Unit-III

Bayesian Learning, Computational Learning Theory, and Instance based Learning

- Bayesian Learning
  - Introduction
  - Bayes Theorem
  - Bayes Theorem and Concept Learning
  - Maximum Likelihood and Least-Squared Error Hypothesis
  - Maximum Likelihood Hypothesis for Predicting Probabilities
  - Minimum Description Length Principle
  - Bayes Optimal Classifier
  - Gibbs Algorithm
  - Naïve Bayes Classifier
  - An Example: Learning to Classify Text
  - Bayesian Belief Networks
  - The EM Algorithm

Bayesian Learning, Computational Learning Theory, and Instance based Learning

- Computational Learning Theory
  - Introduction
  - Probably Learning an Approximately Correct Hypothesis
  - Sample Complexity for Finite Hypothesis Spaces
  - Sample Complexity for Infinite Hypothesis Spaces
  - The Mistake Bound Model of Learning
- Instance Based Learning
  - Introduction
  - K-Nearest Neighbor Learning
  - Locally Weighted Regression
  - Radial Basis Functions
  - Case Based Reasoning
  - Remarks on Lazy and Eager Learning

Bayesian Learning

- Bayesian reasoning provides a probabilistic approach to inference.
- It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data.
- It is important to machine learning because it provides a quantitative approach to weighing the evidence supporting alternative hypotheses.
- Bayesian reasoning provides the basis for learning algorithms that directly manipulate probabilities, as well as a framework for analyzing the operation of other algorithms that do not explicitly manipulate probabilities.

Introduction

- Bayesian learning methods are relevant to our study of machine learning for two different reasons.
  - 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.
  - Bayesian methods are important to our study of machine learning because they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities.
    - Here we analyze algorithms such as FIND-S and Candidate-Elimination algorithms to determine conditions under which they output the most probable hypothesis given the training data.
    - Bayesian methods can be used in choosing to minimize the sum of squared errors and can be used to provide an alternative error function cross entropy.
    - It can be used to analyze the inductive bias of decision tree learning algorithms that favor short decision trees and examine the closely related Minimum Description Length principle.

Introduction

- 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. Prior knowledge can be provided by asserting:
    - A prior probability for each candidate hypothesis
    - A probability distribution over observed data for each possible 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.
  - Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

Introduction

- Practical difficulties in applying Bayesian methods are:
  - They require initial knowledge of many probabilities.
  - The significant computational cost required to determine the Bayes optimal hypothesis in the general case. In certain specialized situations, this computational cost can be significantly reduced.

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 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.

Bayes Theorem

- We write P(h) to denote the initial probability that hypothesis h holds, before we have observed the training data. P(h) is called the prior probability of h and reflects any background knowledge we have about the chance that h is a correct hypothesis.
- We write P(D) to denote the prior probability that training data D will be observed.
- We write P(D/h) to denote the probability of observing data D given some world in which hypothesis h holds.
- In machine learning problems we are interested in the probability P (h/D) that h holds given the observed training data D. P (h/D) is called the **posterior probability** of h, because it reflects our confidence that h holds after we have seen the training data D.

Bayes Theorem

- Bayes Theorem:

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

- 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, the less evidence D provides in support of h.
- In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis or at least one of the maximally probable h ϵ H given the observed data D

Bayes Theorem

- The MAP (maximum a posterior) hypotheses can be determined by using Bayes theorem to calculate the posterior probability of each candidate hypothesis.

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

- In the final step P(D) is dropped because it is a constant independent of h.
- In some cases we assume every hypothesis in H is equally probable and p(h) can be considered constant.
- 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

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

An Example- Bayes Theorem

- Let us consider a medical diagnosis problem in which there are two alternative hypothesis:
  - The patient has a particular form of cancer
  - The patient does not
- The possible outcomes are positive or negative

$$P(cancer) = .008, \quad P(\neg cancer) = .992$$
$$P(\oplus|cancer) = .98, \quad P(\ominus|cancer) = .02$$
$$P(\oplus|\neg cancer) = .03, \quad P(\ominus|\neg cancer) = .97$$

- Let us consider a new patient is for whom the test result is positive. The maximum a posterior hypothesis is:

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$

$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$

An Example- Bayes Theorem

- h_{MAP} is $\neg cancer$. The exact posterior probability can also be determined by normalizing the equations so that they sum to 1.

p(Cancer/positive)=.0078/.0078+.0298=.21

- Although p(positive) is not given it can be calculated as we know that p(cancer/positive)+p($\neg cancer$/positive)=1.

An Example- Bayes Theorem

- *Product rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B

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

- *Sum rule*: probability of a disjunction of two events A and B

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

- *Bayes theorem*: the posterior probability $P(h|D)$ of h given D

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

- *Theorem of total probability*: if events $A_1, \ldots, A_n$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$, then

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Bayes Theorem and Concept Learning

- Brute-Force Bayes Concept Learning
- MAP Hypothesis and Consistent Learners

Bayes Theorem and Concept Learning

- We can use Bayes theorem as the basis for a straight forward learning algorithm that calculates the probability for each possible hypothesis then outputs the most probable.
- Here we consider a brute-force Bayesian concept learning algorithm, then compare it to the concept learning algorithm already discussed.

Brute-Force Bayes Concept Learning

- Let us assume the learner is given some sequence of training examples $\langle\langle x_1, d_1\rangle \ldots \langle x_m, d_m\rangle\rangle$ where x_i is some instance for X and d_i is the target value of x_i (i.e., $d_i = c(x_i)$).
- We assume the sequence of instances <x1,……xm> is fixed so that the training data D can be written as a sequence of target values $D = \langle d_1 \ldots d_m \rangle$.
- We now design a concept learning algorithm to output the maximum a posterior hypothesis, based on Bayes theorem.

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

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

Brute-Force MAP Learning Algorithm

- Now we must choose the values for P(h) and P(D/h).
- Let us choose them to be consistent with the following assumptions:
  - The training data 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 prior reason to believe that any hypothesis is more probable than any other.
- We assume: $$P(h) = \frac{1}{|H|} \quad \text{for all } h \text{ in } H$$
- Since we assume noise free training data the probability of observing classification d_i given h is just 1 if $d_i=h(x_i)$ and 0 if $d_i!=h(x_i)$.

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

Brute-Force MAP Learning Algorithm

- Let us now consider the first step of the MAP algorithm which computes the posterior probability p(h/D) of each hypothesis h given the observed training data D.

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

- First consider the case where h is inconsistent with the training data D.

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

Brute-Force MAP Learning Algorithm

- Now consider the case where h is consistent with the training data D.

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

Where $VS_{H,D}$ is the subset of hypothesis from H consistent with D.

$P(D)=|VS_{H,D}|/|H|$ because the sum over all hypothesis of $p(h/D)=1$

Brute-Force MAP Learning Algorithm

- One more way to derive p(D) is:

$$
\begin{aligned}
P(D) &= \sum_{h_i \in H} P(D|h_i) P(h_i) \\
&= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|} \\
&= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} \\
&= \frac{|VS_{H,D}|}{|H|}
\end{aligned}
$$

- The posterior probability under our assumption of p(h) and p(D/h) is

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

Which implies every consistent hypothesis is a MAP hypothesis

MAP Hypotheses and Consistent Learners

- The discussion we had until now translates into a statement about a general class of learners called consistent learners.
- A learning algorithm is a consistent learner if it outputs a hypothesis that commits zero errors over the training examples.
- We conclude that every consistent learner outputs a MAP hypothesis if we assume a uniform prior probability distribution over H ($p(h_i)=p(h_j)$ for all i,j) and if we assume deterministic, noise free training data (p(D/h)=1 if D and h are consistent 0 otherwise).

MAP Hypotheses and Consistent Learners

- Let us consider the FIND-S algorithm. It searches the hypothesis space H from specific to general hypotheses, outputting a maximally specific consistent hypothesis.
- We know that FIND-S will output a MAP hypothesis under the probability distributions defined already.
- FIND-S outputs a maximally specific hypothesis from the version space and it will be a MAP hypothesis relative to any prior probability distribution that favors more specific hypothesis.
- The Bayesian framework characterizes the behavior of learning algorithms even when learning algorithm does not explicitly manipulate probabilities.

MAP Hypotheses and Consistent Learners

- The discussion we had until now corresponds to a special case of Bayesian reasoning, because we considered the case where p(D/h) takes the values of only 0 and 1.
- We shall see that it is possible to model learning from noisy training data by allowing p(D/h) to take values other that 0 and 1.
- We shall also introduce into p(D/h) additional assumptions about the probability distributions that govern the noise.

Maximum Likelihood and Least Squared Error Hypothesis

- By now we understand the fact that Bayes analysis can be used to show that a particular learning algorithm outputs MAP hypothesis even though it may not use Bayes rule explicitly.
- Here we consider the problem of learning a continuous valued function.
- We shall use Bayes analysis to 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.

Maximum Likelihood and Least Squared Error Hypothesis

- Learner L considers an instance space X and hypothesis space H consisting of some class of real valued functions defined over X (i.e., h in H is a function of the form h: X□R where R is the set of real numbers).
- L has to learn an unknown target function f: X□R drawn from H.
- m training examples are provided where the target value of each example is corrupt by a random noise drawn according to a normal probability distribution.
- Each training example is a pair of the. form $<x_i,d_i>$ where $d_i=f(x_i)+e_i$.

Maximum Likelihood and Least Squared Error Hypothesis

- $f(x_i)$ is the noise free value of the target function and e_i is a random variable representing the noise.
- It is assumed that the values of e_i are drawn independently and that they are distributed according to a normal distribution with 0 mean.
- The task of the learner is to output a maximum likelihood hypothesis.

