 1, then the 1-NEAREST NEIGHBOUR algorithm assigns to $f(x_q)$ the value f (x_i) where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

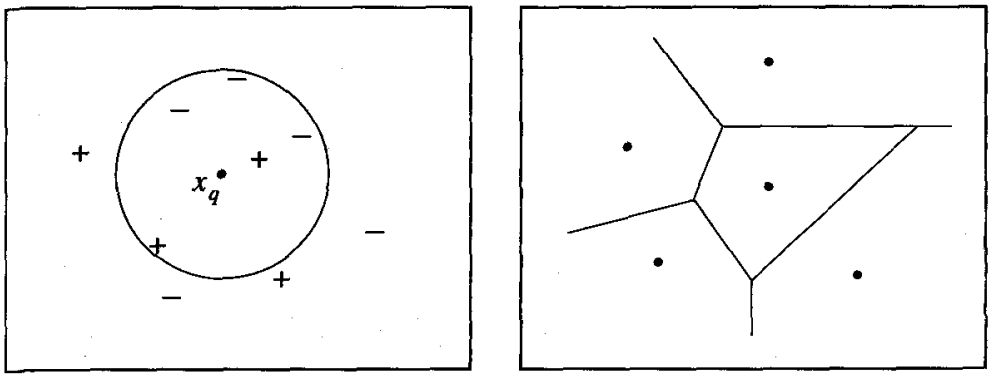

FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

The right side diagram of Figure 8.1 is often called the **Voronoi diagram** of the set of training examples.

Distance-Weighted NEAREST NEIGHBOR Algorithm:

Consider applying k-NEAREST NEIGHBOR to a problem in which each instance is described by 20 attributes, but where only 2 of these attributes are relevant to determining the classification for the particular target function. **As a result, the similarity metric used by k-NEAREST NEIGHBOR—depending on all 20 attributes-will be misleading.** The distance between neighbors will be dominated by the large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the curse of dimensionality. Nearest-neighbor approaches are especially sensitive to this problem. One interesting approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances.

One obvious refinement to the k-NEAREST NEIGHBOR algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point x_q, giving **greater weight to closer neighbors**.

This can be accomplished by replacing the final line of the algorithm by

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

For example, Classifying T-Shirt based on value of K

X2 (Height)	Y2 (Weight)	T-Shirt	X1-X2	SQUARE (X1-X2)	Y1-Y2	SQUARE (Y1-Y2)	SUM	Square root
160	60	M	1	1	1	1	2	1.414213562
163	61	M	-2	4	0	0	4	2
160	59	M	1	1	2	4	5	2.236067977
163	60	M	-2	4	1	1	5	2.236067977
160	64	L	1	1	-3	9	10	3.16227766
158	59	M	3	9	2	4	13	3.605551275
158	63	M	3	9	-2	4	13	3.605551275
163	64	L	-2	4	-3	9	13	3.605551275
165	61	L	-4	16	0	0	16	4
165	62	L	-4	16	-1	1	17	4.123105626

158	58	M	3	9	3	9	18	4.242640687
165	65	L	-4	16	-4	16	32	5.656854249
168	62	L	-7	49	-1	1	50	7.071067812
168	63	L	-7	49	-2	4	53	7.280109889
168	66	L	-7	49	-5	25	74	8.602325267
170	63	L	-9	81	-2	4	85	9.219544457
170	64	L	-9	81	-3	9	90	9.486832981
170	68	L	-9	81	-7	49	130	11.40175425

EUCLIDEAN DISTANCE	**SQRT(SQUARE(Y1-Y2)+(SQUARE(X1-X2))**			
NEW INPUT DATA	**161CM**	**HEIGHT**	**161**	**X1**
	61KG	**WEIGHT**	**61**	**Y1**

LOCALLY WEIGHTED REGRESSION

Locally weighted regression is a generalization of nearest-neighbor approach. The phrase "locally weighted regression" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Let us consider the case of locally weighted regression in which the target function f is approximated near x_q using a linear function of the form.

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

$a_i(x)$ denotes the value of the i^{th} attribute of the instance x.

The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below. Note we write the error $E(x_q)$ to emphasize the fact that now the error is being defined as a function of the query point x_q.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Criterion three is a good approximation and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number k of neighbors considered.

