MACHINE LEARNING

NOTES MATERIAL

UNIT 3

For

B. TECH (CSE)

3rd YEAR – 2nd SEM (R18)

RAVIKRISHNA B

DEPARTMENT OF CSE

VIGNAN INSTITUTE OF TECHNOLOGY & SCIENCE
DESHMUKHI

UNIT - III

Bayesian learning – Introduction, Bayes theorem, Bayes theorem and concept learning, Maximum Likelihood and least squared error hypotheses, maximum likelihood hypotheses for predicting probabilities, minimum description length principle, Bayes optimal classifier,

Gibs algorithm, Naïve Bayes classifier, an example: learning to classify text, Bayesian belief networks, the EM algorithm.

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

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

Bayesian Learning:

- Bayesian reasoning provides a probabilistic approach to inference. It's based on the assumption that the quantities of interest are governed by probability distributions. 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.
- Bayesian learning methods are relevant to our study of machine learning for two different reasons. First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems. We discuss its application to the problem of learning to classify text documents such as electronic news articles. For such learning tasks, the naive Bayes classifier is among the most effective algorithms known.
- The second reason that Bayesian methods are important to our study of machine learning is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities. For example, in this chapter we analyze algorithms such as the FIND-S and CANDIDATE ELIMINATION algorithms to determine conditions under which they output the most probable hypothesis given the training data.
- Bayesian analysis to justify a key design choice in neural network learning algorithms: choosing to minimize the sum of squared errors when searching the space of possible neural networks. We also derive an alternative error function, cross entropy, that is more appropriate than sum of squared errors when learning target functions that predict probabilities.

Features of Bayesian learning methods include:

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that eliminate a hypothesis if it is found to be inconsistent with any single example.
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting a prior probability for each candidate hypothesis, and 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.

Bayes Theorem Example:

Sample Space for Events A and B:

A holds	T	T	F	F	T	F	T
B holds	T	F	T	F	T	F	F

P(A)=4/7 P(B)=3/7 P(B|A)=2/4 P(A|B)=2/3

Check Bayes Theorem with below computations:

P(B|A)= P(A|B)P(B)/P(A) = (2/3 * 3/7) / 4/7 = 2/4 (Check with Table Data.. It's Correct)

P(A|B)=P(B|A)P(A)/P(B) = (2/4 * 4/7) / 3/7 = 2/3 (Check with Table Data.. It's Correct)

BAYES THEOREM:

- In machine learning we are often interested in determining the best hypothesis from some space H, given the observed training data D. Bayes theorem provides a direct method for calculating such probabilities. More precisely, 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.
- To define Bayes theorem precisely, let us first introduce a little notation.
  P(h): Initial Probability that hypothesis h holds, before we have observed the training data.
  P(h) is also called the prior probability of h and may reflect any background knowledge we have about the chance that h is a correct hypothesis.
- P(D) to denote the prior probability that training data D will be observed (i.e., the probability of D given no knowledge about which hypothesis holds).
- 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). 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. Notice the posterior probability P(h|D) reflects the influence of the training data D, in contrast to the prior probability P(h) , which is independent of D.

- Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability P(hlD), from the prior probability P(h), together with P(D) and P(D(h).

  $P(h)$: Prior Probability of hypothesis h

  $P(D)$: Prior Probability of training data D

  $P(h \mid D)$: Probability of h given D (Posterior Density)

  $P(D \mid h)$: Probability of D given h (likelihood of D given h)

  The Goal of Bayesian learning is to find the most probable hypothesis given training data

  (Maximum A Posteriori Hypothesis h_{MAP}

Bayes Theorem:

- Bayes' theorem is a way to figure out conditional probability. Conditional probability is the probability of an event happening, given that it has some relationship to one or more other events
- Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability $P(h \mid D)$, from the prior probability P(h), together with P(D) and P(D(h).

$$P(h \mid D) = \frac{P(D \mid h)\, P(h)}{P(D)}$$

- The learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D (or at least one of the maximally probable if there are several). Any such maximally probable hypothesis is called a MAXIMUM A POSTERIORI (MAP) HYPOTHESIS. We can determine the MAP hypotheses by using Bayes theorem to calculate the posterior probability of each candidate hypothesis. More precisely, we will say that MAP is a MAP hypothesis provided.

$$\begin{aligned} h_{MAP} &= \underset{h \in H}{argmax}\, P(h \mid D) \\ &= \underset{h \in H}{argmax}\, \frac{P(h \mid D)\, P(h)}{P(D)} \\ &= \underset{h \in H}{argmax}\, P(h \mid D)\, P(h) \quad \ldots\ldots (1) \end{aligned}$$

- Notice in the final step above we dropped the term P(D) because it is a constant independent of h.
- In some cases, we will assume that every hypothesis in H is equally probable a priori,

$$P(h_i = h_j) \quad for\ all\ h_i\ and\ h_j\ in\ H\ .$$

- In this case we can further simplify Equation (1) and need only consider the term $P(D|h)$ to find the most probable hypothesis. $P(D|h)$ is often called the likelihood of the data D given h, and any hypothesis that maximizes $P(D|h)$ is called a maximum likelihood (ML) hypothesis, h_{ML}.

$$h_{ML} = \underset{h \in H}{argmax}\, P(h|D)P(h) \quad \text{..... (2)}$$

Basic Formulas for Probabilities:

- Product Rule: probability P(A^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:

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

- Bayes Theorem: the Posterior Probability of P(h|D) of h given D is:

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

- Theorem of Total Probability: if events A1, ..., An are mutually exclusive with

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

Example on Bayes' Theorem:

To illustrate Bayes rule, consider a medical diagnosis problem in which there are two alternative hypotheses:

(1) that the patient has a- articular form of cancer and

(2) that the patient does not. The available data is from a particular laboratory test with two possible outcomes: +ve (positive) and -ve (negative).

We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease. The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present.

$$P(Cancer) = 0.008 \quad P(\sim Cancer) = 0.992$$
$$P(+/Cancer) = 0.98 \Rightarrow P(-/Cancer) = 1 - 0.98 = 0.02$$
$$P(+/\sim Cancer) = 0.03 \Rightarrow P(-/\sim Cancer) = 1 - 0.03 = 0.97$$
$$A\ patient\ takes\ a\ lab\ test\ and\ comes\ back\ Positive.$$

$$P(Cancer/+) = \frac{P(+/Cancer)P(Cancer)}{P(+)}$$

$$P(\sim Cancer/+) = \frac{P(+/\sim Cancer)P(\sim Cancer)}{P(+)}$$

$$Note\ that\ P(E1/A) = \frac{P(A/E1)P(E1)}{P(A/E1)P(E1) + P(A/E2)P(E2)}$$

$$P(+) = P(+/Cancer)P(Cancer) + P(+/\sim Cancer)P(\sim Cancer)$$
$$= (0.98*0.008) + (0.03*0.992)$$
$$= 0.00784 + 0.02976$$
$$= 0.0376$$

$$P(Cancer/+) = \frac{P(+/Cancer)P(Cancer)}{P(+)}$$

$$P(Cancer/+) = \frac{0.98*0.008}{0.0376} = \frac{0.00784}{0.0376} = 0.2085$$

$$P(\sim Cancer/+) = \frac{P(+/\sim Cancer)P(\sim Cancer)}{P(+)}$$

$$P(\sim Cancer/+) = \frac{0.03*0.98}{0.0376} = 0.7819$$

Example 2:

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition.