Maximum Likelihood and Least Squared Error Hypothesis

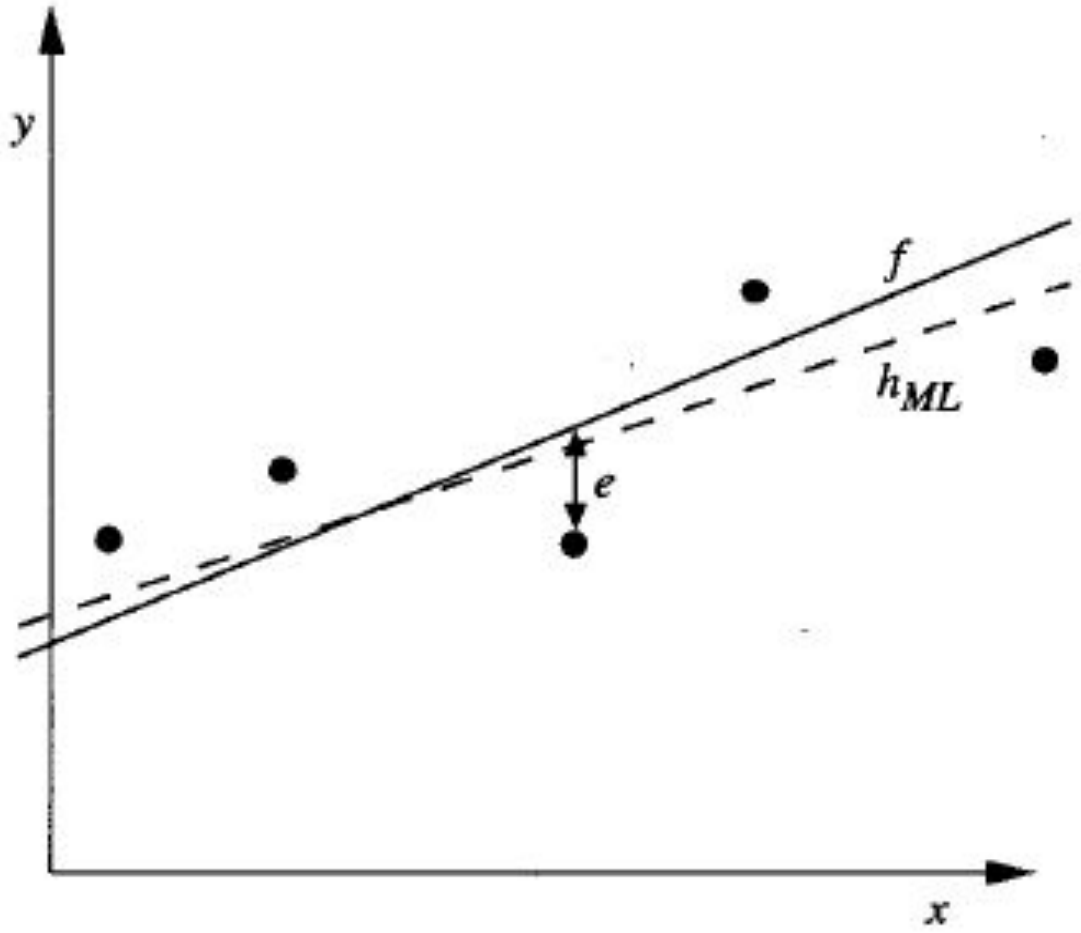


FIGURE 6.2
Learning a real-valued function. The target function f corresponds to the solid line. The training examples $\langle x_i, d_i \rangle$ are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$. The dashed line corresponds to the linear function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

Maximum Likelihood and Least Squared Error Hypothesis

- Here we try to understand two basic concepts from probability theory:
  - Probability densities and
  - Normal Distribution
- In order to discuss probabilities over continuous variable such as e we need to understand probability densities.
- We wish the total probability over all possible values of a random variable to sum to 1. In the case of continuous variables this cannot be achieved by assigning finite probability to each of the infinite set of possible values for the random variable.

Maximum Likelihood and Least Squared Error Hypothesis

- The probability density function p(x0) with respect to probability P is defined as follows:

Probability density function:

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

- We already mentioned that e is generated by a normal probability distribution which is characterized by mean μ and standard deviation σ.
- We now show that the least squared error hypothesis is the maximum likelihood hypothesis. Here we use p to refer to probability density.

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

Maximum Likelihood and Least Squared Error Hypothesis

- We know $d_i=f(x_i)+e_i$. Assuming the training examples are mutually independent given h we can write P(D/h) as the product of the various $p(d_i/h)$.

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

- Given that e_i obeys a normal distribution with o mean and unknown variance σ^2 each di must also obey normal distribution with variance σ^2 centered around the true target value $f(x_i)$ rather than 0.
- $p(d_i/h)$ can be written as a normal distribution with variance σ^2 and mean $\mu=f(x_i)$.
- The general formula for probability density of normal distribution is:

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

Maximum Likelihood and Least Squared Error Hypothesis

- Here we write the expression for the probability of di given that h is the correct description of the target function f, we will substitute μ=f(x)=h(x).

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

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

- We choose to maximize the logarithm of the above complicated function

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

Maximum Likelihood and Least Squared Error Hypothesis

- The first term in the expression is a constant independent of h and can 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 the negative quantity is equivalent to minimizing the corresponding positive quantity.

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

- Discarding the constants which are independent of h. We get

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

Maximum Likelihood and Least Squared Error Hypothesis

- There are a few limitations in the problem setting discussed.
- The analysis considers noise only in the target value of the training example and does not consider noise in the attributes describing the instances themselves.
- The analysis becomes significantly more complex as these simplifying assumptions are removed.

Maximum Likelihood Hypotheses for Predicting Probabilities

- Here we will determine maximum likelihood hypothesis for the problem of learning to predict probabilities.
- Let us consider a setting in which we wish to learn a non-deterministic (probabilistic) function f: X□{0,1}, which has two discrete output values.
- We can use a neural network whose output is the probability that f(x)=1.
- We want to learn a target function f': X□[0,1] such that f'(x)=P(f(x)=1).
- In order to obtain a maximum likelihood hypothesis for f' we must obtain an expression for P(D/h).

Maximum Likelihood Hypotheses for Predicting Probabilities

- Let us assume that the training dataset D is of the form D={<x1,d1>… <xm,dm>}.
- Let us treat both xi and di as random variables and assuming that each training example is drawn independently, wee can write P(D/h) as

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

- We also assume that the probability of encountering any particular instance xi is independent of the hypothesis h.
- When x is independent of h the above expression becomes

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

Maximum Likelihood Hypotheses for Predicting Probabilities

- We know that h is out hypothesis regarding the target function which computes the probability P(di/h,xi) of observing di=1 for a single instance xi. Therefore

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

- Let us re-express it in a more mathematical form

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

Maximum Likelihood Hypotheses for Predicting Probabilities

- Substituting the previous equation in the equation for P(D/h)

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

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

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

Maximum Likelihood Hypotheses for Predicting Probabilities

- Now we take the logarithm of the likelihood which yields

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

- The above equation shows the quantity that must be maximized in order to obtain maximum likelihood hypothesis in the given problem.
- There is a similarity between the above function and the general form of entropy function $-\sum_i p_i \log p_i$, . Therefore the negation of the above quantity is called the **cross entropy**.

Minimum Description Length Principle

- Here we recall the discussion of Occam's razor which says "choose the shortest explanation for the observed data".
- Here we consider a Bayesian perspective on this issue and a closely related principle called the Minimum Description Length (MDL) principle.
- The Minimum Description Length Principle is motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory.

Minimum Description Length Principle

- Consider the definition of h_{MAP}:

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

- The above equation can be equivalently expressed in terms of maximizing the $\log_2$.

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

- Minimizing the negative of this quantity

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

Minimum Description Length Principle

- The last statement can be interpreted as a statement that short hypotheses are preferred.
- Let us consider the problem of designing a code to transmit messages drawn at random, where the probability of encountering message i is 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.

Minimum Description Length Principle

- It was proved by Shannon and Weaver that the optimal code assigns $-\log_2 pi$ bits to encode message i.
- This number of bits required to encode message i using code C is referred to as description length of message i with respect to C which is denoted by $L_C(i)$.

Minimum Description Length Principle

- Now we will try to compare the last equation with the description just given.

  - $-\log_2 P(h)$ is the description length of h under the optimal encoding for the hypothesis space H. In other words, this is the size of the description of hypothesis h using this optimal representation. In our notation, $L_{C_H}(h) = -\log_2 P(h)$, 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. In our notation, $L_{C_{D|h}}(D|h) = -\log_2 P(D|h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h.

Minimum Description Length Principle

- Now we can rewrite the equation for h_{MAP} as:

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

Where C_H and $C_{D/h}$ are the optimal encodings for H and for D given h respectively.

- The minimum description length principle recommends choosing the hypothesis that minimizes the sum of these two description lengths.
- Assuming we use codes C1 and C2 to represent the hypothesis and the data given the hypothesis we state the MDL principle as:

Minimum Description Length principle: Choose h_{MDL} where

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

Minimum Description Length Principle

- If C_1 is the optimal encoding of hypothesis C_H and C_2 be the optimal encoding of $_{CD/h}$ then $h_{MDL}=h_{MAP}$.
- Thus the MDL principle provides a way of trading off hypothesis complexity for the number of errors committed by the hypothesis.
- It might select a shorter hypothesis that makes a few errors over a longer hypothesis that perfectly classifies the training data.
- Viewed in this light, it provides one method for dealing with the issue of overfitting the data.

Bayes Optimal Classifier

- The question that we have considered so far are “What is the most probable hypothesis given the training data?”
- The question that we would like to answer now is “What is the most probable classification of the new instance given the training data?”
- Let us consider a hypothesis space containing three hypotheses h1,h2 and h3 whose posterior probability given the training data are .4,.3 and .3 respectively. Therefore h1 is the MAP.
- A new instance x is encountered which is classified positive by h1 and negative by h2 and h3.
- Since h1 is MAP we consider the classification to be positive.

Bayes Optimal Classifier

- The most probable hypothesis in this case is different from the classification generated by MAP hypothesis.
- If the possible classification of a new instance can take on any value vj from some set V then the probability of P(vj/D) the correct classification for the new instance vj is:

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

- The optimal classification for the new instance is the value vj for which P(vj/D) is maximum.

Bayes optimal classification:

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

Bayes Optimal Classifier

- Let us look at an example:

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

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

Bayes Optimal Classifier

- Any system that classifies new instances according to the process discussed 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.
- One important property of Bayes optimal classifier is that the predictions it makes can correspond to a hypothesis not contained in H.

Gibbs Algorithm

- Bayes optimal classifier can be quite costly to apply. It computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance.
- A less optimal method is the Gibbs algorithm defined as follows:
  - Choose a hypothesis h from H at random, according to the posterior probability distribution over H.
  - Use h to predict the classification of the next instance x.
- Under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes Optimal classifier.

Naïve Bayes Classifier

- The performance of Naïve Bayes classifier has been shown to be comparable to that of neural network and decision tree learning.
- The Naïve Bayes classifier applies to learning tasks where each instance x is described by a conjugation 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.

Naïve Bayes Classifier

- The Bayesian approach is to assign the most probable target value V_{MAP} given the attributes <a1,a2….,an>.

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

We can use Bayes theorem to rewrite this expression as

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

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

- Calculating P(vj) is very easy but the calculation of P(a1,a2,….an/vj) needs the number of terms equal to the number of possible instances times the number of possible target values.
- We need to see every instance in the instance space many times in order to obtain real estimates.

Naïve Bayes Classifier

- The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value.
- The probability of observing the conjunction a1, a2.. .an, is just the product of the probabilities for the individual attributes: P(a1, a2 . . . an, / vj) = π_i P(a_i/v_j).
- Substituting the above in the previous equation we get:

Naive Bayes classifier:

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

- Where VNB denotes the target value output by the Naïve Bayes classifier and the number of distinct P(ai/vj) is the number of distinct attribute values times the number of target values.

