Machine Learning

Module-5

SURYA Gold
Date _____ Page _____

Chapter-5

Evaluating hypothesis.

Finding accuracy of hypothesis is fundamental to Machine learning

5.1 Motivation

To evaluate the performance of learned hypothesis is possible. by simply understand use of hypothes.

For instances learning from limited-size database indicating effectiveness of different medical treatments.

→ It is important to understand accuracy of learned hypothes.

→ second reason is evaluating hypothes is an integral component of many learning methods.

Estimating the accuracy of a hypothesis is relatively straightforward when data is more, but challenge in learning the accuracy in limited set of data.

2 difficulties arise:

1. Bias in the estimate:

first observed accuracy of the learned hypothesis over the training examples is often poor in accuracy over future examples Because they provide biased estimation of accuracy over future examples.

To obtain unbiased estimation

of future accuracy typically test the hypotheses on set of examples chosen independently of the training examples and hypothesis.

2. Variance in the estimate:

Second, even if the hypothesis accuracy is measured over unbiased set of test examples independent of training examples the measured accuracy still vary from true accuracy.

5.2 Estimating hypothesis Accuracy

On Evaluating the learned hypothesis we are most often interested in estimating accuracy which classify future instances. Also we like to know error in accuracy estimated

Consider following learning problem.

Consider :- Target function = People plan to purchase new skis (Skis is a strip of rigid material worn underfoot to glide snow)

Given = sample of training data collected by surveying people arrive at ski resort

Instance space X = Space of all people. describe attribute age occupation etc.

SURYA Gold
Date _____ Page _____

Distribution D : specifies each person x encountered as the next person arriving at ski resort.

Target function $f: X \rightarrow \{0, 1\}$ classifies each person according to plan to buy or not.

Two questions are set of interest.

1). Given hypothesis h and data samples n drawn at some distribution D what is the accuracy of h for future instances.

2). What is the probable error in accuracy estimated?

5.2.1 Sample Error and True error.

To answer above question. Lets distinguist 2 notions.

One is error rate of hypothesis over sample data that is available.
Second is error rate of hypothesis over unknown distribution D

Sample error :

Definition: The sample error (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where

n = no of examples in S

quantity $\delta(f(x), h(x)) = 1$ if $f(x) \neq h(x)$ and 0 otherwise.

Sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.

True error: True error (denoted $error_D(h)$) of hypothesis h with respect to target function f and distribution D, is the probability that h will misclassify an instance drawn at random according to D.

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

True error of hypothesis will misclassify a single randomly drawn instances from the distribution D.

5.2.2 Confidence intervals for Discrete-valued hypothesis

Detecting how good we estimate $error_D(h)$ is provided by $error_S(h)$?

For the case in which h is a discrete valued hypothesis

To find true error for discrete valued hypothesis h is based on sample error over samples.

Where

→ Sample S contains n eg drawn independently of one another and independent of h. according to probability distribution D

→ $n \geq 30$

→ hypothesis h commits r errors over these n examples.

Under 3 conditions we make following assertions.

1. Given no other information the most probable value of $error_D(h)$ is $error_S(h)$

2. With approximately 95% probability the true $error_D(h)$ lies in the interval

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)\,(1 - error_S(h))}{n}}$$

For example

Consider S contains $n = 40$ eg & h commits $r = 12$ errors over data.

In this case sample error

$$error_S(h) = 12/40 = .30$$

We must expect probable estimate of true

Consider S' contain 40 new eg. error$_S$(h) vary slightly with error$_S$(h)

There is a random diff between S & S' we repeat over & over 40 new eg. we find approximately 95% of eg. calculating intervals we get true error.

∴ we call 95% confidence interval estimate for error$_D$(h)

From above value r=12, n=40 95% confidence interval.

We can write.
$0.30 \pm (1.96 \cdot .07) = 0.30 \pm .14$.

Value of Z_N for two sided N% confidence level.

Confidence level N%	50%	68%	80%	90%	95%	98%
Constant Z_N	0.67	1.00	1.28	1.64	1.96	2.33

The above expression for 95% confidence generalized to a constant 1.96.

A different constant Z_N is used to calculate N% confidence interval.

General expression for approximate N% Confidence intervals for error$_D$(h) is

$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

Constant Z_N is chosen depending on confidence level refer above table.

From above equation error bars or confidence levels are estimated for $error_T(h)$ based on $error_S(h)$

5.3 Basics of Sampling Theory

- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- The *expected value*, or *mean*, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent E[Y].
- The *variance* of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The *standard deviation* of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used used to represent the standard deviation of Y.
- The *Binomial distribution* gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p.
- The *Normal distribution* is a bell-shaped probability distribution that covers many natural phenomena.
- The *Central Limit Theorem* is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An *estimator* is a random variable Y used to estimate some parameter p of an underlying population.
- The *estimation bias* of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A *N% confidence interval* estimate for parameter p is an interval that includes p with probability N%.

TABLE 5.2
Basic definitions and facts from statistics.

PTO

This section introduces notions from statistics and sampling theory including probability distributions, expected value, variance, Binomial & normal distribution etc.