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

$$P(Yes / Sunny) = \frac{P(Sunny / Yes) * P(Yes)}{P(Sunny)}$$

$$P(Sunny / Yes) = \frac{3}{9} = 0.33$$

$$P(Yes) = \frac{9}{14} = 0.64$$

$$P(Sunny) = \frac{5}{14} = 0.36$$

$$\Rightarrow P(Yes / Sunny) = \frac{P(Sunny / Yes) * P(Yes)}{P(Sunny)}$$

$$\Rightarrow P(Yes / Sunny) = \frac{0.33 * 0.64}{0.36} = 0.60$$

$$\Rightarrow P(Yes / Sunny) = 0.60$$

Here we have has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

What are the Pros and Cons of Naive Bayes?

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Applications of Naive Bayes Algorithms

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not.

BAYES THEOREM AND CONCEPT LEARNING:

Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, we can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable. We consider such a brute-force Bayesian concept learning algorithm, then compares it to concept learning algorithms.

Brute-Force Bayes Concept Learning:

Consider the concept learning problem, assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept c: X → {0,1}. we assume that the learner is given some sequence of training examples $\langle\langle x_1, d_1\rangle,...,\langle x_m, d_m\rangle\rangle$, where x_i is some instance from X and where d_i is the target value of x_i, i.e., $d_i = c(x_i)$.

we assume the sequence of instances $\langle x_1...x_m\rangle$ is held fixed, so that the training data D can be written simply as the sequence of target values $D = \langle d_1...d_m\rangle$.

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

BRUTE-FORCE MAP LEARNING algorithm:

1. For each hypothesis h in H, calculate the posterior probability $P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$

2. Output the hypothesis h_{MAP} with the highest posterior probability $h_{MAP} = \underset{h \in H}{argmax}\, P(h \mid D)$

This algorithm may require significant computation, because it applies Bayes theorem to each hypothesis in H to calculate P(h | D).

We may choose the probability distributions P(h) and P(D1h) in any way we wish, to describe our prior knowledge about the learning task. Here let us choose them to be consistent with the following assumptions:

1. The training data D is noise free ($i.e., d_i = c(x_i)$).

2. The target concept c is contained in the hypothesis space H

3. We have no a priori reason to believe that any hypothesis is more probable than any other.

Furthermore, because we assume the target concept is contained in H we should require that these prior probabilities sum to 1. Together these constraints imply that we should choose

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

P(D | h) is the probability of observing the target values $D = \langle d_1, ..., d_m \rangle$ for the fixed set of instances $\langle x_1, ..., x_m \rangle$, given a world in which hypothesis h holds (i.e., given a world in which h is the correct description of the target concept c). Since we assume noise-free training data, the probability of observing classification di given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

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

In other words, the probability of data D given hypothesis h is 1 if D is consistent with h, and 0 otherwise. Given these choices for P(h) and for P(Dlh) we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm. Let us consider the first step of this algorithm, which uses Bayes theorem to compute the posterior probability P(h1D) of each hypothesis h given the observed training data D.

Recalling Bayes theorem, we have

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

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

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

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

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

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

where $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D (i.e., $VS_{H,D}$ is the version space of H with respect to D as defined in Chapter 2). It is easy to verify that $P(D) = \frac{|VS_{H,D}|}{|H|}$ above, because the sum over all hypotheses of $P(h|D)$ must be one and because the number of hypotheses from H consistent with D is by definition $|VS_{H,D}|$. Alternatively, we can derive $P(D)$ from the theorem of total probability (see Table 6.1) and the fact that the hypotheses are mutually exclusive (i.e., $(\forall i \neq j)(P(h_i \wedge h_j) = 0)$)

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

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

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

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

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

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

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

MAP Hypotheses and Consistent Learners:

The MAP analysis shows that in the given setting, every hypothesis consistent with D is a MAP hypothesis. This statement translates directly into an interesting statement about a general class of learners that we might call consistent learners. We will say that a learning algorithm is a consistent learner provided it outputs a hypothesis that commits zero errors over the training examples. Given the above analysis, we can conclude that every consistent learner output a MAP hypothesis, if we assume a uniform prior probability distribution over H

$$\text{i.e., } P(h_i) = P(h_j) \text{ for all i, j,}$$

and if we assume deterministic, noise free training data

$$i.e., P(D|h) = \begin{cases} 1 & \text{if } D \text{ and } h \text{ are consistent} \\ 0 & \text{otherwise} \end{cases}$$

MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES:

we consider the problem of learning a continuous-valued target function-problem faced by many learning approaches such as neural network learning, linear regression, and polynomial curve fitting.

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis.

Consider the following problem setting.

Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X (i.e., each h in H is a function of the form $h: X \rightarrow R$, where R represents the set of real numbers).

The problem faced by L is to learn an unknown target function $f: X \rightarrow R$ drawn from H.

A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution.

More precisely, each training example is a pair of the form (xi, di)

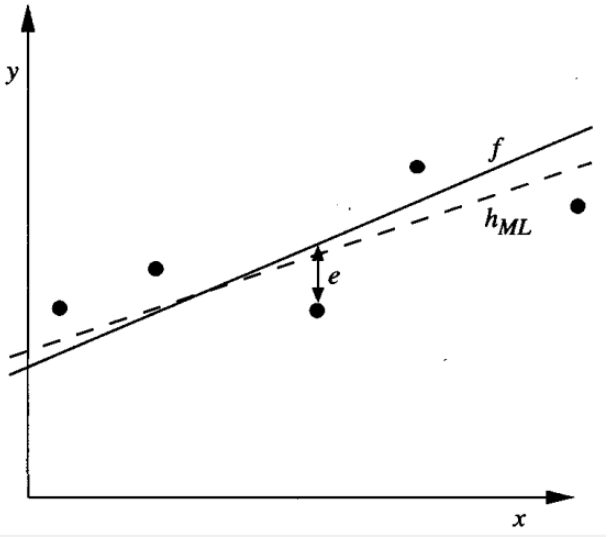
where $d_i = f(x_i) + e_i$.

Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.

It is assumed that the values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean.

The task of the learner is to output a maximum likelihood hypothesis, or, equivalently, a MAP hypothesis assuming all hypotheses are equally probable a priori.