Naïve Bayes Classifier

- Whenever the naive Bayes assumption of conditional independence is satisfied, this naive Bayes classification V_{NB} is identical to the MAP classification.
- The difference between the naive Bayes learning method and other learning methods we have considered is that there is no explicit search through the space of possible hypotheses.
- The hypothesis is formed without searching, simply by counting the frequency of various data combinations within the training examples.

An Illustrative 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

The given instance is:

$$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong \rangle$$

An Illustrative Example

- Our task is to predict the target value of the target concept PlayTennis for the given instance.
- The target value is given by the equation:

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

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

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

An Illustrative Example

- As we already know:

$$P(PlayTennis = yes) = 9/14 = .64$$
$$P(PlayTennis = no) = 5/14 = .36$$

- We can also estimate the conditional probabilities:

$$P(Wind = strong|PlayTennis = yes) = 3/9 = .33$$
$$P(Wind = strong|PlayTennis = no) = 3/5 = .60$$

- Now the calculation of V_{NB} is as follows:

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$
$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Estimating Probabilities

- In the previous example we estimated P(wind=strong/playtennis=no) by the fraction n_c/n.
- Let us assume P(wind=strong/playtennis=no)=.08 and we have a sample containing only 5 examples for which playtennis=no. Here the most probable value for n_c=0.
- Multiplying all other probabilities with 0 makes the product 0.

Estimating Probabilities

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

***m*-estimate of probability:**

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

- 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.

An Example: Learning to Classify Text

- Here we are going to learn about a general algorithm for learning to classify text based on the Naïve Bayes classifier.
- Let us now try to understand the general settings for the problem.
- Let us consider an instance space X consisting of all possible text documents.
- We are given training examples of some unknown target function f(x), which can take on any value from some finite set V.
- The task is to learn from these training examples to predict the target value for subsequent text documents.
- Here we consider the target function classifying documents as like and dislike.

An Example: Learning to Classify Text

- Here we have two design issues:
  - How to represent an arbitrary text document in terms of attribute values.
  - How to estimate the probabilities required by the Naïve Bayes classifier.
- Our approach to representing arbitrary text documents is disturbingly simple:
  - Given a text document, 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.
- We are given a set of 700 documents that are classified as dislike and 300 documents that are classified as like.

An Example: Learning to Classify Text

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 111 attribute values, corresponding to the 111 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.

Now we instantiate the equation to calculate the Naïve Bayes classification as:

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

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

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

An Example: Learning to Classify Text

- The independent assumption P(a1,a2,……,a111/vj)=π_1^{111}P(ai/vj) states in this setting that the word probabilities for one text position are independent of the words that occur in other positions given the document classification vj.
- This assumption is clearly incorrect.
- In practice the naive Bayes learner performs remarkably well in many text classification problems despite the incorrectness of this independence assumption.
- To estimate VNB using the above expression, we require estimates for the probability terms $P(v_j)$ and $P(a_i=w_k/v_j)$.

An Example: Learning to Classify Text

- We know that P(like)=.3 and P(dislike)=.7.
- Estimating the class conditional probabilities (eg., P(a1="our"/dislike)) is more problematic because we have to estimate 2*111*50000 (distinct words in English)terms for the training data.
- We make an additional reasonable assumption that the probability of encountering a specific word wk is independent of the specific word position being considered.
- Now we require only 2*50000 distinct terms of the form P(wk/vj).
- To choose a method for estimating the probability terms we adopt the m-estimate with uniform priors and m=|vocabulary|. The estimate of P(wk/vj) is

$$\frac{n_k + 1}{n + |Vocabulary|}$$

An Example: Learning to Classify Text

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

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

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.

Experimental Results

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%.

Bayesian Belief Networks

- Conditional Independence
- Representation
- Inference
- Learning Bayesian Belief Networks
- Gradient Ascent Training of Bayesian Networks
- Learning the Structure of Bayesian Networks

Bayesian Belief Networks

- Naïve Bayesian classifier makes significant use of the assumption that the values of the attributes $a_1, a_2 \ldots a_n$ are conditionally independent given the target value v.
- In many cases this conditional independence assumption is overly restrictive.
- 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.
- Here we introduce the key concepts and the representation of Bayesian belief networks.

Bayesian Belief Networks

- 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 $V(Y_i)$.
- The joint space of the set of variables Y is defined to be the cross product $V(Y_1)*V(Y_2)\ldots\ldots V(Y_n)$.
- The probability distribution over joint space is called the joint probability distribution.
- A Bayesian belief network describes the joint probability distribution for a set of variables.

Conditional Independence

- We say X is conditionally 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 $x_i \epsilon V(X)$, $y_j \epsilon V(Y)$, and $z_k \epsilon V(Z)$. We commonly write the above expression in abbreviated form as P(X/Y,Z)=P(X/Z).

- This definition can be extended to sets of variables:

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

Conditional Independence

- The given definition can be compared with the conditional independence in the definition of Naïve Bayes classifier.
- Naïve Bayes classifier assumes attribute A_1 is conditionally independent of attribute A_2 given target value V.
- The Naïve Bayes classifier to calculate $P(A_1,A_2/V)$ is as follows

$$P(A_1, A_2|V) = P(A_1|A_2, V)P(A_2|V)$$
$$= P(A_1|V)P(A_2|V)$$

ndependence

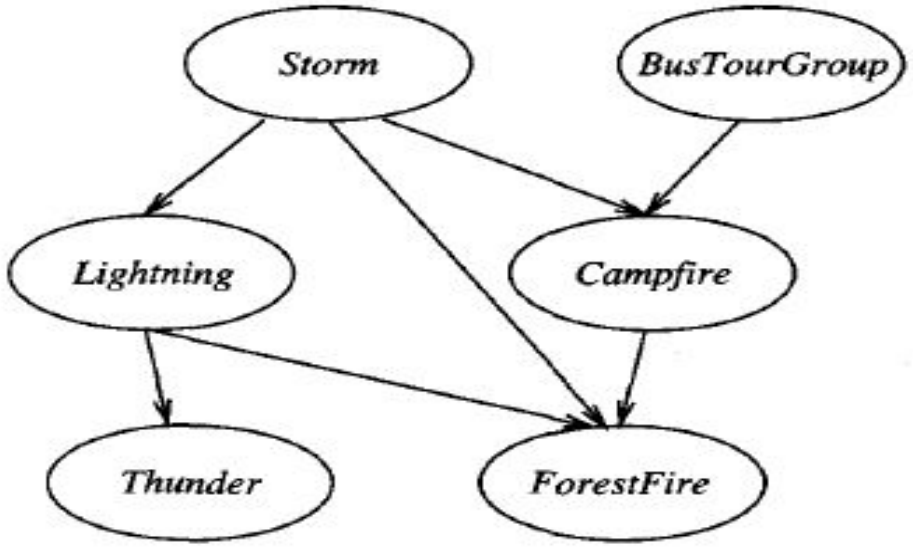
Representation

- A Bayesian Belief Network represents the joint probability distribution for a set of variables.
- Let us look at an example which represents the joint probability distribution over the Boolean variables Storm, Lightening, Thunder, ForestFire, CampFire, and BusTourGroup.
- Each variable in the joint space is represented by a node in the Bayesian Network.
- For each variable two types of information are specified:
  - 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.
  - A conditional probability table is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors.

Representation

- The joint probability for any desired assignment of values <y1,…yn> to the tuple of network variables can be computed as:

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



	S,B	S,¬B	¬S,B	¬S,¬B
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

Campfire

Representation

- One important 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.
- This conditional independence assumption is implied by the arcs in the Bayesian network.

Inference

- Bayesian network can be used to infer the value of some target variable given the observed values of other variables.
- Given that we are dealing with random variables it is not correct to assign the target variable a single determined value. We can infer a probability distribution for the target variable.
- In the general case we wish to infer the probability distribution for some variable given observed values for only a subset of other variables.
- We can use exact inference methods or approximate inference methods for calculating probabilistic inference in Bayesian networks.

Learning Bayesian Belief Networks

- Can we devise algorithms for learning Bayesian belief networks from training data?
- The network structure might be given in advance or might be inferred from the training data (difficult).
- The problem is with the case where the network structure is given but only some variable values are observable in training data.
- This problem is similar to learning weights for the hidden units in ANN.
- We use gradient ascent procedure which 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.

Gradient Ascent Training of Bayesian Networks

- 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}, which is the conditional probability that the network variable Y_i will take on the value y_{ij} given that its immediate parent U_i takes on the values given by u_{ik}, denote a single entry in one of the conditional probability tables. (Example)
- The gradient of ln P(D/h) is given by the derivative

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

Gradient Ascent Training of Bayesian Networks

- To calculate the derivative of ln P(D/h) with respect to an entry in the table given (example) we will have to calculate P(campfire=True, Storm=False, BusTourGroup=False/d) for each training example d in D.
- There is one more step before we start the gradient ascent procedure.
  - We require that as the weights w_{ijk} are updated they must remain valid probabilities in the interval [0,1].
  - We require that the sum $\sum w_{ijk}$ remains 1 for all i,k.

Gradient Ascent Training of Bayesian Networks

- These constraints can be satisfied by updating the weights in a two step process:
- First we update each weight w_{ijk} by gradient ascent

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

Where η is a constant called learning rate.

- We normalize the weights so that the constraints mentioned are satisfied.

Learning the Structure of Bayesian Networks

- Learning the Bayesian networks when the network structure is not known in advance is difficult.
- An algorithm called k_2 for learning network structure was propped by Cooper and Herskovits which performs a greedy search that trades off network complexity for accuracy over the training data.
- Constraint based approaches for learning Bayesian network structure have also been developed.
- These approaches infer independence and dependence relationship from the data and use these relationships to construct Bayesian networks.