5.3.1 Error Estimation and Estimating Binomial proportions.

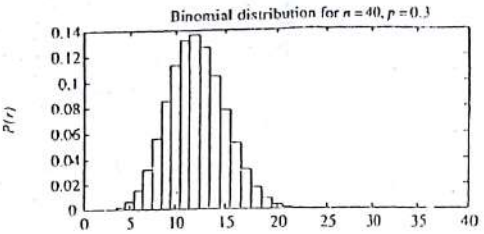
As we know Sample error & true error depends on size of data sample.

Consider a problem in :-
Estimating proportion of population by observed proportion with sample. Wher h misclassifies the eg.

The key to answer the Pblm is we measure the sample error with random outcome.
We collect sample - S
For a distribution-D then measure
Sample error - $error_S(h)$

We repeat many times on different values like $error_{S_i}(h_i)$ are the outcome of i^{th} experiment of random variable.

Imagine we run k such random experiments. Measuring random variables $error_{S_1}(h), error_{S_2}(h), \ldots error_{S_i}(h)$.



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

TABLE 5.3
The Binomial distribution.

Imagine further that we had plotted a histogram displaying the frequency with which we observed each possible error value.

As we allowed k to grow the histogram would approach the form of distribution in above table.

This table describe probability distribution called the Binomial distribution.

5.3.2 The Binomial Distribution

Consider a problem:
Estimating probability of heads P.
Coin n times tossed
Records no of times r that turns heads
Reasonable estimate of P is r/n.

Note no of heads r to vary in 1st experiments yielding different value of p.

Binomial distribution describes for each possible value of r given sample n.

Estimating P coin tosses = estimating $error_D(h)$ with h random samples of instances
Probability of single instance drawn at random will be misclassified.

Thus r/n correspond to $error_S(h)$.

! estimating P for coins to estimating $error_D(h)$

Binomial distribution depends on specific sample sizes or $error_D(h)$.

The general setting to binomial distribution is

1. There is a base (eg toss of the coin) outcomes a random variable say Y.
Y take 2 possible values (eg: Y=1 if heads, Y=0 if tails)

2. The probability that $Y=1$ on single trial is given by constant P, independed of outcome of any other experiment of $Y=0$

3. A series of n independent trials of underlying experiment is performed producing sequence of independent variables $Y_1, Y_2, \ldots Y_n$ let R denote no of trials $Y_i = 1$ in series of n experiments

$$R = \sum_{i=1}^{n} Y_i$$

4. Probability of random variables R on specific value r in Binomial distribution

$$P_r(R=r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

5.3.3 Mean and Variance

Two properties of random variables value are mean and Variance.

Defn: Consider a random variable Y that takes on the possible values $Y_1 \ldots Y_n$.
The Expected value of Y, $E[Y]$ is

$$E[Y] = \sum_{i=1}^{n} y_i P_r(Y = y_i)$$

When Binomial distribution is shown that

$$E[Y] = np.$$

Where n and P are the parameters of Binomial distribution.

Variance

Definition: The variance of a random variable Y, Var[Y] is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

Square root of the variance is called Stand deviation of Y denoted σ_Y.

Standard deviation:

The standard deviation of a random variable Y, σ_Y is

$$\sigma_Y = \sqrt{E[(Y - E(Y))^2]}$$

random variable Y is governed by Binomial distribution. Then the variance and standard deviation are given by

$$Var[Y] = np(1-p)$$
$$\sigma_Y = \sqrt{np(1-p)}$$

5.3.4 Estimators, Bias and Variance

Consider a random variable $error_S(h)$ obeys a Binomial distribution

Consider Binomial distribution we have

$$error_S(h) = \frac{r}{n}$$

$$error_D(h) = p$$

Where n = no of instances in sample S
r = no of instances from S misclassified by h
p = Probability of misclassifying single instances drawn from D.

Definition: The estimation bias of an estimator Y for an arbitrary parameter p is

$$E[Y] - p$$

If estimation bias is zero Y is unbiased estimate for p.

Two remarks regarding estimation bias.
1) $error_S(h)$ give an unbiased estimate of $error_D(h)$ when h and S must be choosen independently.
2) estimation bias should not be confused with inductive bias.
Where estimation bias = numerical quantity
Inductive bias = Set of assertions.

Consider example. We test hypothesis
commits $r = 12$ error
sample of $n = 40$
Then unbiased estimate for $error_D(h)$ is given by $error_S(h) = r/n = 0.3$

Binomial distributed it variance is given by

$$\sigma_Y = \sqrt{np(1-p)} \quad \text{(from prove topic 5.3.3)}$$

An general given is errors in sample h

independently drawn test examples, the standard deviation for $error_S(h)$ is given by

$$\sigma_{error_S(h)} = \frac{\sigma}{n} = \sqrt{\frac{p(1-p)}{n}}$$

Substitute $r/n = error_S(h)$ for p

$$\sigma_{error_S(h)} = \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

5.3.5 Confidence Intervals

Estimate the uncertainty given an interval within which true value is expected to fall. Along with probability with which it is expected to fall in the interval.

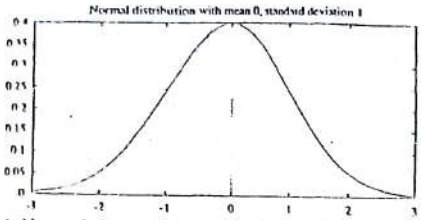
Such estimates are called confidence interval estimates.