A simple example of such a problem is learning a linear function, though our analysis applies to learning arbitrary real-valued functions.



The figure illustrates Learning a real-valued function. The target function f corresponds to the solid line. The training examples (x_i, d_i) 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 x training examples.

a linear target function f depicted by the solid line, and a set of noisy training examples of this target function. The dashed line corresponds to the hypothesis h_{ML} with least-squared training

error, hence the maximum likelihood hypothesis. Notice that the maximum likelihood hypothesis is not necessarily identical to the correct hypothesis, f, because it is inferred from only a limited sample of noisy training data.

Given this background we now return to the main issue: showing that the least-squared error hypothesis is, in fact, the maximum likelihood hypothesis within our problem setting. We will show this by deriving the maximum likelihood hypothesis starting with our earlier definition Equation (6.3), but using lower case p to refer to the probability density

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

As before, we assume a fixed set of training instances $\langle x_1 \ldots x_m \rangle$ and therefore consider the data D to be the corresponding sequence of target values $D = \langle d_1 \ldots d_m \rangle$. Here $d_i = f(x_i) + e_i$. 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 the noise e_i obeys a Normal distribution with zero mean and unknown variance σ^2, each d_i must also obey a Normal distribution with variance σ^2 centered around the true target value $f(x_i)$ rather than zero. Therefore $p(d_i|h)$ can be written as a Normal distribution with variance σ^2 and mean $\mu = f(x_i)$. Let us write the formula for this Normal distribution to describe $p(d_i|h)$, beginning with the general formula for a Normal distribution from Table 5.4 and substituting the appropriate μ and σ^2. Because we are writing the expression for the probability of d_i given that h is the correct description of the target function f, we will also substitute $\mu = f(x_i) = h(x_i)$, yielding

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

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

We now apply a transformation that is common in maximum likelihood calculations: Rather than maximizing the above complicated expression we shall choose to maximize its (less complicated) logarithm. This is justified because $\ln p$ is a monotonic function of p. Therefore maximizing $\ln p$ also maximizes p.

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

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

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

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

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

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

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2 \qquad (6.6)$$

Thus, Equation (6.6) shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values di and the hypothesis predictions h(xi). This holds under the assumption that the observed training values di are generated by adding random noise to the true target value, where this random noise is drawn independently for each example from a normal distribution with zero mean.

MINIMUM DESCRIPTION LENGTH PRINCIPLE:

Occam's razor: Prefer the simplest hypothesis that fits the data.

William of Occam was one of the first to discuss the question, around the year 1320, so this bias often goes by the name of Occam's razor.

Recall from this discussion of Occam's razor, a popular inductive bias that can be summarized as "choose the shortest explanation for the observed data." In that chapter we discussed several arguments in the long-standing debate regarding Occam's razor. 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. Consider again the now familiar definition of h_{MAP}.

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

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

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

or alternatively, minimizing the negative of this quantity

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

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. Clearly, to minimize the expected code length we should assign shorter codes to messages that are more probable. We will refer to the number of bits required to encode message i using code C as the description length of message i with respect to C.

- $-\log_2 P(h)$ 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.
    - Therefore we can rewrite Equation (6.16) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

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

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

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths. Of course to apply this principle in practice we must choose specific encodings or representations appropriate for the given learning task. Assuming we use the codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis, we can state the MDL principle as

Minimum Description Length principle: Choose h_{MDL} where

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

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H, and if we choose C_2 to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$.

BAYES OPTIMAL CLASSIFIER:

So far we have considered the question "what is the most probable *hypothesis* given the training data?" In fact, the question that is often of most significance is the closely related question "what is the most probable *classification* of the new instance given the training data?" Although it may seem that this second question can be answered by simply applying the MAP hypothesis to the new instance, in fact it is possible to do better.

To develop some intuitions consider a hypothesis space containing three hypotheses, h_1, h_2, and h_3. Suppose that the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus, h_1 is the MAP hypothesis. Suppose a new instance x is encountered, which is classified positive by h_1, but negative by h_2 and h_3. Taking all hypotheses into account, the probability that x is positive is .4 (the probability associated with h_1), and the probability that it is negative is therefore .6. The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis.

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

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

The optimal classification of the new instance is the value v_j, for which $P(v_j|D)$ is maximum.

Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{6.18}$$

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

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

therefore

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

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

and

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

GIBBS ALGORITHM:

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply. The expense is due to the fact that it computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance. An alternative, less optimal method is the Gibbs algorithm (Opper and Haussler 1991), defined as follows:

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H.

2. Use h to predict the classification of the next instance x.

Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution.

It can be shown that under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier.

More precisely, the expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner. Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier.

NAIVE BAYES CLASSIFIER:

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

In some domains its performance has been shown to be comparable to that of neural network and decision tree learning.

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

The Bayesian approach to classifying the new instance is to assign the most probable target value, v_{MAP}, given the attribute values $\langle a_1, a_2, \ldots a_n \rangle$ that describe the instance.

$$v_{MAP} = \arg\max_{v \in V} P(v_j \mid a_1, a_2, \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \arg\max_{v \in V} \frac{P(a_1, a_2, \ldots a_n \mid v_j) P(v_j)}{P(a_1, a_2, \ldots a_n)}$$

$$\Rightarrow v_{MAP} = \arg\max_{v \in V} P(a_1, a_2, \ldots a_n \mid v_j) P(v_j) \quad \ldots\ldots\ldots(1)$$

We can estimate each of the $P(v_j)$ simply by counting the frequency with which each target value v_j occurs in the training data.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $\langle a_1, a_2, \ldots a_n \rangle$ is just the product of the probabilities for the individual attributes:

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

Substituting this into Equation(1), we have the approach used by the naive Bayes classifier.

Naive Bayes classifier:

$$v_{NB} = \arg\max_{v \in V} P(v_j) \prod_i P(a_i \mid v_j)$$

Note: Verify the Illustrative example for Naive Bayesian classifier form Text Book only.

BAYESIAN BELIEF NETWORKS:

- A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities.
- In contrast to the naive Bayes classifier, which assumes that all the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables.
- Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether.
- Bayesian belief networks are an active focus of current research, and a variety of algorithms have been proposed for learning them and for using them for inference.
- The naive Bayes classifier makes significant use of the assumption that the values of the attributes a1 . . .a, are conditionally independent given the target value v. This assumption dramatically reduces the complexity of learning the target function.
- the naive Bayes classifier outputs the optimal Bayes classification. However, in many cases this conditional independence assumption is clearly 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. In contrast to the naive Bayes classifier, which assumes that all the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether. Bayesian belief networks are an active focus of current research, and a variety of algorithms have been proposed for learning them and for using them for inference.