The EM (Expectation Maximization) Algorithm

- Estimating means of K-Gaussians
- General Statement of EM Algorithm
- Derivation of K-means Algorithm

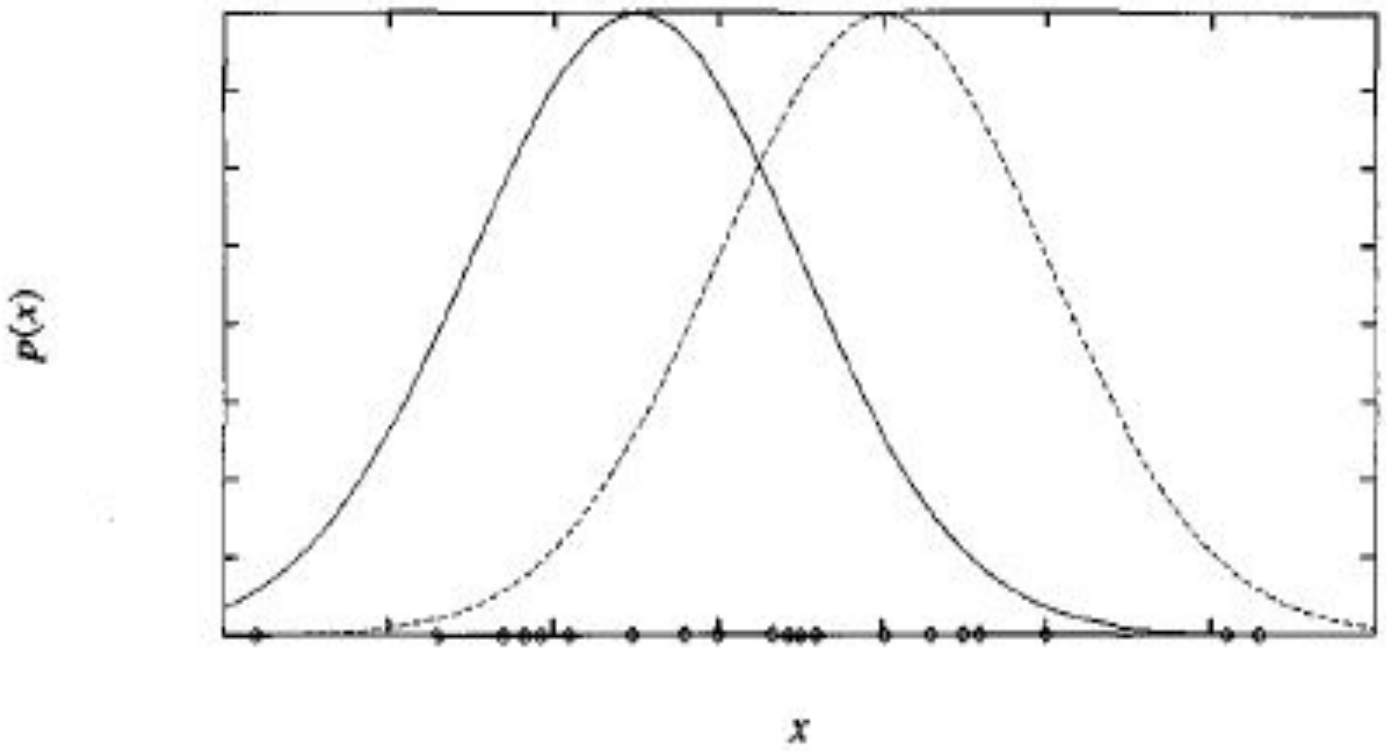
The EM Algorithm

- In many practical learning settings, only a subset of the relevant instance features might be observable.
- We have seen a few approaches while discussing about decision tree induction.
- 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.
- The EM algorithm has been used to train Bayesian belief networks as well as radial basis function networks.
- The EM algorithm is also the basis for many unsupervised clustering algorithms and it is the basis for the widely used Baum-Welch forward-backward algorithm for learning Partially Observable Markov Models.

Estimating Means of K-Gaussians

- Let us 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.
- This problem setting is for the case where k = 2 and where the instances are the points shown along the ***x*** axis.
- 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 xi is generated according to this selected distribution.

Estimating Means of K-Gaussians



Estimating Means of K-Gaussians

- We consider a 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_2,.....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(Dlh).

Estimating Means of K-Gaussians

- It is easy to calculate the maximum likelihood hypothesis for the mean of a single normal distribution given the observed data instances $x_1, x_2, \ldots\ldots, x_m$ drawn from a 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

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

- The problem here is there are a mixture of k different normal distributions and we cannot observe which instance was generated by which distribution.

Estimating Means of K-Gaussians

- In the given example we can think of the full description of each instance as the triple <xi,zi1,zi2> where xi is the observed value of the ith instance and where zi1 and zi2 indicate which of the two normal distributions was used to generate the value xi.
- xi is the observed value and zi1 and zi2 are the hidden values.
- Since zi1 and zi2 are hidden values we use EM algorithm.
- Applied to the k-means problem the EM algorithm searches for a ML hypothesis by repeatedly re-estimating the expected values of the hidden variables zij given its current hypothesis <μ1,μ2,….μk>.
- It then recalculating the ML hypothesis using these expected values for the hidden variables.

Estimating Means of K-Gaussians

- In the given problem in the figure the EM algorithm initializes h=<μ1, μ2>.
- It then iteratively re-estimates h by using the following two steps until h converges to a stationary value:
  - Calculate the expected value E[zij] of each hidden variable zij assuming the current hypothesis < μ1,μ2 > holds.
  - Calculate a new ML hypothesis h'=< μ1',μ2' > assuming the value taken on by each hidden variable zij is its expected value E[zij] calculated in the previous step. Replace the hypothesis h=<μ1,μ2> by h'=< μ1',μ2' > and iterate.

Estimating Means of K-Gaussians

- Step 1 must calculate the expected value of each zij.
- This E[zij] is just the probability that instance xi was generated by the jth Normal distribution

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

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

- In the second step we use the E[zij] calculated during Step 1 to derive a new maximum likelihood hypothesis h' = (μ1',μ2').

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

General Statement of EM algorithm

- More generally, 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(xi) of the full data (<xi,zi1,zi2>) produced by this distribution.
- In general X={x1,x2….xm}, Z={z1,z2,…,zm}, Y=XUZ.
- 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.

General Statement of EM algorithm

- Let us define a function Q(h'lh) that gives E[ln P(Ylh')] as a function of h', under the assumption that θ = h and given the observed portion X of the full data Y

$$Q(h'|h) = E[\ln p(Y|h')|h, X]$$

- In the general form the EM algorithm repeats the following two steps until convergence:

Step 1: *Estimation (E) step:* Calculate $Q(h'|h)$ using the current hypothesis h and the observed data X to estimate the probability distribution over Y.

$$Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$$

Step 2: *Maximization (M) step:* Replace hypothesis h by the hypothesis h' that maximizes this Q function.

$$h \leftarrow \underset{h'}{\operatorname{argmax}}\, Q(h'|h)$$

Computational Learning Theory

- Introduction
- Probably Learning An Approximately Correct Hypothesis
- Sample Complexity for Finite Hypothesis Spaces
- Sample Complexity for Infinite Hypothesis Spaces
- The Mistake Bound Model of Learning

Computational Learning Theory

- Two questions related to theoretical characterization of the difficulty of several types of machine learning problems and the capabilities of several machine learning algorithms. The questions are:
  - Under what conditions is successful learning possible and impossible?
  - Under what conditions is a particular learning algorithm assured of learning successfully?
- Two specific frameworks for analyzing learning algorithms are considered:
  - **Probably Approximately Correct (PAC) Framework**:- We identify classes of hypotheses that can and cannot be learned from a polynomial number of training examples and we define a natural measure of complexity for hypothesis spaces that allows bounding the number of training examples required for inductive learning.
  - **Mistake Bound Framework**:- We examine the number of training errors that will be made by a learner before it determines the correct hypothesis.

Introduction

- A few questions asked while discussing about machine learning algorithms are:
  - What are the general laws that govern machine learners?
  - Is it possible to identify classes of learning problems that are inherently difficult or easy independent of the learning algorithm?
  - Can one characterize the number of training examples necessary or sufficient to assure successful learning?
  - How is this number affected if the learner is allowed to pose queries to the trainer, versus observing a random sample of training examples?
  - Can one characterize the number of mistakes that a learner will make before learning the target function?
  - Can one characterize the inherent computational complexity of classes of learning problems?

Introduction

- We focus here on the problem of inductively learning an unknown target function, given only training examples of this target function and a space of candidate hypotheses.
- We will be chiefly concerned with questions such as how many training examples are sufficient to successfully learn the target function, and how many mistakes will the learner make before succeeding.
- It is possible to set quantitative bounds on these measures, depending on attributes of the learning problem such as:
  - the size or complexity of the hypothesis space considered by the learner
  - the accuracy to which the target concept must be approximated
  - the probability that the learner will output a successful hypothesis
  - the manner in which training examples are presented to the learner

Introduction

- Here we would like to answer questions such as:
- **Sample complexity.** How many training examples are needed for a learner to converge to a successful hypothesis?
- **Computational complexity.** How much computational effort is needed for a learner to converge to a successful hypothesis?
- **Mistake bound.** How many training examples will the learner misclassify before converging to a successful hypothesis?

Probably Learning An Approximately Correct Hypothesis

- The Problem Setting
- Error of A Hypothesis
- PAC Learnability

Probably Learning An Approximately Correct Hypothesis

- Here we consider a particular setting for the learning problem called the probably approximately correct (PAC) learning model.
- We specifying the problem setting that defines the PAC learning model, then consider the questions of how many training examples and how much computation are required in order to learn various classes of target functions within this PAC model.
- Here we restrict the discussion to the case of learning Boolean valued concepts from noise-free training data.
- Many of the results can be extended to the more general scenario of learning real-valued target functions, and some can be extended to learning from certain types of noisy data

The Problem Setting

- Let X refer to the set of all possible instances over which target functions may be defined.
- X might represent the set of all people, each described by the attributes age (e.g., young or old) and height (short or tall).
- Let ***C*** refer to some set of target concepts that our learner might be called upon to learn.
- Each target concept ***c*** in ***C*** corresponds to some subset of X, or equivalently to some Boolean-valued function ***c*** : X □ {0, 1}.