Definition: An N% confidence interval for some parameter p is an interval that is expected with probability N% to contain p.

For example $r = 12$ errors

$n = 40$ we can say approximately 95% probability that interval 0.30 ± 0.14 contains true error $error_D(h)$.

The Normal distribution summarized in table below.



A Normal distribution (also called a Gaussian distribution) is a bell-shaped distribution defined by the probability density function

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

A Normal distribution is fully determined by two parameters in the above formula: μ and σ.

If the random variable X follows a normal distribution, then:

- The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- The expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- The variance of X, $Var(X)$, is

$$Var(X) = \sigma^2$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$

The Central Limit Theorem (Section 5.4.1) states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately Normal.

TABLE 5.4
The Normal or Gaussian distribution.

It is bell-shaped distribution fully specified mean M and standard deviation σ.

for large n any binomial distribution is very closely approximated by normal distribution.

We prefer to work with normal distribution specifying size of the interval.

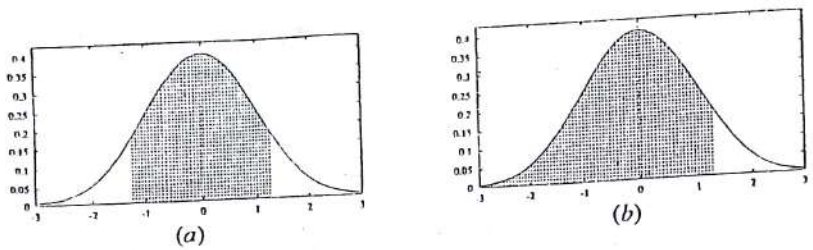
From figure 5.1 illustrates such an interval To summarize if a random variable y obeys a normal distribution with mean M and standard deviation σ

than measured by random value of y of Y with interval N%

$$M \pm ZN\sigma$$

Mean M will fall in to following interval N%

$$y \pm ZN\sigma$$



(a) (b)

FIGURE 5.1
A Normal distribution with mean 0, standard deviation 1. (a) With 80% confidence, the value of the random variable will lie in the two-sided interval [−1.28, 1.28]. Note $z_{.80} = 1.28$. With 10% confidence it will lie to the right of this interval, and with 10% confidence it will lie to the left. (b) With 90% confidence, it will lie in the one-sided interval [−∞, 1.28].

for discrete valued hypotheses.

$$error_S(h) \pm ZN \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

5.3.6 Two sided and one-sided Bounds

Two sided bound it bounds estimated quantity from above and from below In some cases we are interested only in one-sided bound

For example :- what is the probability that $error_D(h)$ is at most 6?" this kind of

One sided question is natural bounding the maximum error of h and do not mind if the true error is much smaller than estimated.

Two sided confidence can be converted to one sided interval with twice the confidence.

5.4 A General approach for deriving Confidence Intervals

Estimating error(h) for a discrete-valued hypothesis h, based on n independently drawn instances

Problem in estimating the mean of a population based on the mean of a randomly drawn sample of size n.

General process includes following steps:

1) Identify the underlying population parameter p to be estimated for eg $error_D(h)$
2) Define the estimator Y (eg. $error_S(h)$) that describe to choose a minimum variance
3) Determine the probability distribution by that governs the estimator Y including mean and variance
4) Determine the N% confidence interval by finding thresholds L and U such that N% of mass probability distribution lies between L and U

SURYA Gold
Date Page

5.4.1 Central Limit Theorem.

Attempts to derive confidence intervals is Central Limit Theorem.

Consider general setting of n independently drawn random variables $Y_1 \ldots Y_n$ with unknown probability distribution (eg: n tosses of the same coin).

μ = denotes mean of each Y_i
σ = standard deviation
Y_i = independent identically distributed random variables.

Mean μ of distribution governing Y_i
then mean $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$

Central Limit Theorem states probability distribution governing Y_n

Theorem
Central Limit Theorem
Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by arbitrary probability distribution with mean μ and finite variance σ^2.
Define the sample mean, $\bar{Y}_n = 1/n \sum_{i=1}^{n} Y_i$.

Then as $n \to \infty$, the distribution governing

$\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$ approaches a normal distribution with zero mean and standard deviation equal to 1.

Central limit theorem describes how mean and variance of $\bar{Y}$ can be used to determine the mean and variance of individual Y_i.

Central limit theorem very useful by defining estimator that is mean of some sample where estimator can be approximated by a normal distribution for sufficiently larger.

5.5 Difference in error of two hypothesis

Consider 2 hypotheses h_1 & h_2 for some discrete valued target function

→ hypothesis h_1 has been tested on samples, S_1 containing n_1 random eg.

→ h_2 has been tested independent S_2 containing h_2

Suppose we find the difference d b/w true errors of 2 hypotheses.

$$d \equiv error_D(h_1) - error_D(h_2)$$

We will use the generic 4-step procedure described in 5.4

→ Since $\hat{d}$ is to be estimated next define estimator. only choice is to find difference between the sample errors

denoted by $\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$

→ here we follow distribution approximately normal with mean d.