RADIAL BASIS FUNCTIONS

One approach to function approximation that is closely related to distance-weighted regression and also artificial neural networks is learning with radial basis functions. In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

where each x_u is an instance from X and where the kernel function K_u ($d(x_u, x)$) is defined so that it decreases as the distance $d(x_u, x)$ increases. Here k is a user provided constant that specifies the number of kernel functions to be included.

The above Equation can be viewed as describing a **two layer network** where the **first layer of units computes the values of the various K_u ($d(x_u, x)$)** and where the **second layer computes a linear combination of these first-layer unit values**. An example radial basis function (RBF) network is illustrated in Figure 8.2.

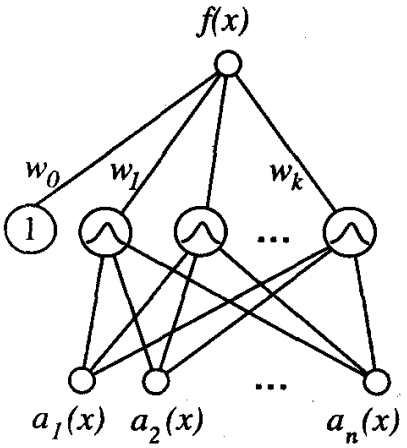

FIGURE 8.2
A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u. Therefore, its activation will be close to zero unless the input x is near x_u. The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

Given a set of training examples of the target function, RBF networks are typically trained in a **two-stage process**. First, **the number k of hidden units is determined** and each hidden unit **u** is defined by choosing the values of x_u and σ_u^2 that define its kernel function $K_u(d(x_u, x))$. Second, **the weights w_u are trained to maximize the fit of the network to the training data**, using the global error criterion.

The **first approach** is to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at the point x_u with some variance.

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

One advantage of this kernel functions is that it allows the RBF network to fit the training data exactly. That is, for any set of m training examples the weights $w_o \ldots w_m$ for combining the m Gaussian kernel functions can be set so that $\hat{f}(x_i) = f(x_i)$ for each training example $\{x_i, f(x_i)\}$.

The **second approach** is to choose a set of kernel functions that is **smaller than the number of training examples**. This approach can be much more efficient than the first approach, especially when the number of training examples is large.

One key advantage to RBF networks is that they can be trained much more efficiently than feed forward networks trained with BACKPROPAGATION. This follows from the fact that the input layer and the output layer of an RBF are trained separately.

CASE BASED REASONING (CBR)

Case-based reasoning (CBR) is a learning paradigm and instances are represented using more symbolic descriptions. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems.