6.11.1 Conditional Independence

Let us begin our discussion of Bayesian belief networks by defining precisely the notion of conditional independence. Let X, Y, and Z be three discrete-valued random variables. We say that X is *conditionally independent* of Y given Z if the probability distribution governing X is independent of the value of Y given a value for Z; that is, if

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

where $x_i \in V(X)$, $y_j \in V(Y)$, and $z_k \in V(Z)$. We commonly write the above expression in abbreviated form as $P(X|Y, Z) = P(X|Z)$. This definition of conditional independence can be extended to sets of variables as well. We say that the set of variables $X_1 \ldots X_l$ is conditionally independent of the set of variables $Y_1 \ldots Y_m$ given the set of variables $Z_1 \ldots Z_n$ if

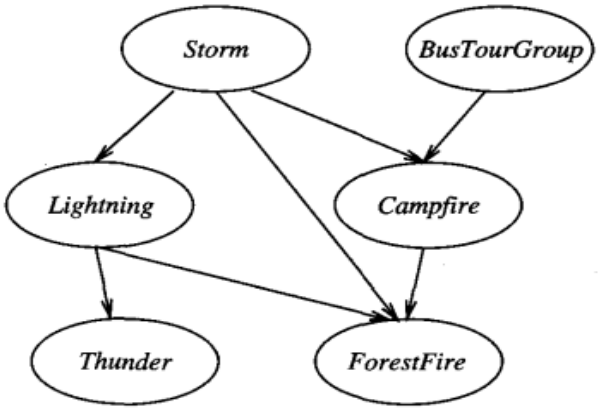
$$P(X_1 \ldots X_l | Y_1 \ldots Y_m, Z_1 \ldots Z_n) = P(X_1 \ldots X_l | Z_1 \ldots Z_n)$$

Note the correspondence between this definition and our use of conditional independence in the definition of the naive Bayes classifier. The naive Bayes classifier assumes that the instance attribute A_1 is conditionally independent of instance attribute A_2 given the target value V. This allows the naive Bayes classifier to calculate $P(A_1, A_2|V)$ in Equation (6.20) as follows

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

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

Equation (6.23) is just the general form of the product rule of probability from Table 6.1. Equation (6.24) follows because if A_1 is conditionally independent of A_2 given V, then by our definition of conditional independence $P(A_1|A_2, V) = P(A_1|V)$.



	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

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

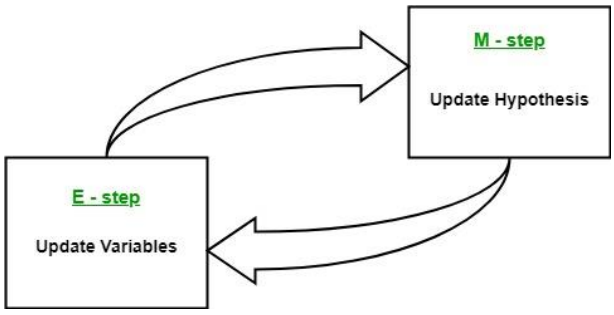
Expectation-Maximization Algorithm:

In the real-world applications of machine learning, it is very common that there are many relevant features available for learning but only a small subset of them are observable. So, for the variables which are sometimes observable and sometimes not, then we can use the instances when that variable is visible is observed for the purpose of learning and then predict its value in the instances when it is not observable.

On the other hand, ***Expectation-Maximization algorithm*** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning. It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

Algorithm:

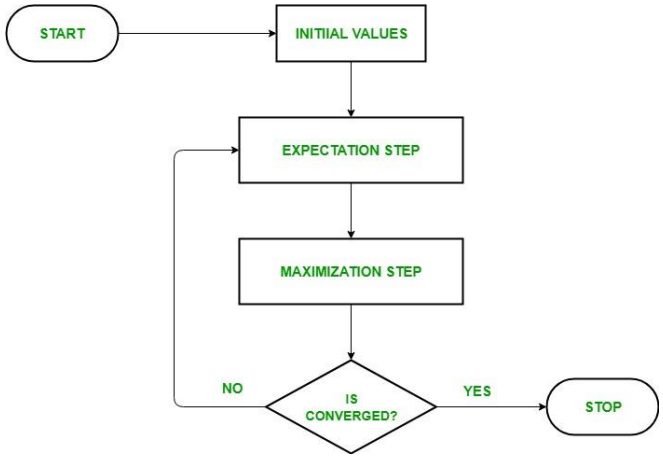
1. Given a set of incomplete data, consider a set of starting parameters.
2. **Expectation step (E – step):** Using the observed available data of the dataset, estimate (guess) the values of the missing data.



3. **Maximization step (M – step):** Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.

The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

- Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.



- The next step is known as "Expectation" – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.
- The next step is known as "Maximization"-step or *M-step*. In this step, we use the complete data generated in the preceding "Expectation" – step in order to update the values of the parameters. It is basically used to update the hypothesis.
- Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. "Expectation" – step and "Maximization" – step until the convergence occurs.

Usage of EM algorithm –

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm –

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

Disadvantages of EM algorithm –

- It has slow convergence.
- It makes convergence to the local optima only.
- It requires both the probabilities, forward and backward (numerical optimization requires only forward probability).

COMPUTATIONAL LEARNING THEORY

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. Within this setting, 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. As we shall see, 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 way training examples are presented to the learner
- For the most part, we will focus not on individual learning algorithms, but
- Rather on broad classes of learning algorithms characterized by the hypothesis
- Spaces they consider, the presentation of training examples, etc.

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

Sample complexity:

How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?

Computational complexity:

How much computational effort is needed for a learner to converge (with high probability) 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:

we consider a particular setting for the learning problem, called the probably approximately correct (PAC) learning model. We begin by 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***. For the sake of simplicity, we restrict the discussion to the case of learning Boolean valued concepts from noise-free training.

Problem Setting:

let X refer to the set of all possible instances over which target functions may be defined. For example, X might represent the set of all people, each described by the attributes. 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 $c : X \rightarrow \{0,1\}$.

We assume instances are generated at random from X according to some probability distribution D. For example, D might be the distribution of instances. In general, D may be any distribution, and it will not generally be known to the learner. All that we require of D is that it be stationary; that is, that the distribution not change over time. 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. To be fair, 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. W

Within this setting, we are interested in characterizing the performance of various learners L using various hypothesis spaces H, when learning individual target concepts drawn from various classes C. Because we demand that L be general enough to learn any

target concept from C regardless of the distribution of training examples, we will often be interested in worst-case analyses over all possible target concepts from C and all possible instance distributions D.

Error of a Hypothesis:

Error depends strongly on the unknown probability distribution. We are interested in how closely the learner's output hypothesis h approximates the actual target concept c, let us begin by defining the true error of a hypothesis h with respect to target concept c and instance distribution D. Informally, the true error of h is just the error rate we expect when applying h to future instances drawn according to the probability distribution D.

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

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

The concepts c and h are depicted by the sets of instances within X that they label as positive. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree (i.e., their set difference).