Using (5.3.4) we can obtain approximate variance of each of these distribution we have

$$\sigma^2_{\hat{d}} \approx \frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}$$

→ using approximate variance $\sigma^2_{\hat{d}}$ given above approximate N% confidence interval estimate for d is

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}}$$

Above all analysis consider h_1 and h_2 are tested on single samples S we redefine $\hat{d}$ as

$$\hat{d} \equiv error_S(h_1) - error_S(h_2)$$

5.5.1 Hypothesis Testing

Suppose for example
What is the probability that
$error_D(h_1) > error_D(h_2)$?"
Suppose we measure sample errors for
h_1 and h_2 using two independent samples
S_1 and S_2 size 100
$error_{S_1}(h_1) = .30$ and $error_{S_2}(h_2) = .20$
observed difference is $\hat{d} = .10$

Then difference in sample errors even
when $error_D(h_1) \leq error_D(h_2)$.
Then what is the probability of
$error_D(h_1) > error_D(h_2)$ from above difference
is $\hat{d} = .10$. In this case what is the
probability that $d > 0$ given $\hat{d} = .10$?

-> Put another way $\hat{d} < d + .10$ falls in to
one sided interval.
Then mean for one sided interval $\hat{d} < M_{\hat{d}} + .10$

Summarize Probability $Pr(d > 0)$ then $\hat{d}$
falls in to one sided $\hat{d} < M_{\hat{d}} + .10$

Let us begin re-expressing the interval
$\hat{d} < M_{\hat{d}} + .10$
Then standard deviations from the mean
we find $\sigma_{\hat{d}} \approx .060$
We are express the interval as approximately

$$\hat{d} < M_{\hat{d}} + 1.64\sigma_{\hat{d}}$$

Therefore given observed $\hat{d} = .10$ the probability that $error_D(h_1) > error_D(h_2)$ is approximately .95.
Also we accept hypothesis that $error_D(h_1) > error_D(h_2)$.

5.6 Comparing learning algorithms

We always compare performance of two learning algorithms L_A and L_B rather than two specific hypothesis.

Suppose we wish to determine best method to estimate performance first we determine learning method on target function f.

In other words we estimate the expected value of difference in their errors

$$E_{S \subset D}[error_D(L_A(S)) - error_D(L_B(S))]$$

Where $L(S)$ = denotes hypothesis o/p of learning method L
S - sample of training data
$S \subset D$ - indicates that expected value is taken over samples S
The above expression describes the expected value of difference in error b/w learning method L_A and L_B

The training data can be used to train both L_A and L_B and test data can be used to compare accuracy of 2 learned hypotheses

$$\text{error}_{T_0}(L_A(S_0)) - \text{error}_{T_0}(L_B(S_0))$$

2 key difference b/w this estimator & the quantity in above eqn

T_0 = disjoint test set T_0 where quality is to divide D_0 in to a training set S_0.

One way to improve above equation is repeatedly Partion data D_0 of training & test sets this leads to the table 5.1 -

1. Partition the available data D_0 into k disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do
   use T_i for the test set, and the remaining data for training set S_i
   - $S_i \leftarrow (D_0 - T_i)$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=1}^{k}\delta_i \qquad \text{(T5.1)}$$

TABLE 5.5
A procedure to estimate the difference in error between two learning methods L_A and L_B. Approximate confidence intervals for this estimate are given in the text.

Estimating difference in errors, tests learning algorithm K times

The difference in errors δ is returned to estimate the difference b/w 2 learning algorithm.

The quantity $\bar{\delta}$ returned by the procedure from table 5.5 above.

We can view $\bar{\delta}$ an estimate the quantity,

$$E_{S \subset D_0}[error_D(L_A(S)) - error_D(L_B(S))]$$

Where S represent random sample of size $\frac{k-1}{k}|D_0|$ drawn uniform from D_0.

The approximate N% confidence interval for estimating the quantity using $\bar{\delta}$ is given by

$$\bar{\delta} \pm t_{N,k-1} s_{\bar{\delta}}$$

Where $t_{N,k-1}$ is a constant plays a role analogous to Z_N

Where $s_{\bar{\delta}}$ is the estimate of the standard deviation of distribution governing $\bar{\delta}$.

$s_{\bar{\delta}}$ defined as

$$s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$$

Constant $t_{N,k-1}$ has 2 subscripts.
1st describes confidence level & 2nd constant called degree of freedom.

5.6.1 Paired t Tests

Let us comparing 2 learning methods given a fixed set of data.

The best way to understand the Justification For the Confidence interval From this above equation

$$\bar{\delta} \pm t_{N,k-1} S_{\bar{\delta}}$$

Consider the following estimation problem.

-> We are given the observed value of a set of independent, identically distributed random variables $Y_1, Y_2, \ldots Y_k$.

-> We wish to estimate the mean M of the probability distribution governing these Y_i.

-> The estimator we will use is the sample mean $\bar{Y}$

$$\bar{Y} \equiv \frac{1}{k} \sum_{i=1}^{k} Y_i$$

This problem of estimating the distribution mean M based on the sample mean $\bar{Y}$ is general.

Consider table 5.5 in previous topic for Comparing learning methods. We can modify the procedure.