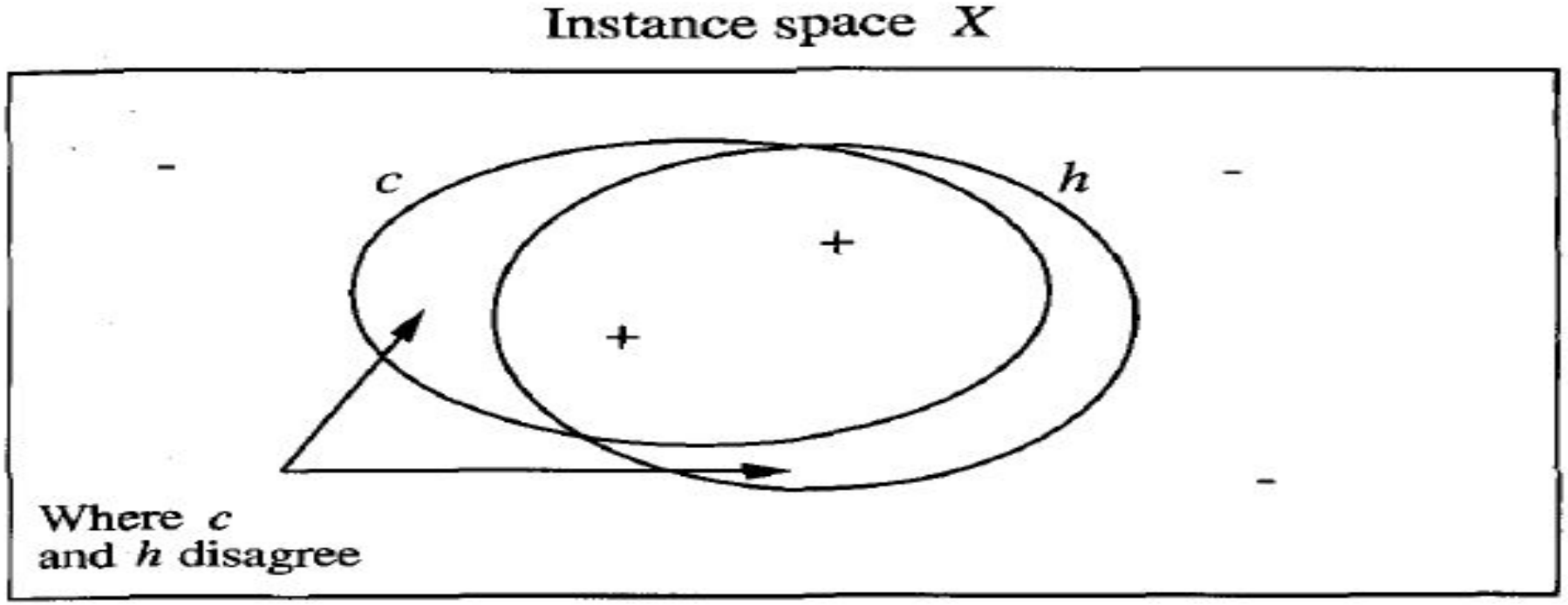
The Problem Setting

- We assume instances are generated at random from X according to some probability distribution D.
- Training examples are generated by drawing an instance x at random according to D, then presenting x along with its target value, c(x), to the learner.
- The learner L considers some set H of possible hypotheses when attempting to learn the target concept.
- After observing a sequence of training examples of the target concept c, L must output some hypothesis h from H, which is its estimate of c.
- We evaluate the success of L by the performance of h over new instances drawn randomly from X according to D, the same probability distribution used to generate the training data.

Error of a Hypothesis

Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target concept c and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$$



Error of a Hypothesis

- The error of h with respect to c is not directly observable to the learner.
- L can only observe the performance of h over the training examples, and it must choose its output hypothesis on this basis only.
- The term training error refers to the fraction of training examples misclassified by h, in contrast to the true error.
- Much of our analysis of the complexity of learning centers around the question "how probable is it that the observed training error for h gives a misleading estimate of the true $error_D(h)$?"

PAC Learnability

- Our aim here is to characterize classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation.
- We might try to characterize the number of training examples needed to learn a hypothesis h for which $error_D(h)=0$ (which is impossible).
- Given that the training examples are drawn randomly there will always be some non-zero probability that the training examples encountered by the learner will be misleading.

PAC Learnability

- To accommodate the difficulties the demands on the learner are relaxed a little.
- We will not require that the learner output a zero error hypothesis, we require that its error be bounded by some constant ϵ that can be made arbitrarily small.
- We will not require that the learner succeed for every sequence of randomly drawn training examples, we require only that its probability of failure be bounded by some constant δ that can be made arbitrarily small.
- 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.

PAC Learnability

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 Spaces

- Agnostic Learning and Inconsistent Hypothesis
- Conjunctions and Boolean Literals are PAC- Learnable
- PAC Learnability of Other Concept Classes
  - Unbiased Learners
  - K-Term DNF and k-CNF Concepts

Sample Complexity for Finite Hypothesis Spaces

- PAC-learnability is largely determined by the number of training examples required by the learner.
- The growth in the number of required training examples with problem size, called the ***sample complexity*** of the learning problem, is the characteristic that is usually of greatest interest.
- Here we present a general bound on the sample complexity for a very broad class of learners, called ***consistent learners.***
- A learner is ***consistent*** if it outputs hypotheses that perfectly fit the training data, whenever possible.

Sample Complexity for Finite Hypothesis Spaces

- To derive a bound on the number of training examples required by any consistent learner independent of any algorithm let us recall the definition of version space.

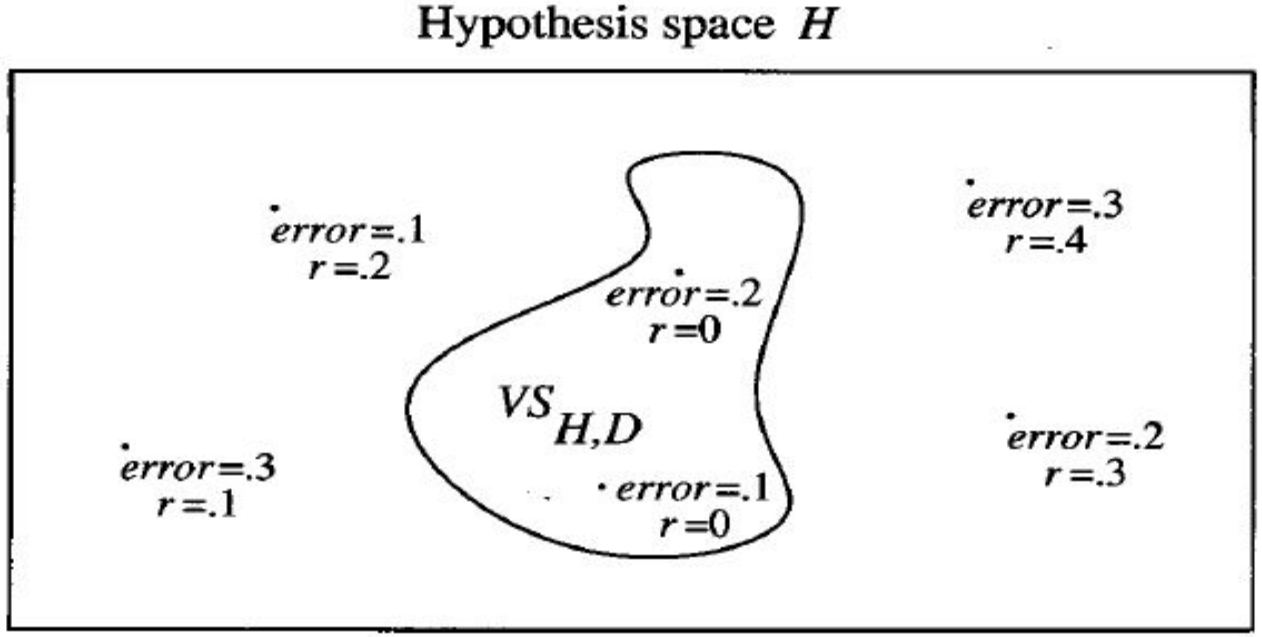
$$VS_{H,D} = \{h \in H | (\forall \langle x, c(x) \rangle \in D)\ (h(x) = c(x))\}$$

- 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 hypothesis.

Sample Complexity for Finite Hypothesis Spaces

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 $\mathcal{D}$.

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



Sample Complexity for Finite Hypothesis Spaces

Theorem 7.1. ϵ-exhausting the version space. If the hypothesis space H is finite, and D is a sequence of $m \geq 1$ independent randomly drawn examples of some target concept c, then for any $0 \leq \epsilon \leq 1$, the probability that the version space $VS_{H,D}$ is not ϵ-exhausted (with respect to c) is less than or equal to

$$|H|e^{-\epsilon m}$$

- This provides an upper bound on the probability that the version space is not ϵ-exhausted, based on the number of examples m, the allowed error ϵ and the size of H.
- Now let us use this result to determine the number of training examples required to reduce this probability of failure below some desired level δ.

$$|H|e^{-\epsilon m} \leq \delta$$

Rearranging terms to solve for m, we find

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

Agnostic Learning and Inconsistent Hypothesis

- A learner that makes no assumption that the target concept is representable by *H* and that simply finds the hypothesis with minimum training error, is often called an **agnostic** learner, because it makes no prior commitment about whether or not *C is contained in* H.
- Let D denote the particular set of training examples available to the learner, in contrast to **D,** which denotes the probability distribution over the entire set of instances.
- The $error_D(h)$ over the particular sample of training data D may differ from the true error $error_D(h)$ over the entire probability distribution **D.**

Agnostic Learning and Inconsistent Hypothesis

- Let h_{best} denote the hypothesis from ***H*** having lowest training error over the training examples.
- Now the question is how many training examples are required to ensure that the true error $error_D(h_{best})$ will not be more than $\epsilon + error_D(h_{best})$. In the previous section $error_D(h_{best})$ is zero.
- The Hoeffding bounds states that if the training error $error_D(h)$ is measured over the set D containing m randomly drawn examples then:

$$\Pr[error_{\mathcal{D}}(h) > error_D(h) + \epsilon] \leq e^{-2m\epsilon^2}$$

- This gives us a bound on the probability that an arbitrarily chosen single hypothesis has a very misleading training error.

Agnostic Learning and Inconsistent Hypothesis

- To assure the best hypothesis found by L has an error bounded, we must consider the probability that any one of the |H| hypothesis could have a large error.

$$\Pr[(\exists h \in H)(error_{\mathcal{D}}(h) > error_{D}(h) + \epsilon)] \leq |H|e^{-2m\epsilon^2}$$

- If we call the probability δ and the number of examples m required to hold δ to a required value we get:

$$m \geq \frac{1}{2\epsilon^2}(\ln|H| + \ln(1/\delta))$$

Conjunctions of Boolean Literals Are PAC-Learnable

- Now we want to determine the sample complexity and PAC-Learnability of some specific concept classes.
- Let us consider the class C of target concepts described by conjugations of Boolean literals. For example the target concept can be **old ∧ !tall.** Is C PAC Learnable?
- Let us use the equation where the hypothesis space H is identical to C to compute the number m of training examples sufficient to ensure that L will with a probability(1-δ) output a hypothesis with maximum error **ϵ**.
- In the case of conjugations of Boolean literals based on n Boolean variables the $|H|=3^n$.

Conjunctions of Boolean Literals Are PAC-Learnable

- Substituting $|H|=3^n$ in the previous equation we get:

$$m \geq \frac{1}{\epsilon}(n \ln 3 + \ln(1/\delta))$$

- If n=10 ,(1-δ)=95% and ϵ=.1 then m =1/.1(10 ln3 + ln(1/.05))=140.
- We observe that m grows linearly in the number of literals, linearly in 1/ϵ and logarithmically in 1/δ.
- The overall computational effort depends on the specific algorithm.
- The computation required is polynomial.

Theorem 7.2. PAC-learnability of boolean conjunctions. The class C of conjunctions of boolean literals is PAC-learnable by the FIND-S algorithm using $H = C$.

PAC-Learnability of Other Concept Classes

- Unbiased Learners
- K-term DNF and k-CNF Concepts

Unbiased Learners

- Now let us consider the unbiased concept class C that contains every teachable concept relative to X.
- The set C of all definable target concepts corresponds to the power set of X. $|C|=2^{|X|}$.
- Let $|X|=2^n$, then $|H|=|C|=2^{2n}$. Substituting the value of H in the equation we get:

$$m \geq \frac{1}{\epsilon}(2^n \ln 2 + \ln(1/\delta))$$

- We can see that the sample complexity for the unbiased concept class is exponential in n.