Definition: Training Error:

Training Error refers to the fraction of the training examples misclassified by h, in contrast to the true error defined above.

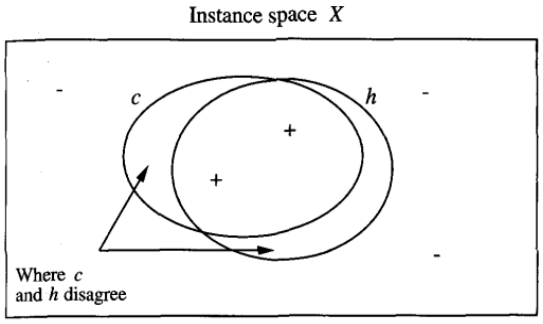
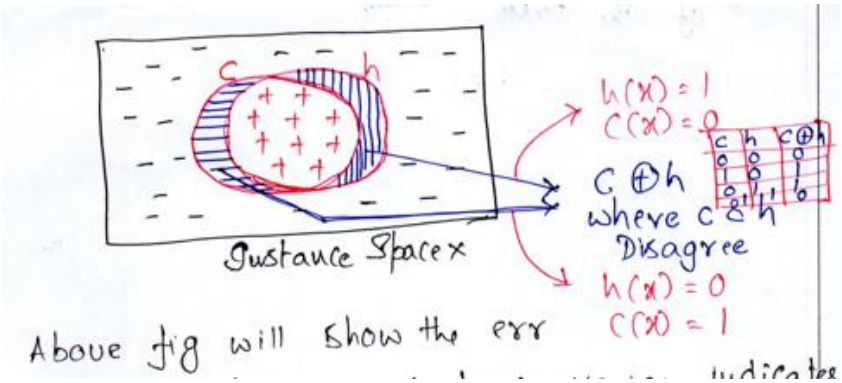


Figure shows this definition of error in graphical form. The error of hypothesis h with respect to target concept c. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree on its classification. The + and - points indicate positive and negative training examples.

Note h has a nonzero error with respect to c even though h and c agree on all five training examples observed thus far.



Note we have chosen to define error over the entire distribution of instances-not simply over the training examples-because this is the true error we expect to encounter when actually using the learned hypothesis h on subsequent instances drawn from D.

PAC Learnability:

Our aim 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 will require only that its probability of failure be bounded by some constant, δ, that can be made arbitrarily small. In short, we require only that the learner probably learn a hypothesis that is approximately correct-hence the term probably approximately correct learning, or PAC learning for short.

Sample complexity for finite hypothesis space:

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.

The reason is that in most practical settings the factor that most limits success of the learner is the limited availability of training data.

Let us now 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.

It is quite reasonable to ask that a learning algorithm be consistent, given that we typically prefer a hypothesis that fits the training data over one that does not.

Can we derive a bound on the number of training examples required by any consistent learner, independent of the specific algorithm it uses to derive a consistent hypothesis? The answer is yes.

To accomplish this, it is useful to recall the definition of version space

There we defined the version space, $VS_{H,D}$ to be the set of all hypotheses $h \in H$ that correctly classify the training

examples D.

Sample Complexity for finite Hypothesis Space:

Here we can understand that the growth in the no. of required training examples with problem size, called the Sample Complexity of learning problem.

Here we present a general bound on the Sample Complexity for a very broad class of learners. Called "Consistant Learners".

A learner is consistant if it outputs hyp that perfectly fit the training data, whenever possible.

Can we derive a bound on the no. of training examples required by any constant learner? yes, To accomplish this Let us recall Version Space over H and D,

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

From Haussler (1988), to bound the no. of examples needed, we need only bound the no. of examples needed to assure that the VS contains no unacceptable hyp. i.e.,

$$\forall h \in VS_{H,D}\ error_D(h) \leq \epsilon$$

then we say that $VS_{H,D}$ is ϵ-exhausted with c & D.

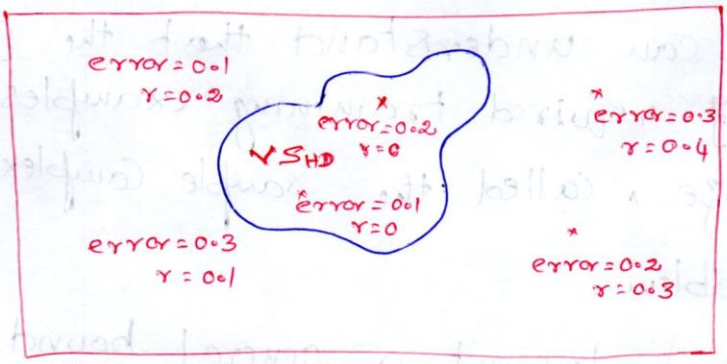


fig: Exhausting the version space. The space $VS_{H,D}$ is the subset of $h \in H$, which have zero training error (denoted by $r = 0$ in fig and the true $error_D(h)$ (denoted by error in the fig) may be non zero

ϵ-exhausting the version Space:

If the hypothesis 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 (wrt c) is less than or equal to $|H| e^{-\epsilon m}$.

i.e., let us use this result to find m, to reduce the probability of failure below some desired level δ.

$\delta \leq 1$

$\Rightarrow 1 - \delta \geq 0$

$0 \leq \delta \leq 1$

$\Rightarrow 0 \geq -\delta \geq -1$

$\Rightarrow 1 \geq 1 - \delta \geq 0$

$\Rightarrow 0 \leq 1 - \delta \leq 1$

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

Rearranging the term (Refer Unit-1)

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

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

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

Proof. Let $h_1, h_2, \ldots h_k$ be all the hypotheses in H that have true error greater than ϵ with respect to c. We fail to ϵ-exhaust the version space if and only if at least one of these k hypotheses happens to be consistent with all m independent random training examples. The probability that any single hypothesis having true error greater than ϵ would be consistent with one randomly drawn example is at most $(1-\epsilon)$. Therefore the probability that this hypothesis will be consistent with m independently drawn examples is at most $(1-\epsilon)^m$. Given that we have k hypotheses with error greater than ϵ, the probability that at least one of these will be consistent with all m training examples is at most

$$k(1-\epsilon)^m$$

And since $k \leq |H|$, this is at most $|H|(1-\epsilon)^m$. Finally, we use a general inequality stating that if $0 \leq \epsilon \leq 1$ then $(1-\epsilon) \leq e^{-\epsilon}$. Thus,

$$k(1-\epsilon)^m \leq |H|(1-\epsilon)^m \leq |H|e^{-\epsilon m}$$

which proves the theorem. □

We have just proved an upper bound on the probability that the version space is not ϵ-exhausted, based on the number of training examples m, the allowed error ϵ, and the size of H. Put another way, this bounds the probability that m training examples will fail to eliminate all "bad" hypotheses (i.e., hypotheses with true error greater than ϵ), for any consistent learner using hypothesis space H.