1st note the size of the test sets T_i. Choosen at least 30 examples. We require standard deviation of this distribution

The t test applies to precisely in which task estimate sample mean of a collection of independent, identically and normally distributed random variables. We can restate the equation.

$$M = \bar{Y} \pm t_{N.k-1} S_{\bar{Y}}$$

where $S_{\bar{Y}}$ is the estimated standard deviation of the sample mean.

$$S_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (y_i - \bar{y})^2}$$

Where $t_{N.k-1}$ is a constant.

5.6.2 Practical considerations

Above discussion justifies the use of the confidence interval

Sample mean $\bar{Y}$ to estimate the mean of a sample containing k independent, identically and normal distributed random variable. This fits the idealized methods.

Table 5.5 doesn't strictly apply problem is only way to generate new S_i is to resample D_0 dividing it into training test sets in different way.

S_i not independent on one another are based on overlapping set of training examples

SURYA Gold

Date _____ Page _____

To summarize no single procedure for comparing learning methods based on limited data satisfies all the constraints. It is wise to keep in mind statistical models rarely fit perfectly in testing learning algorithm when available data is limited.

ML

CHAPTER - 8

MODULE 5

Instance Based Learning

Instance-based learning methods simply store the training examples,in contrast to learning methods that construct a general, explicit description of the target function when training examples are provided. Generalizing beyond these examples is postponed until a new instance must be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Instance-based methods are sometimes referred to as "lazy" learning methods because they delay processing until a new instance must be classified. 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.

Types of instance based learning methods

1. Nearest Neighbor
2. Locally weighted Regression
3. Case based Reasoning

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. Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance. One key difference between these approaches and the other methods is that instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified. This is called local approximation to the target function. This applies in the neighbourhood of the new query instance, and never constructs an approximation designed to perform well over the entire instance space.

Advantage:

When the target function is very complex, it can be described by a collection of less complex local approximations.

Disadvantages:

Cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.

They typically consider all attributes of the instances when attempting to retrieve similar training examples from memory.

K-NEAREST NEIGHBOR LEARNING

The most basic instance-based method is the K-NEAREST NEIGHBOR algorithm. This algorithm assumes all instances correspond to points in the n-dimensional space R". The nearest neighbours 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 $<a_1(x),a_2(x)\ldots\ldots\ldots\ldots a_n(x)>$

where $a_r(x)$ denotes the value of the rth attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

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

In nearest-neighbor learning the target function may be either discrete-valued or real-valued. Discrete-valued target functions are to be learnt of the form, $f : \Re^n \rightarrow V$ where V is the finite set $\{v_l, \ldots v_s\}$.

KNN for learning discrete valued function

Training algorithm :

- For each training example<x, f(x)>, add the example to the list training examples

Classification algorithm :

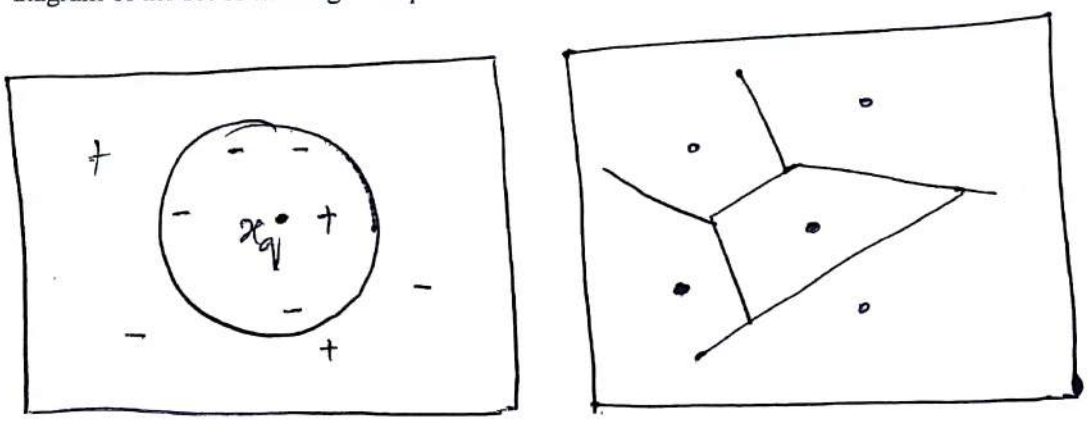
- Given a query instance x_q to be classified'
  - Let $x_1 \ldots\ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return
    $\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$

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

NEAREST NEIGHBOR algorithm assigns to $f(x_q)$ the value $f(x_i)$ where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

Figure illustrates the operation of the k-NEARESNTEIGHBOR algorithm for the case where the instances are points in a two-dimensional space and where the target function is boolean valued. The positive and negative training examples are shown by "+" and "-" respectively. A query point x_q is shown as well. Note the 1-NEARESTNEIGHBOR algorithm classifies x_q, as a positive example in this figure, whereasthe 5- NEARESTNEIGHBOR algorithm classifies it as a negative example. The decision surface shown on the right side of the figure is a combination of convex polyhedra surrounding each of the training examples. 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.



A set of positive and negative training examples is shown on the left, along with a query instance x_q, to be classified. The I-NEAREST NEIGHBOR algorithm classifies x, positive, whereas 5-NEARESTNEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEARESTNEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point .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.