K-Term DNF and k-CNF Concepts

- It is also possible to find concept classes that have polynomial sample complexity, but nevertheless cannot be learned in polynomial time.
- Let us look at a concept class C which is k-term DNF. It is $T_1 v T_2 \ldots v T_k$.
- Each T_i is a conjugation of n Boolean attributes and their negations.
- Assuming H=C $|H|=3^{nk}$. Substituting this value in the equation we get:

$$m \geq \frac{1}{\epsilon}(nk \ln 3 + \ln(1/\delta))$$

- K-term DNF has polynomial sample complexity but the computational complexity is not polynomial.

K-Term DNF and k-CNF Concepts

- An interesting fact is that k-term DNF can be converted to K-CNF which has both polynomial sample complexity and computational complexity.
- Hence the concept class k-term DNF is PAC-Learnable by an efficient algorithm using H= K-CNF.

Sample Complexity for Infinite Hypothesis Spaces

- Shattering a set of Instances
- The Vapnik-Chervonenkis Dimension
  - Illustrative Examples
- Sample Complexity and VC Dimension
- VC Dimension for Neural Networks

Sample Complexity for Infinite Hypothesis Spaces

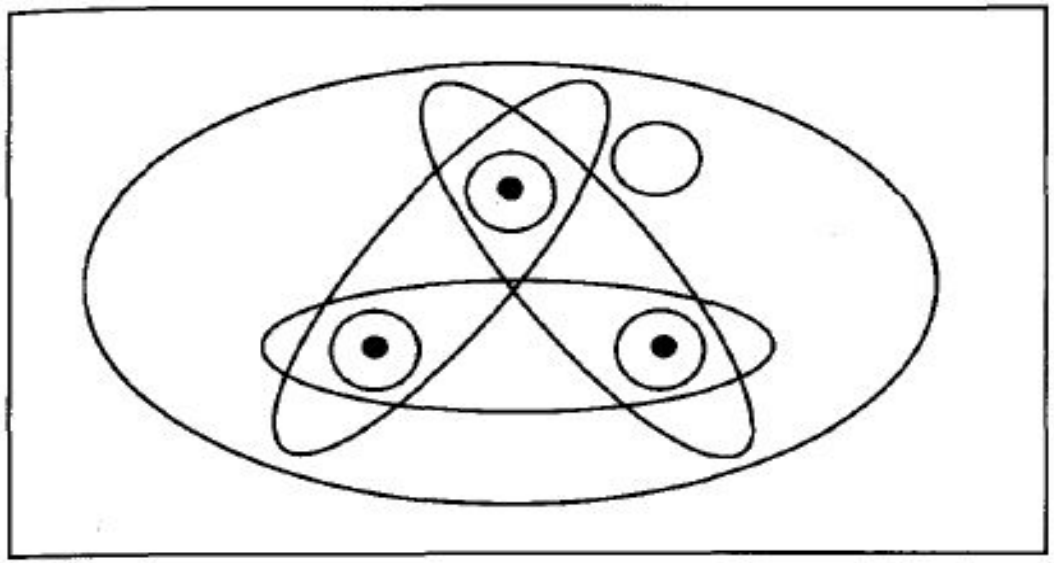
- There are two drawbacks in the previous equation used for finding the number of training examples required:
  - It can lead to quite weak bounds
  - In case of infinite hypothesis space we cannot apply the equation at all.
- Here we consider a second measure of complexity of H called Vapnik-Chervonenkis dimension of H.
- Here we can state bounds on sample complexity that use VC(H) rather than |H|.
- These bounds will be tighter and allow us to characterize the sample complexity of many infinite hypothesis spaces.

Shattering a set of Instances

- 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: A set of instances S is **shattered** by hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

- If a set of instances is not shattered by a hypothesis space then there must be some concept that can be defined over the instances but cannot be represented by the hypothesis.
- The ability of H to shatter a set of instances is thus a measure of its capacity to represent target concepts defined over these instances.



The Vapnik-Chervonenkis Dimension

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 $VC(H) \equiv \infty$.

- For any finite H VC(H)<=$\log_2|H|$. To prove this let VC(H)=d. H will require 2^d hypotheses to shatter d instances. Hence 2^d<= |H| and d=VC(H)=$\log_2|H|$.

Illustrative Example

- Let the instance space be set of real numbers X=R.
- H is the set of intervals on the real number line. H is the set of hypothesis of the for a<x<b where a and b may be any real constants.
- The VC dimension of H is at least 2.
- No subset S of size three can be shattered.
- Therefore VC(H)=2.

Illustrative Example

- What is VC(H)

Sample Complexity and VC Dimension

- The question "How many randomly drawn training examples suffice to probably approximately learn any target concept in C?" using VC(H) as a measure for complexity of H is as follows:

$$m \geq \frac{1}{\epsilon}(4\log_2(2/\delta) + 8VC(H)\log_2(13/\epsilon))$$

- The required training examples m grows logarithmically in 1/δ. It now grows log times linear in 1/ϵ, rather than linearly.
- The above equation provides an upper bound on the number of training examples.

Theorem 7.3. Lower bound on sample complexity. Consider any concept class C such that $VC(C) \geq 2$, any learner L, and any $0 < \epsilon < \frac{1}{8}$, and $0 < \delta < \frac{1}{100}$. Then there exists a distribution $\mathcal{D}$ and target concept in C such that if L observes fewer examples than

$$\max\left[\frac{1}{\epsilon}\log(1/\delta), \frac{VC(C)-1}{32\epsilon}\right]$$

then with probability at least δ, L outputs a hypothesis h having $error_{\mathcal{D}}(h) > \epsilon$.

VC Dimension for Neural Networks

- Let us consider a network G of units which form a layered directed acyclic graph.
- We can bound the VC dimension of such networks based on their graph structure and the VC dimension of the primitive units from which they are constructed.
- Let *n* be the number of inputs to the network G, and let us assume that there is just one output node.
- Let each internal unit Ni of G have at most r inputs and implement a Boolean valued function $c_i: R^r \rightarrow \{0,1\}$.

VC Dimension for Neural Networks

- We now define the G-Composition of C to be the class of all functions that can be implemented by the network G assuming individual units in G take on functions from the class C.
- The G-Composition of C is the hypothesis represented by the network G.

Theorem 7.4. VC-dimension of directed acyclic layered networks. (See Kearns and Vazirani 1994.) Let G be a layered directed acyclic graph with n input nodes and $s \geq 2$ internal nodes, each having at most r inputs. Let C be a concept class over $\mathfrak{R}^r$ of VC dimension d, corresponding to the set of functions that can be described by each of the s internal nodes. Let C_G be the G-composition of C, corresponding to the set of functions that can be represented by G. Then $VC(C_G) \leq 2ds \log(es)$, where e is the base of the natural logarithm.

VC Dimension for Neural Networks

- The bound of the VC dimension of acyclic layered networks containing s perceptrons each with r inputs is:

$$\mathrm{VC}(C_G^{perceptrons}) \leq 2(r+1)s\log(es)$$

- Substituting the above equation in the previous equation for m we get:

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

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

The Mistake Bound Model of Learning

- Computational Learning theory considers a variety of different settings and questions. The different learning settings that have been studied vary by:
  - how the training examples are generated (e.g., passive observation of random examples, active querying by the learner)
  - noise in the data (e.g., noisy or error-free)
  - the definition of success (e.g., the target concept must be learned exactly, or only probably and approximately)
  - Assumptions made by the learner (e.g., regarding the distribution of instances and whether C is contained in H)
  - the measure according to which the learner is evaluated (e.g., number of training examples, number of mistakes, total time)
- Here we consider the mistake bound model of learning, in which the learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis.

The Mistake Bound Model of Learning

- The question considered here is "How many mistakes will the learner make in its predictions before it learns the target concept?‘
- This question is significant in practical settings where learning must be done while the system is in actual use, rather than during some off-line training stage.
- This mistake bound learning problem may be studied in various specific settings.
- Here we consider the number of mistakes made before learning the target concept exactly.
- Learning the target concept exactly means converging to a hypothesis such that $(\forall x)h(x) = c(x)$.

Mistake Bound For the Find-S Algorithm

- Let us consider the hypothesis space H consisting of conjunctions of up to n Boolean literals $l_1, l_2, \ldots l_n$ and their negations.
- A straight forward implementation of FIND-S for the hypothesis space H is as follows:
  - Initialize h to the most specific hypothesis $l_1 \wedge !l_1 \wedge l_2 \wedge !l_2 \ldots\ldots \wedge l_n \wedge !l_n$.
  - For each positive training instance x
    - Remove from h any literal that is not satisfied by x
  - Output hypothesis h.

Mistake Bound For the Find-S Algorithm

- Can we prove a bound on the total number of mistakes that FIND-S will make before exactly learning the target concept c?
- To calculate the number of mistakes Find-S will make, we need only count the number of mistakes it will make misclassifying truly positive examples as negative.
- The total number of mistakes Find-S can make is at most $n + 1$.
- This number of mistakes will be required in the worst case, corresponding to learning the most general possible target concept $(\forall x)c(x) = 1$ and corresponding to a worst case sequence of instances that removes only one literal per mistake.

Mistake Bound for the Halving Algorithm

- Let us consider an algorithm (Candidate-Elimination and List-Then-Eliminate) that learns by maintaining a description of the version space, incrementally refining the version space as each new training example is encountered.
- Here we derive a worst-case bound on the number of mistakes that will be made by such a learner, for any finite hypothesis space H, assuming again that the target concept must be learned exactly.
- Let us assume the prediction is made by taking a majority vote among the hypotheses in the current version space.
- This combination of learning the version space, together with using a majority vote to make subsequent predictions, is often called the **Halving algorithm.**

Mistake Bound for the Halving Algorithm

- Learning the target concept "exactly" corresponds to reaching a state where the version space contains only a single hypothesis.
- Each mistake reduces the size of the version space by at least half.
- Given that the initial version space contains only IHI members, the maximum number of mistakes possible before the version space contains just one member is $\log_2 IHI$.
- Halving algorithm can be extended by allowing the hypothesis to vote with different weights.

Optimal Mistake Bounds

- We now analyzed the worst-case mistake bounds for Find-S and Candidate-Elimination algorithms.
- An interesting question is what is the optimal mistake bound for an arbitrary concept class C assuming H=C.
- Let $M_A(c)$ denote the maximum over all possible sequences of training examples of the number of mistakes made by ***A*** to exactly learn c.
- Let $M_A(C) \equiv \max_{c \in C} M_A(c)$
- We already showed that $M_{Find\text{-}S}(C)=n+1$ and $M_{Halving}(C)=\log_2(|C|)$ for any concept class C.

Optimal Mistake Bounds

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 $M_A(C)$.