Let us use this result to determine the number of training examples required to reduce this probability of failure below some desired level δ.

$$|H|e^{-\epsilon m} \leq \delta \tag{7.1}$$

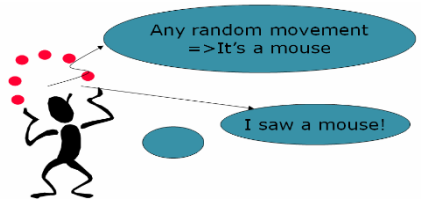
Rearranging terms to solve for m, we find

$$m \geq \frac{1}{\epsilon}(\ln|H| + \ln(1/\delta)) \tag{7.2}$$

To summarize, the inequality shown in Equation (7.2) provides a general bound on the number of training examples sufficient for *any consistent learner* to successfully learn any target concept in H, for any desired values of δ and ϵ. This number m of training examples is sufficient to assure that any consistent hypothesis will be probably (with probability $(1-\delta)$) approximately (within error ϵ) correct. Notice m grows linearly in $1/\epsilon$ and logarithmically in $1/\delta$. It also grows logarithmically in the size of the hypothesis space H.

INSTANCE BASED LEARNING:

Eager Learning:



- In artificial intelligence, eager learning is a learning method in which the system tries to construct a general, explicit description of input-independent target function during training of the system, where generalization beyond the training data is delayed until a query is made to the system.

- The main advantage gained in employing an eager learning method, such as an artificial neural network, is that the target function will be approximated globally during training, thus requiring much less space than using a lazy learning system.

- Eager learning systems also deal much better with noise in the training data.

- Eager learning is an example of offline learning, in which post-training queries to the system have no effect on the system itself, and thus the same query to the system will always produce the same result.

Lazy Learning: (Instance based Learning)

- Lazy learning is a learning method in which generalization of the training data is delayed until a query is made to the system, where the system tries to generalize the training data before receiving queries.

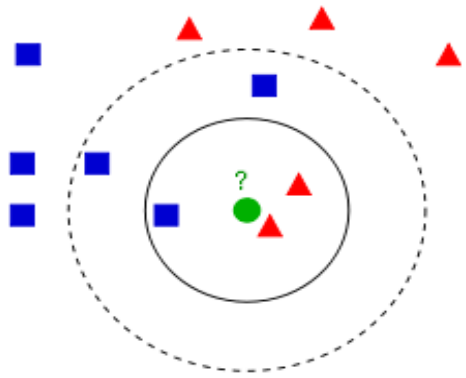
- Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.

- Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.

- Instance based learning includes nearest Neighbour 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 which are also referred as "lazy" learning methods because they delay processing until a new instance must be classified. The lazy learner can create many local approximations.

- 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. This is due to the fact that nearly all computation takes place at classification.



- A second disadvantage is too 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. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

Eager vs. Lazy Learning:

Eager Learning:

- Munges the training data as soon as it receives it.
- It's fast as it has pre-calculated algorithm
- Eager learning methods construct general, explicit description of the target function based on the provided training examples.
- Eager learning methods use the same approximation to the target function, which must be learned based on training examples and before input queries are observed.

Lazy Learning (Instance Learning):

- Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.
- It's slow as it calculates based on the current data set instead of coming up with an algorithm based on historic data
- Localized data so generalization takes time at every iteration
- Lazy learning methods can construct a different approximation to the target function for each encountered query instance.
- Lazy learning is very suitable for complex and incomplete problem domains, where a complex target function can be represented by a collection of less complex local approximations.

K - Nearest Neighbour Learning:

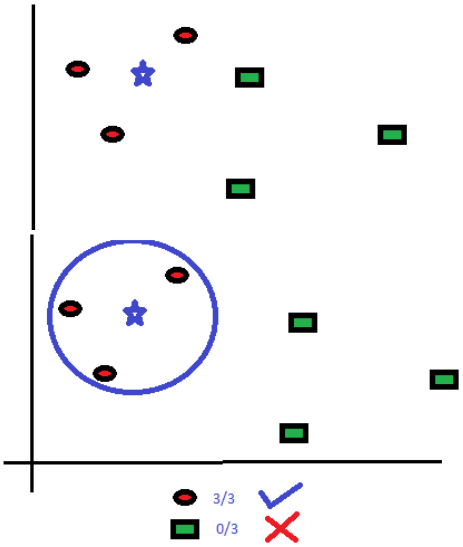
KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

How does the KNN algorithm work?

Let's take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):

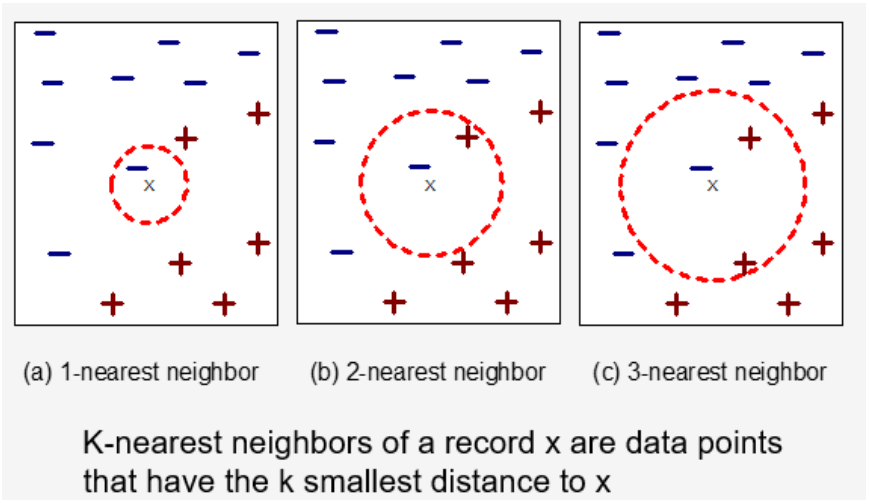


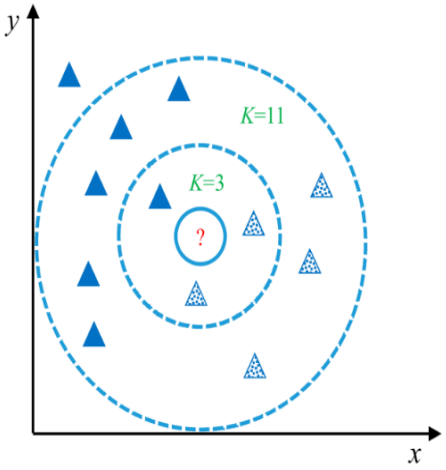
You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The "K" is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three data points on the plane. Refer to following diagram for more details:

How do we choose the factor K?

First let us try to understand what exactly does K influences in the algorithm. If we see the last example, given that all the 6-training observation remain constant, with a given K value we can make boundaries of each class. These boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.