The k- NEARESTNEIGHBOR algorithm never forms an explicit general hypothesis $\hat{f}$ regarding the target function f. It simply computes the classification of each new query instance as needed.

KNN for Continuous valued function

The k-NEARESTNEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions. The algorithm calculates 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 we replace the final line of the above algorithm by the line

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

Distance-Weighted NEAREST NEIGHBOR Algorithm

A refinement to the k-NEARESNTEIGHBOR 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, in the algorithm, which approximates discrete-valued target functions, each neighbour is weighted according to the inverse square of its distance from x_q

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

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

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

The above is the approximation for the weighted case of KNN for discrete valued target functions. The same can be extended to real valued functions by replacing the final line of the algorithm by

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

Advantages of KNN

The distance-weighted k-NEARESNTEIGHBOR 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. By taking the weighted average of the k neighbours nearest to the query point, it can smooth out the impact of isolated noisy training examples.

Issues with KNN

1. Curse of dimensionality. 2. Indexing

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

Consider applying k-NEARESNTEIGHBOR to a problem in which each instance is described by 20 attributes, but where only 2 of these attributes are relevant to determining the classification for the particular target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20-dimensional instance space. As a result, the similarity metric used by k-NEARESNT EIGHBOR--depending on all 20 attributeswill be misleading. The distance between neighbours will be dominated by the large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the curse of dimensionality. Nearest-neighbour approaches are especially sensitive to this problem.

Overcoming Curse of Dimensionality

1. This problemcan be overcome by weighting each attribute differently when calculating the distance between two instances. This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes. The amount by which each axis should be stretched can be determined automatically using a cross-validation approach. This process of stretching the axes in order to optimize the performance of k-NEAREST NEIGHBOR provides a mechanism for suppressing the impact of irrelevant attributes.

2. Completely eliminate the least relevant attributes from the instance space.

3. Stretch axes by a constant value that varies over the instance space.

Indexing

One additional practical issue in applying k-NEARESNTE IGHBOR is efficient memory indexing. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query.

Locally Weighted Regression

Locally weighted regression is a generalization of KNN.It constructs an explicit approximation to f over a local region surrounding x_q. Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f. For example, we might approximate the target function in the neighbourhood surrounding x, using a linear function, a quadratic function, a multilayer neural network, or some other functional form. 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, and regression because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighbourhood surrounding x_q. This approximation is then used to calculate the value f'(x,), 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.

Consider the case of locally weighted regression in which the target function f is approximated near x, using a linear function of the form

$$\hat{f}(x) \leftarrow w_0 + w_1 a_1(x) + \ldots\ldots + w_n a_n(x)$$

$a_i(x)$ denotes the value of the ith attribute of the instance x.

The error criterion that defines the difference between the target and the local approximation in LWR can be represented by three different criteria:

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

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \epsilon k nearestnbrsof x_q} (f(x) - \hat{f}(x))^2$$

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 :

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

3. Combine 1 and 2 :

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

Criterion two allows every training example to have an impact on the classification of x_q. However, this approach requires computation that grows linearly with the number of training examples. Criterion three is a good approximation to criterion two and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number k of neighbours considered.

The criterion 3 above can be used to derive the gradient descent rule for LWR for minimizing the error function and it is represented as follows:

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

Terminology in LWR

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

Radial Basis Functions

Radial Basis functions are closely related to distance-weighted regression and to ANNs. The learned hypotheses have the form

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

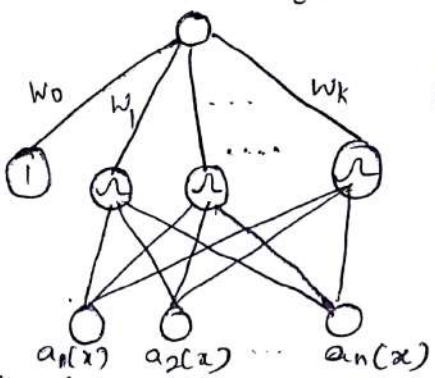
where

- each x_u is an instance from X and
- $K_u(d(x_u, x))$decreases as $d(x_u, x)$increases and
- K is a user-provided constant

Though $\hat{f}(x)$ is a global approximation to f(x), the contribution of each of the Ku terms is localized to a region nearby the point x_u

It is common to choose each function $K_u(d(x_u, x))$to be a Gaussian function centered at x_u with some variance σ^2

The function given by Equation (1) can be viewed as describing a two- layer network where the first layer of units computes the values of the various K_u,(d(x_u, x)) and where the second layer computes a linear combination of these first-layer unit values. An example radial basis function (RBF) network is illustrated in figure.



A radial Basis N/w.

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.

Training of RBF network

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process. First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_u^2 that define its kernel function K_u(d(x_u, x)). Second, the weights w_u are trained to maximize the fit of the network to the training data, using the global error criterion that minimizes the error between target and local approximation. Because the kernel functions are held fixed during this second stage, the linear weight values w_u, can be trained very efficiently.

Choice of the number of kernel functions (hidden untis)

1.Allocate a Gaussian kernel function for each training example (x_i, f (x_i)), centering this Gaussian at the point xi. Each of these kernels may be assigned the same width σ_2. Given this approach, the RBF network learns a global approximation to the target function in which each training example (x_i, f (x_i)) can influence the value of f only in the neighborhood of x_i. One advantage of this choice of kernel functions is that it allows the RBF network to fit the training data exactly.