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

For any concept class C there is an interesting relationship among the optimal mistake bound for C, the bound of the halving algorithm, and the VC dimension of C.

$$VC(C) \leq Opt(C) \leq M_{Halving}(C) \leq log_2(|C|)$$

Weighted-Majority Algorithm

- Here we consider a generalization of the HALVING algorithm called the Weighted-Majority algorithm.
- The algorithm makes predictions by taking a weighted vote among a pool of prediction algorithms (alternative hypothesis) and learns by altering the weight associated with each prediction algorithm.
- The prediction algorithm predicts the value of the target concept given an instance.
- Weighted-Majority can accommodate inconsistent training data because it does not eliminate a hypothesis that is found to be inconsistent with training examples, but rather reduces its weights.

Weighted-Majority Algorithm

---

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 $w_i \leftarrow \beta w_i$

Weighted-Majority Algorithm

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

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

Instance Based Learning

- Introduction
- K-Nearest Neighbor Learning
- Locally Weighted Regression
- Radial Basis Functions
- Case-Based Reasoning
- Remarks on Lazy and Eager Learning

Introduction

- Instance-based learning methods simply store the training examples and generalizing beyond these examples is postponed until a new instance must be classified.
- Instance-based learning includes nearest neighbor and locally weighted regression methods that assume instances can be represented as points in a Euclidean space.
- It also includes case-based reasoning methods that use more complex, symbolic representations for instances.
- Instance-based methods are sometimes referred to as "lazy" learning methods because they delay processing until a new instance must be classified.

Introduction

- A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.
- One disadvantage of instance-based approaches is that the cost of classifying new instances can be high.
- A second disadvantage to many instance-based approaches, especially nearest neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory.

K-Nearest Neighbor Learning

- K-nearest neighbor learning
- Distance-Weighted Nearest Neighbor Algorithm
- Remarks on k-nearest neighbor algorithm
- A Note on Terminology

K-Nearest Neighbor Learning

- This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard **Euclidean distance**.
- Let an arbitrary instance ***x*** be described by the feature vector:
  - $<a_1(x), a_2(x), \ldots\ldots a_n(x)>$

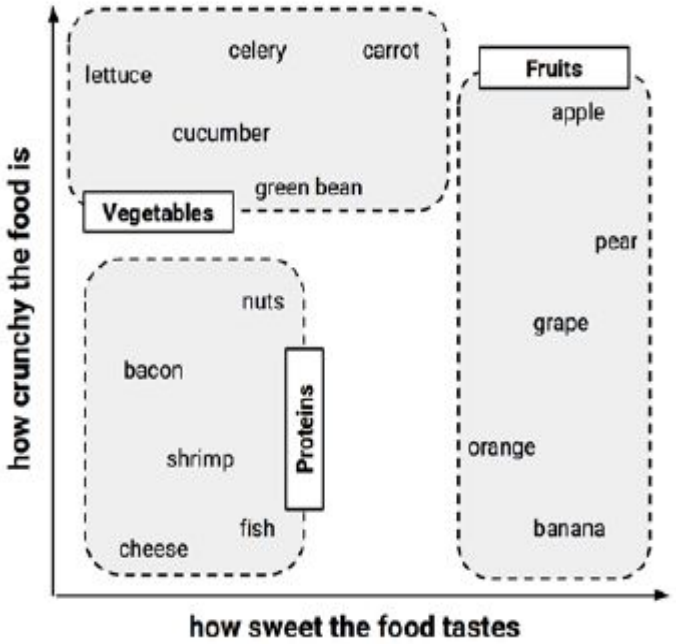
Where $a_r(x)$ denotes the value of the r^{th} attribute of instance x.

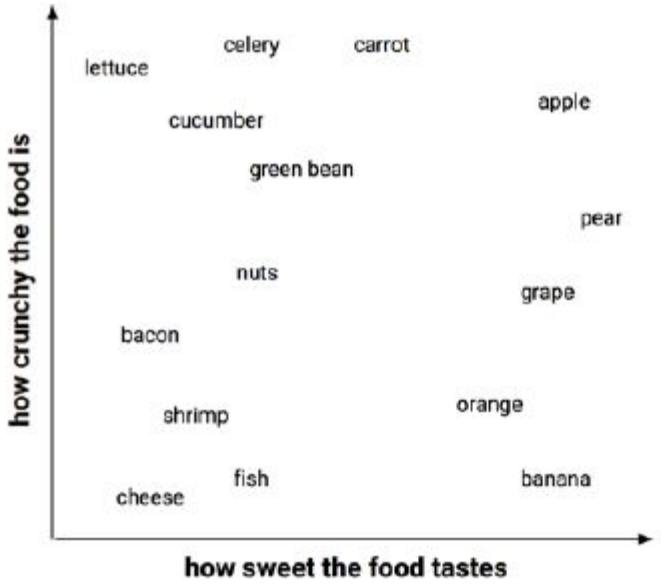
- The distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$

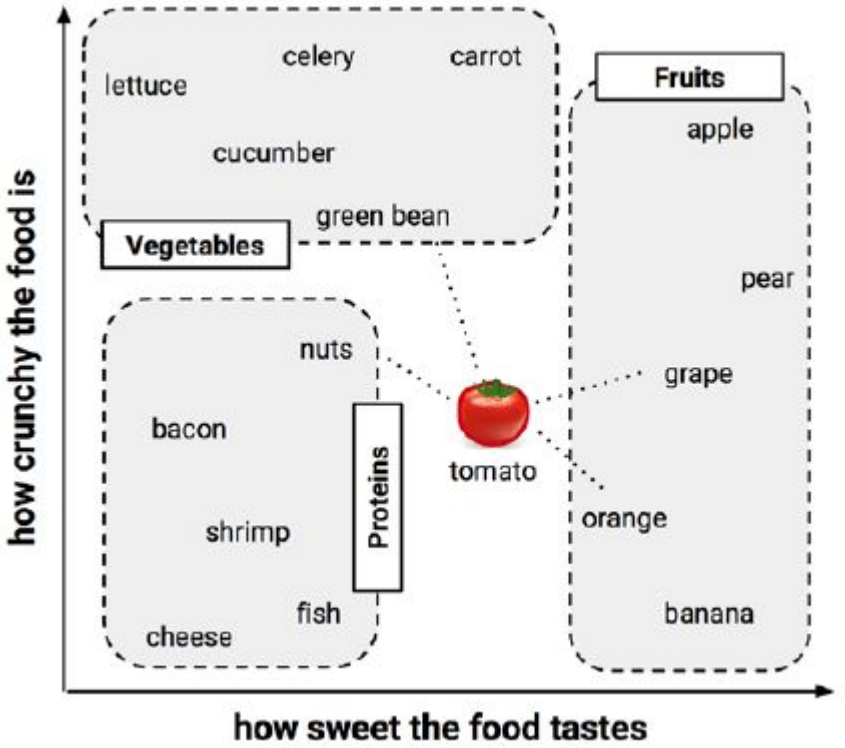
$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Example

Ingredient	Sweetness	Crunchiness	Food type
apple	10	9	fruit
bacon	1	4	protein
banana	10	1	fruit
carrot	7	10	vegetable
celery	3	10	vegetable
cheese	1	1	protein







Example

$$\text{dist}(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

$$\text{tomato}\,(\textit{sweetness} = 6, \textit{crunchiness} = 4),$$

Ingredient	Sweetness	Crunchiness	Food type	Distance to the tomato
grape	8	5	fruit	*sqrt((6 - 8)^2 + (4 - 5)^2) = 2.2*
green bean	3	7	vegetable	*sqrt((6 - 3)^2 + (4 - 7)^2) = 4.2*
nuts	3	6	protein	*sqrt((6 - 3)^2 + (4 - 6)^2) = 3.6*
orange	7	3	fruit	*sqrt((6 - 7)^2 + (4 - 3)^2) = 1.4*

$$\text{dist}(\text{tomato, green bean}) = \sqrt{(6-3)^2 + (4-7)^2} = 4.2$$

K-Nearest Neighbor Learning

Training algorithm:

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

Classification algorithm:

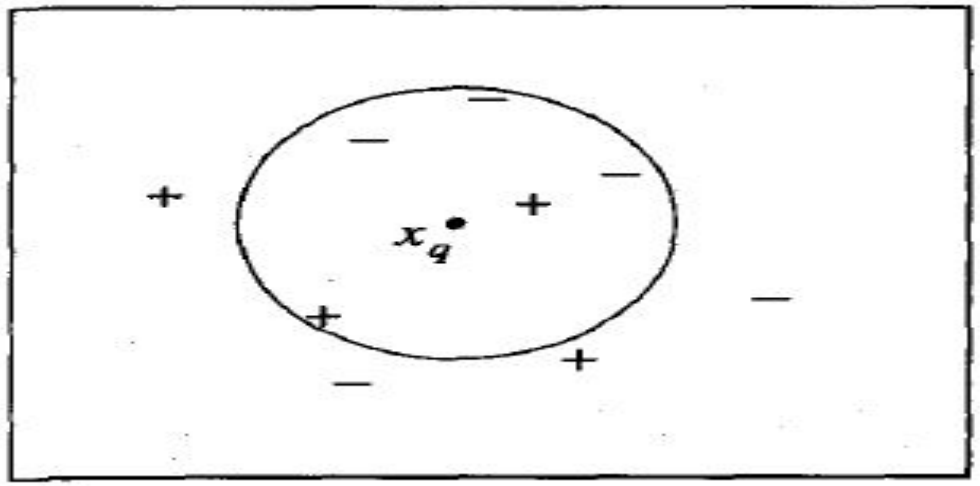
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

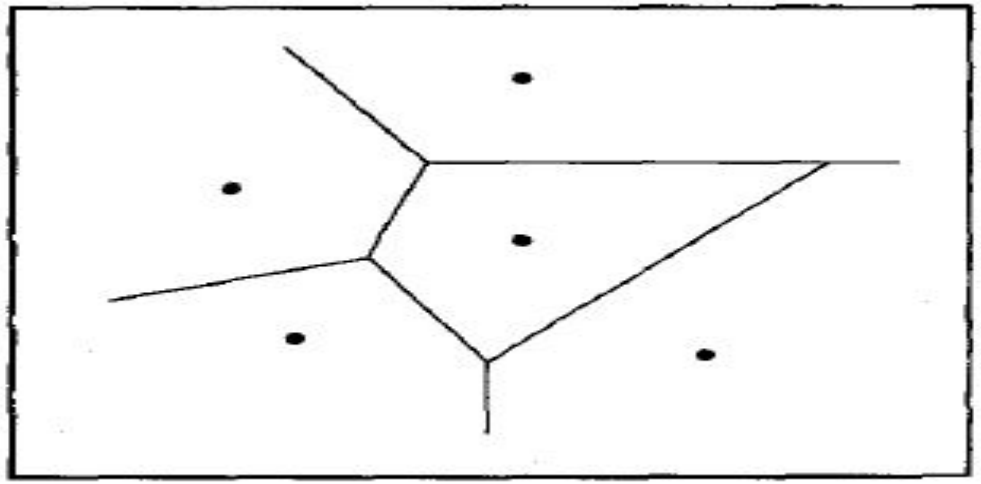
$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

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 $f : \Re^n \rightarrow V$.