Some other





Simulations/examples of KNN :

Pseudo Code of KNN

We can implement a KNN model by following the below steps:

1. Load the data.
2. Initialize the value of k.
3. For getting the predicted class, iterate from 1 to total number of training data points
    1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    2. Sort the calculated distances in ascending order based on distance values.
    3. Get top k rows from the sorted array.
    4. Get the most frequent class of these rows.
    5. Return the predicted class.

Description of the KNN:

The most basic instance-based method is the k-NEARESNT NEIGHBORING Algorithm. The main idea behind k-NN learning is so-called majority voting. This algorithm assumes all instances correspond to points in the n-dimensional space $\mathbb{R}^n$. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. More precisely, let an arbitrary instance x be described by the feature vector

$$< a_1(x), a_2(x), a_3(x), ...a_n(x) >$$

Where $a_r(x)$ denoted the value of the rth attribute of instance x, then the distance between two instances x_i and x_j defined to be $d(x_i, x_j)$ where

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

In nearest-neighbour learning the target function may be either discrete-valued or real-valued.

Let us first consider learning discrete-valued target functions of the form $f : \mathbb{R}^n \to V$,

where V is the finite set such that $V < -\{v_1, v_2, ..., v_m\}$.

The k-NEAREST NEIGHBOUR algorithm for approximation of a discrete-valued target function is given below.

As shown there, the value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_i. If we choose k = 1, then the 1-NEAREST NEIGHBOUR algorithm assigns to $f(x_i)$. The value $f(x_i)$ where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

Basic KNN algorithm:

For each training example $< x, f(x) >$, add the example to the list of *training_examples*.
Classification algorithm:

Given a query instance x_q to be classified,

Let $(x_1, x_2, x_3, ..., x_k)$ denote the k instances from *training_examples* that are nearest to x_q

Return $\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$

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

The k-NEAREST NEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions. To accomplish this, we have the algorithm calculate the mean value of the k nearest training examples rather than calculate their most common value. More precisely, to approximate a real-valued target function $f : \mathbb{R}^n \to \mathbb{R}$. We replace the final line of the above algorithm by the line

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

Distance-Weighted NEAREST NEIGHBOUR Algorithm:
One obvious refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbours according to their distance to the query point x_q, giving greater weight to closer neighbours. For example, in the above algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q.
This can be accomplished by replacing the final line of the algorithm by

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

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

To accommodate the case where the query point x, exactly matches one of the training instances xi and the denominator $d(x_q, x_i)^2$ is therefore zero, we assign $\hat{f}(x_q)$ to be $f(x_i)$ in this case.
We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

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

Once we add distance weighting, there is really no harm in allowing all training examples to have an influence on the classification of the x_q, because very distant examples will have very little effect on $\hat{f}(x_q)$. The only disadvantage of considering all examples is that our classifier will run more slowly. If all training examples are considered when classifying a new query instance, we call the algorithm a global method. If only the nearest training examples are considered, we call it a local method. When the rule in Equation (8.4) is applied as a global method, using all training examples, it is known as Shepard's method (Shepard 1968).

Minkowsky: $D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$

Euclidean: $D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$

Manhattan / city-block: $D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$

Camberra: $D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$

Chebychev: $D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$

Quadratic: $D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T Q(\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x} - \boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m}(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x_i})^2 \sum_{i=1}^{m}(y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square: $D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_x} - \frac{y_i}{size_y}\right)^2$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation: $D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\mathrm{sign}(x_i - x_j)\mathrm{sign}(y_i - y_j)$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure . Equations of selected distance functions. ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

VORONOI DIAGRAM:

- The diagram on the right side shows the shape of this decision surface induced by 1-NEAREST NEIGHBOR over the entire instance space. The decision surface is a combination of convex polyhedral surrounding each of the training examples using the concept of perpendicular bisectors.

- For every training example, the polyhedron 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.

Remarks on k-NEAREST NEIGHBOR Algorithm:

- The distance-weighted k-NEAREST NEIGHBOR Algorithm is a highly effective inductive inference method for many practical problems.
- It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data.
- Note that 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.

What is the inductive bias of k-NEAREST NEIGHBOR Algorithm?

- The basis for classifying new query points is easily understood based on the algorithm specified above.
- The inductive bias corresponds to an assumption that the classification of an instance x_q will be most similar to the classification of other instances that are nearby in Euclidean distance.
- One practical issue in applying k-NEAREST NEIGHBOR Algorithm is that the distance between instances is calculated based on all attributes of the instance (i.e., on all axes in the Euclidean space containing the instances).
- This lies in contrast to methods such as rule and decision tree learning systems that select only a subset of the instance attributes when forming the hypothesis.

Remarks on Lazy and Eager Learning:

- we considered three lazy learning methods: the k-NEAREST NEIGH BOR algorithm, locally weighted regression, and case-based reasoning. We call these methods lazy because they defer the decision of how to generalize beyond the training data until each new query instance is encountered. We also discussed one eager learning method: the method for learning radial basis function networks. We call this method eager because it generalizes beyond the training data before observing the new query, committing at training time to the network structure and weights that define its approximation to the target function. In this same sense, every other algorithm discussed elsewhere in this book (e.g., BACKPROPAGATION, C4.5) is an eager learning algorithm.
- There are obviously differences in computation time between eager and lazy methods. For example, lazy methods will generally require less computation during training, but more computation when they must predict the target value for a new query. The more fundamental question is whether there are essential differences in the inductive bias that can be achieved by lazy versus eager methods. The key difference between lazy and eager methods in this regard is
- Lazy methods may consider the query instance x, when deciding how to generalize beyond the training data D.
- Eager methods cannot. By the time they observe the query instance x, they have already chosen their (global) approximation to the target function.
- The lazy method effectively uses a richer hypothesis space because it uses many different local linear functions to form its implicit global approximation to the target function. Note this same situation holds for other learners and hypothesis spaces as well. A lazy version of BACKPROPAGA TION, for example, could learn a different neural network for each distinct query point, compared to the eager version of BACKPROPAGATION.
- The key point in the above paragraph is that a lazy learner has the option of (implicitly) representing the target function by a combination of many local approximations, whereas an eager learner must commit at training time to a single global approximation. The distinction between eager and lazy learning is thus related to the distinction between global and local approximations to the target function

CASE-BASED REASONING (CBR):

Case-based reasoning (CBR) is a problem-solving paradigm that is different from other major A.I. approaches. A CBR system can be used in risk monitoring, financial markets, defense, and marketing just to name a few. CBR learns from past experiences to solve new problems. Rather than relying on a domain expert to write the rules or make associations along generalized relationships between problem descriptors and conclusions, a CBR system learns from previous experience in the same way a physician learns from his patients.