2.A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large. The set of kernel functions may be distributed with centers spaced uniformly throughout the instance space X. Alternatively, we may wish to distribute the centers non-uniformly, especially if the instances themselves are found to be distributed non-uniformly over X.

To summarize, 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. Thus, 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 Backpropagation. This follows from the fact that the input layer and the output layer of an RBF are trained separately.

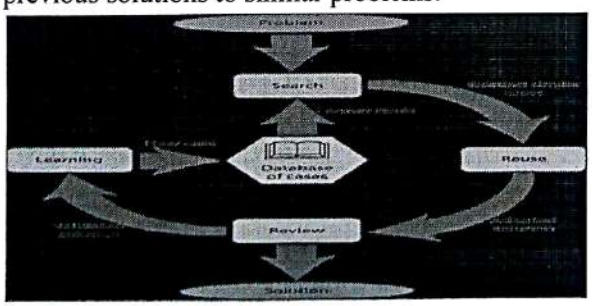
Case Based Reasoning

Instance-based methods such as k-NEAREST NEIGHBOR and locally weighted regression share three key properties.

1. they are lazy learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed.
2. they classify new query instances by ana- lyzing similar instances while ignoring instances that are very different from the query.
3. 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. In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.Each instance is described as a case history ,having both structure and function.
- Example:
  - ((user-complaint error53-on-shutdown)
  - (cpu-model PowerPC)
  - (operating-system Windows)
    (network-connection PCIA)
    (memory 48meg)
  - (installed-applications Excel Netscape VirusScan)
  - (disk 1gig)
    (likely-cause ???))

CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs reasoning about new legal cases based on previous rulings and solving planning andscheduling problems by reusing and combining portions of previous solutions to similar problems.

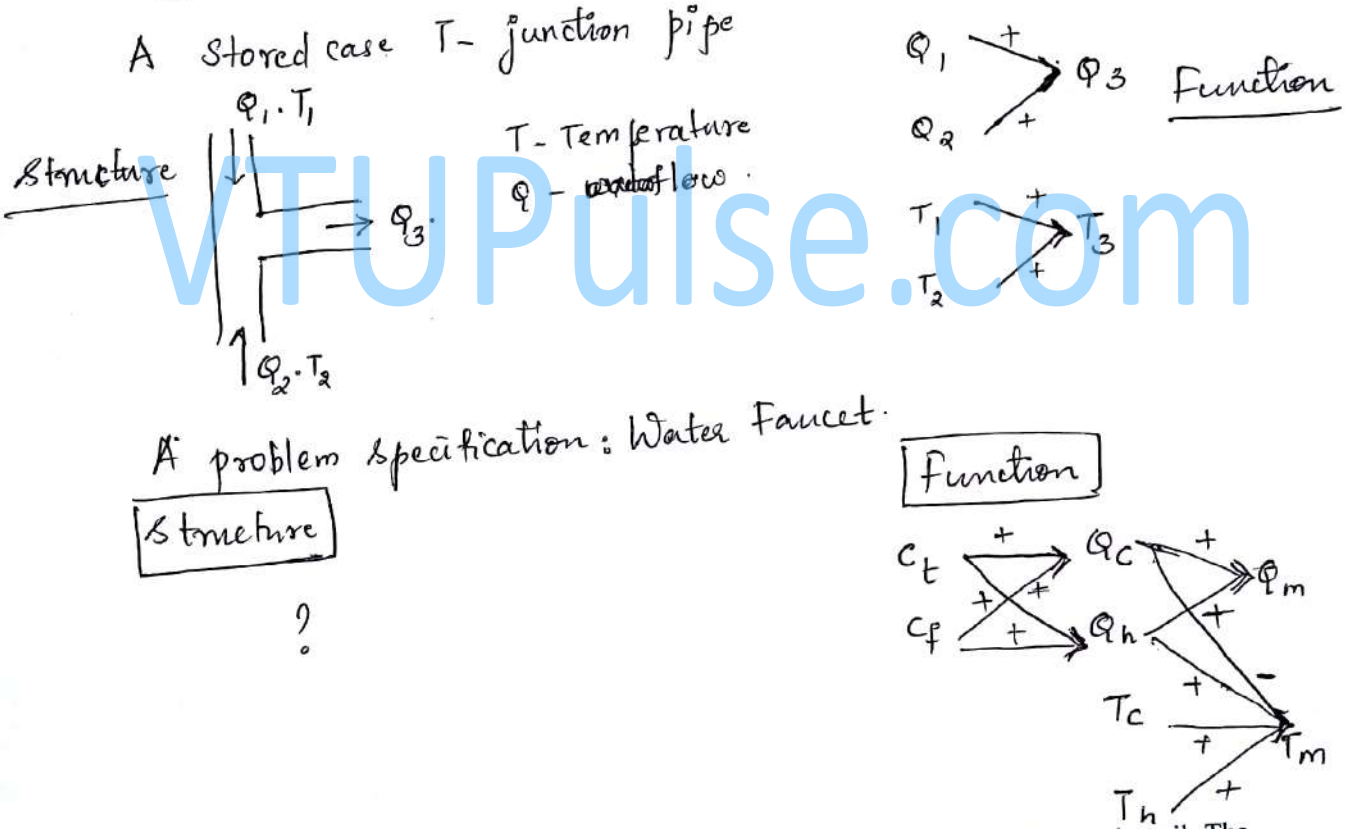


CBR using CADET system

- CADET system is a case based reasoning tool.
- It is popularly used for the conceptual design of electro-mechanical devices.