K-Nearest Neighbor Learning





- For every training example, the polyhedron in the second figure indicates the set of query points whose classification will be completely determined by that training example.
- Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the **Voronoi *diagram*** of the set of training examples.

K-Nearest Neighbor Learning

- The k-Nearest Neighbor algorithm is easily adapted to approximating continuous-valued target functions.
- We have the algorithm calculate the mean value of the k nearest training examples rather than calculate their most common value.
- To approximate a real valued target function $f:R^n \square R$ the final line in the previous algorithm is replaced by:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance Weighted Nearest Neighbor Algorithm

- One obvious refinement to the 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.
- For example we might weight the vote of each neighbor according to the inverse square of its distance from x_q.

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

where

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

- In case x_q exactly matches xi then $f(x_q)=f(x_i)$.

Distance Weighted Nearest Neighbor Algorithm

- In the case the target function is real valued:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

- The case where we use k-nearest neighbors in distance weighted we call it as local method.
- In case all the examples are used we call it as global method.

Remarks on k-nearest neighbor algorithm

- The distance-weighted k-Nearest Neighbor algorithm is a highly effective inductive inference method for many practical problems.
- By taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.
- The inductive bias corresponds to an assumption that the classification of an instance ***x,*** will be most similar to the classification of other instances that are nearby in Euclidean distance.
- One important issue in k-NN is that it considers all the attributes though useful or not while calculating the distance.
- This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the ***curse of dimensionality.***

Remarks on k-nearest neighbor algorithm

- One approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances.
- Here we decide on the factor by which an attribute can be stretched or shortened by using cross validation.
- One more drastic alternative is to completely eliminate the least relevant attributes from the instance space.
- One additional practical issue in applying k-Nearest Neighbor is efficient memory indexing.

A Note on Terminology

- 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.
- The kernel function is the function K such that wi=K(d(xi,xq)).

Locally Weighted Regression

- Locally Weighted Linear Regression
- Remarks on Locally Weighted Regression

Locally Weighted Regression

- The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function f ***(x)*** at the single query point ***x*** = ***xq.***
- Locally Weighted Regression constructs an explicit approximation to f over a local region surrounding ***xq.***
- The phrase "locally weighted regression" is called:
  - ***local*** because the function is approximated based *a* only on data near the query point.
  - ***weighted*** because the contribution of each training example is weighted by its distance from the query point.
  - ***Regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance ***xq*** the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding ***xq.***
- This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.
- The description of $\hat{f}$ may then be deleted, because a different local approximation will be calculated for each distinct query instance.

Locally Weighted Linear Regression

- Let us consider the case of locally weighted regression in which the target function f is approximated near ***xq,*** using a linear function of the form:

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

- The global approximation to the target function was:

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

Locally Weighted Linear Regression

- How shall we modify this procedure to derive a local approximation rather than a global one?

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:

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

Locally Weighted Linear Regression

- Choosing criteria three and deriving the gradient descent rule we get:

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \, (f(x) - \hat{f}(x)) \, a_j(x)$$

Remarks on Locally Weighted Regression

- The locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function.
- In most cases, the target function is approximated by a constant, linear, or quadratic function.
- More complex functional forms are not often found because:
  - the cost of fitting more complex functions for each query instance is prohibitively high.
  - these simple approximations model the target function quite well over a sufficiently small sub-region of the instance space.

Radial Basis Functions

- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions.
- The learned hypothesis is a function of the form: $\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$

x_u is an instance from X, kernel function $K_u(d(x_u,x))$ is defined so that it decreases as the distance $d(x_u,x)$ increases. k is the number of kernel functions to be included.

- $K_u(d(x_u,x))$ is chosen to be a Gaussian function centered at the point x_u with some variance σ_u^2.

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

Radial Basis Functions

- The function in the previous slide can be viewed as a two layered network where the first layer computes the values of the various $k_u(d(x_u,x))$ and the second layer computes a linear combination of these first layer unit values.

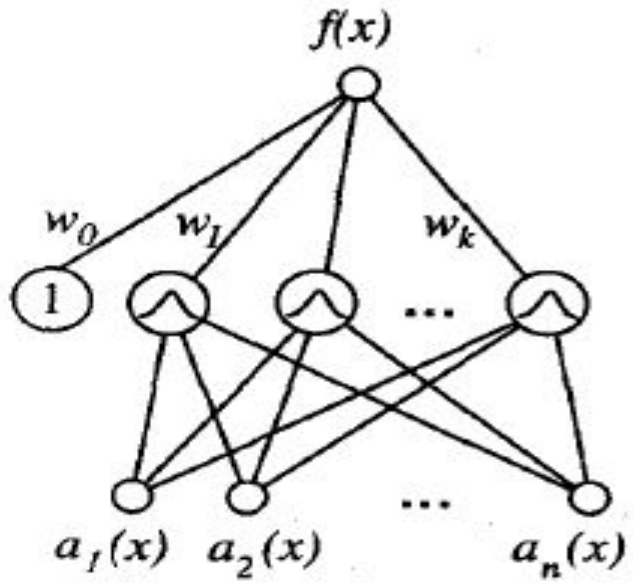


FIGURE 8.2
A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u. Therefore, its activation will be close to zero unless the input x is near x_u. The output unit produces a linear combination of the hidden unit activations.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

Radial Basis Functions

- The RBF networks are trained in a two stage process:
  - 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 $K_u(d(x_u,x))$.
  - The weights w_u are trained to maximize the fit of the network to the training data using the given global error criterion.
- Alternative methods have been proposed for choosing an appropriate number of hidden units or kernel functions:
  - Allocation of a Gaussian kernel function for each training example <xi,f(xi)>, centering this Gaussian at point xi.
  - Choosing a kernel function that is smaller than the number of training examples.

Radial Basis Functions

- Radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions.
- The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular center and width.
- The network can be viewed as a smooth linear combination of many local approximations to the target function.
- One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with backpropogation.
- The input layer and the output layer of an RBF are trained separately.

Case-Based Reasoning

- Instance based methods such as k-nearest neighbor and locally weighted regression share three key properties:
  - They defer the decision of how to generalize beyond the training data until a new query instance is observed.
  - They classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.
  - They represent instances **as** real-valued points in an n-dimensional Euclidean space.
- Case-based reasoning (CBR) is a learning paradigm based on the first two of these principles, but not the third.

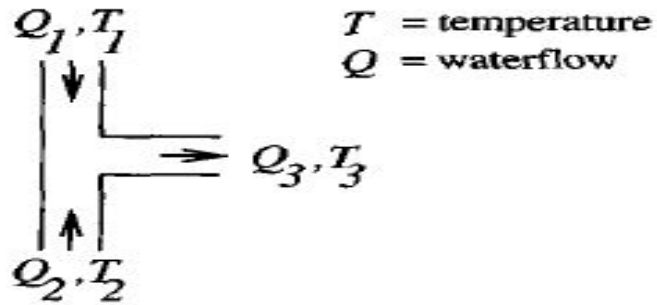
Case-Based Reasoning

- In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.
- Let us look at an example the CADET system to understand case-based reasoning systems better.
- CADET employs case based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
- It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.

Case-Based Reasoning

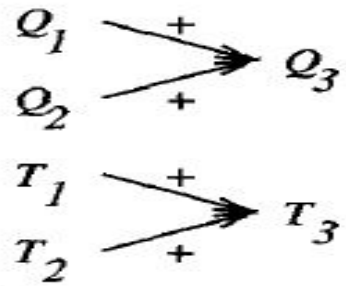
A stored case: T–junction pipe

Structure:



T = temperature
Q = waterflow

Function:

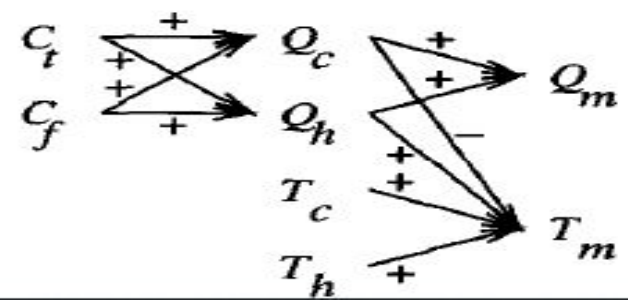


A problem specification: Water faucet

Structure:

?

Function:



Case-Based Reasoning

- 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.
- **CADET** searches for subgraph isomorphism between the two function graphs, so that parts of a case can be found to match parts of the design specification.

Case-Based Reasoning

- The system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases.
- It uses general knowledge about physical influences to create these elaborated function graphs.
- For example, it uses a rewrite rule that allows it to rewrite the influence:

$$A \xrightarrow{+} B$$

as

$$A \xrightarrow{+} x \xrightarrow{+} B$$

Case-Based Reasoning

- In CADET problem the target function f maps function graphs to the structures that implement them.
- Each stored training example (x, f (x)) is a pair that describes some function graph x and the structure f ***(x)*** that implements x.
- The system must learn from the training example cases to output the structure f (xq) that successfully implements the input function graph query xq.

Case-Based Reasoning

- The generic properties of case-based reasoning systems that distinguish them from approaches such as k-Nearest Neighbor are:
  - Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET.
  - Multiple retrieved cases may be combined to form the solution to the new problem.
  - There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving.

Remarks on Lazy and Eager Learners

- We have discussed about three lazy learning methods:
  - K-Nearest Neighbor Algorithm
  - Locally Weighted Regression
  - Case-based Reasoning
- We also discussed about one eager learning method:
  - Radial basis Function Networks
- Now the question is “Are there important differences in what can be achieved by lazy and eager learners?”
- We distinguish them keeping in view two kinds of differences:
  - Differences in computation time
  - Differences in the classifications produced for new queries

Remarks on Lazy and Eager Learners

- As far as the computation time is concerned:
  - Lazy learners require less time during training but more time during prediction of the target value.
  - Eager learners require more time during training but less time during prediction of the target value.
- The key difference between lazy learners and eager learners as far as inductive bias is concerned are:
  - Lazy methods may consider the query instance ***xq,*** when deciding how to generalize beyond the training data D.
  - Eager methods cannot. By the time they observe the query instance ***xq,*** they have already chosen their (global) approximation to the target function.

Remarks on Lazy and Eager Learners

- How does this distinction affect the generalization accuracy of the learner?
  - In the case of lazy learners for each query xq it generalizes from the training data by choosing a new hypothesis based on the training examples near xq.
  - The eager learners use the same hypothesis of linear functions for each query.
- Can we create eager methods that use multiple local approximations to achieve the same effects as lazy local methods?
- Radial basis function networks can be seen as one attempt to achieve this but can not be get the same effect of lazy learners.