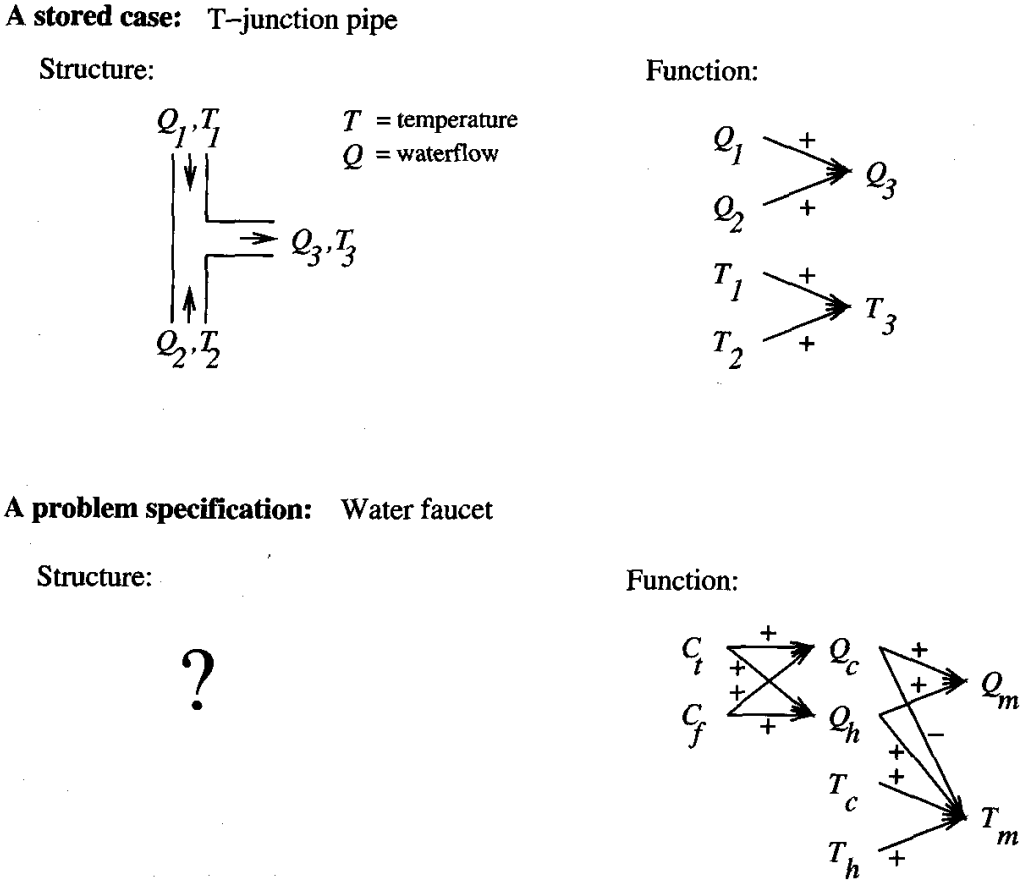

FIGURE 8.3
A stored case and a new problem. The top half of the figure describes a typical design fragment in the case library of CADET. The function is represented by the graph of qualitative dependencies among the T-junction variables (described in the text). The bottom half of the figure shows a typical design problem.

Let us consider, an example of a case-based reasoning system is CADET. CADET **each stored training example describes a function graph** along with the structure that implements it. The **CADET (CAse based DEsign Tool)** system employs case based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.

The problem setting is illustrated in Figure 8.3. Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function. New design problems are then presented by specifying the desired function and requesting the corresponding structure.

T1, T2 – Input temperature
Q1, Q2 – Input waterflow
T3 – Output temperature
Q3 – Output waterflow which increases with increasing input water flow Q1 and Q2.
'+' Lebel indicates that variable at the arrow head increases.
'-' Label indicates that variable at the arrow head decreases.
Ct – Control flow for temperature
Cf – Control signal for water flow
Qc – Flow of cold water ; Qh – Flow of hot water
Qm – Single mixed flow out
Tc – Temperature of cold water
Th – Temperature of hot water
Tm – Temperature of mixed water

Given this functional specification for the new design problem, CADET **searches its library for stored cases** whose functional descriptions match the design problem. **If an exact match is found**, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem. If **no exact match occurs**, CADET may find cases that match various subgraphs of the desired functional specification.

The process of **producing a final solution** from multiple retrieved cases can be very complex. It may require designing portions of the system from first principles, in addition to merging retrieved portions from stored cases. It may also require **backtracking on earlier choices of design subgoals** and, therefore, **rejecting cases** that were previously retrieved. CADET has very limited capabilities for combining and adapting multiple retrieved cases to form the final design.

The system must learn from the training example cases to output the structure f (x_q) that successfully implements the input function graph query x_q.

REMARKS ON LAZY AND EAGER LEARNING

Three lazy learning methods:

The k-NEAREST NEIGHBOR algorithm, locally weighted regression, and case-based reasoning, because they defer the decision of how to generalize beyond the training data until each new query instance is encountered.

One eager learning method:

The method for learning radial basis function networks, because it generalizes beyond the training data before observing the new query, committing at training time to the network structure and weights that define its approximation to the target function.

Difference between Lazy and eager learning methods:

1. **Computation time** - Lazy methods will generally require less computation during training, but more computation when they must predict the target value for a new query.
2. Differences in the **classifications** produced for new queries.
3. Lazy methods may consider the **query instance** x, when deciding how to generalize beyond the training data D. Eager methods cannot. By the time they observe the query instance x_q they have already chosen their (global) approximation to the target function.
4. Differences in the generalization **accuracy**.
5. Lazy learner has the option of (implicitly) representing the target function by a combination of many local **approximations**, whereas an eager learner must commit at training time to a single global approximation.
6. Lazy methods have the **option of selecting a different hypothesis** or local approximation to the target function for each query instance. Eager methods using the same hypothesis space are more restricted because they must commit to a **single hypothesis** that covers the entire instance space.