- Case-Based reasoning (CBR), broadly construed, is the process of solving new problems based on the solutions of similar past problems.
- It is an approach to model the way humans think to build intelligent systems.
- Case-based reasoning is a prominent kind of analogy making.
- CBR: Uses a database of problem solutions to solve new problems.
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling.
- Case-Based Reasoning is a well-established research field that involves the investigation of theoretical foundations, system development and practical application building of experience-based problem solving with base line of by remembering the past experience.
- It can be classified as a sub-discipline of Artificial Intelligence
  - ✓ learning process is based on analogy but not on deduction or induction
  - ✓ best classified as supervised learning *(recall the distinction between supervised, unsupervised and reinforcement learning methods typically made in Machine Learning)*
  - ✓ Learning happens in a memory-based manner.
- Case – previously made and stored experience item
- Case-Base – core of every case – based problem solver - collection of cases

Everyday examples of CBR:

- An auto mechanic who fixes an engine by recalling another car that exhibited similar symptoms
- A lawyer who advocates a particular outcome in a trial based on legal precedents or a judge who creates case law.
- An engineer copying working elements of nature (practicing biomimicry), is treating nature as a database of solutions to problems.

Few commercially/industrially really successful AI methods:

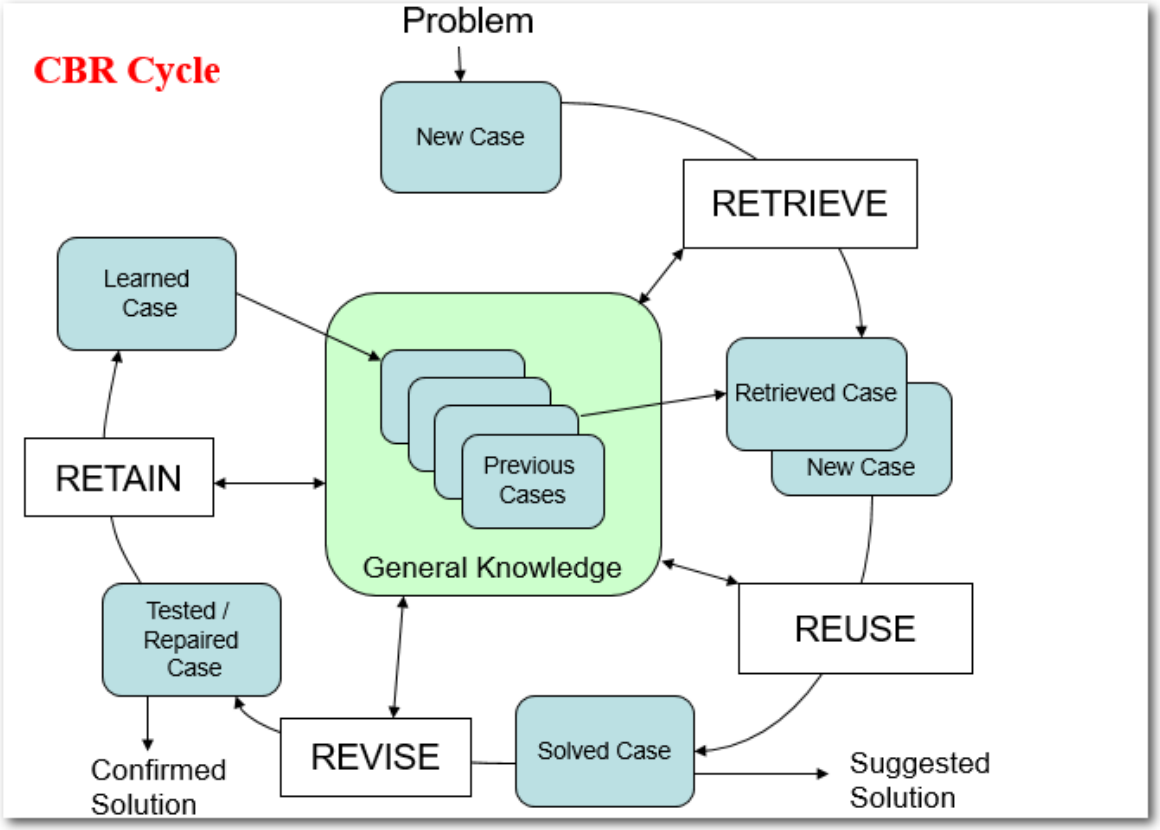
- Customer support, help-desk systems: diagnosis and therapy of customer's problems, medical diagnosis
- Product recommendation and configuration: e-commerce
- Textual CBR: text classification, judicial applications (in particular in the countries where common law (not civil law) is applied)[like USA, UK, India, Australia, many others]
- Applicability also in ill-structured and bad understood application domains.

There are three main types of CBR that difer significantly from one another concerning case representation and reasoning:

1. Structural *(a common structured vocabulary, i.e. an ontology)*
2. Textual *(cases are represented as free text, i.e. strings)*
3. Conversational
    - *A case is represented through a list of questions that varies from one case to another ; knowledge is contained in customer / agent conversations.*

Architecture of CBR/ CBR Cycle:

CBR Cycle:



Despite the many different appearances of CBR systems, the essentials of CBR are captured in a surprisingly simple and uniform process model.

- The CBR cycle is proposed by Aamodt and Plaza.
- The CBR cycle consists of 4 sequential steps around *the knowledge of the CBR system*.
    - RETRIEVE
    - REUSE
    - REVIZE
    - RETAIN

RETRIEVE:

- One or several cases from the case base are selected, based on the modeled similarity.
- The retrieval task is defined as finding a small number of cases from the case-base with the highest similarity to the query.
- This is a k-nearest-neighbor retrieval task considering a specific similarity function.
- When the case base grows, the efficiency of retrieval decreases => methods that improve retrieval efficiency,
  e.g. specific index structures such as kd-trees, case-retrieval nets, or discrimination networks.

REUSE:

- Reusing a retrieved solution can be quite simple if the solution is returned unchanged as the proposed solution for the new problem.
- Adaptation (if required, e.g. for synthetic tasks).
- Several techniques for adaptation in CBR
  - Transformational adaptation
  - Generative adaptation
- Most practical CBR applications today try to avoid extensive adaptation for pragmatic reasons.

REVISE:

- In this phase, feedback related to the solution constructed so far is obtained.
- This feedback can be given in the form of a correctness rating of the result or in the form of a manually corrected revised case.

- The revised case or any other form of feedback enters the CBR system for its use in the subsequent retain phase.
- Retain
- The retain phase is the learning phase of a CBR system (adding a revised case to the case base).
- Explicit competence models have been developed that enable the selective retention of cases (because of the continuous increase of the case-base).
- The revised case or any other form of feedback enters the CBR system for its use in the subsequent retain phase.

RETAIN:

- The retain phase is the learning phase of a CBR system (adding a revised case to the case base).
- Explicit competence models have been developed that enable the selective retention of cases (because of the continuous increase of the case-base).
- The revised case or any other form of feedback enters the CBR system for its use in the subsequent retain phase.