For instance, it uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems. Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function. New design problems are then presented by specifying the desired function and requesting the corresponding structure.

The top half of the figure shows the description of a typical stored case called a T-junction pipe. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs. In the functional description at its right, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. For example, the output waterflow Q3 increases with increasing input waterflow Ql.



a "-" label indicates that the variable at the head decreases with the variable at the tail. The bottom half of this figure depicts a new design problem described by its desired function. This particular function describes the required behavior of one type of water faucet. Here Q_c, refers to the flow of cold water into the faucet, Q_h to the input flow of hot water, and Q_m, to

the single mixed flow out of the faucet. Similarly, T_c, T_h, and T_m , refer to the temperatures of the cold water, hot water, and mixed water respectively. The variable C_t, denotes the control signal for temperature that is input to the faucet, and C_f denotes the control signal for waterflow. Note the description of the desired function specifies that these controls C_t, and C_f are to influence the water flows Q_c, and Q_h, thereby indirectly influencing the faucet output flow Q_m, and temperature T_m,. 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 isomorphisms between the two function graphs, so that parts of a case can be found to match parts of the design specification. Furthermore, 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 $A \rightarrow B$ as $A \rightarrow x \rightarrow B$

This rewrite rule can be interpreted as stating that if B must increase with A, then it is sufficient to find some other quantity x such that B increases with x, and x increases with A. Here x is a universally quantified variable whose value is bound when matching the function graph against the case library.

In fact, the function graph for the faucet shown in Fig. is an elaboration of the original - functional specification produced by applying such rewrite rules. By retrieving multiple cases that match different subgraphs, the entire de- sign can sometimes be pieced together. In general, the process of producing a final solution from multiple retrieved cases can be very complex. It may require designing portions of the system from first principles, in addition to merging retrieved portions from stored cases. It may also require backtracking on earlier choices of design subgoals and, therefore, rejecting cases that were previously retrieved. CADET has very limited capabilities for combining and adapting multiple retrieved cases to form the final design and relies heavily on the user for this adaptation stage of the process.

Differences between Case based systems and other instance based methods

1. Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET. This may require a similarity metric different from Euclidean distance, such as the size of the largest shared subgraph between two function graphs.
2. Multiple retrieved cases may be combined to form the solution to the new problem. This is similar to the k-NEARESNTEIGHBOR approach, in that multiple similar cases are used to construct a response for the new query. However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.

3. There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving. One simple example of this is found in CADET, which uses generic knowledge about influences to rewrite function graphs during its attempt to find matching cases. Other systems have been developed that more fully integrate case-based reasoning into general search- based problem-solving systems

Summary:

Case-based reasoning is an instance-based learning method in which instances (cases) may be rich relational descriptions and in which the re- trieval and combination of cases to solve the current query may rely on knowledge- based reasoning and search-intensive problem-solving methods.

Issues:

The central issue is that of syntactic similarity measures (e.g., subgraph isomorphism between function graphs) provide only an approximate indication of the relevance of a particular case to a particular problem. When the CBR system attempts to reuse the retrieved cases it may uncover difficulties that were not captured by this syntactic similarity measure. For example, in CADET the multiple retrieved design fragments may turn out to be incompatible with one another, making it impossible to combine them into a consistent final design. When this occurs in general, the CBR system may backtrack and search for additional cases, adapt the existing cases, or resort to other problem-solving methods. In particular, if a case is retrieved based on the similarity metric, but found to be irrelevant based on further analysis, then the similarity metric should be refined to reject this case for similar subsequent queries.

Lazy Vs Eager Learners	
Lazy	Eager
Defer the decision of how to generalize beyond the training data until each new query instance is encountered.	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
Local Approximation to the target function	Global Approximation to target function
Require less computation during training, but more computation when they must predict the target value for a new query	More computation time for training ,lesser time for predicting target
Inductive Bias: Lazy methods may consider the query instance x_q, when deciding how to generalize beyond the training data D.	Inductive Bias: Eager methods will have already built the global approximation before observing the query instance x_q
Variation in Generalization accuracy: A lazy learner has the option of (implicitly) representing the target function by a combination of many local approximations. Better accuracy for unseen instances because they are observed before approximation	Variation in Generalization accuracy: An eager learner must commit at training time to a single global approximation.

To summarize, lazy methods have the option of selecting a different hypothesis or local approximation to the target function for each query instance. Eager methods using the same

hypothesis space are more restricted because they must commit to a single hypothesis that covers the entire instance space. Eager methods can, of course, employ hypothesis spaces that combine multiple local approximations, as in RBF networks. However, even these combined local approximations do not give eager methods the full ability of lazy methods to customize to unknown future query instances.

REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where feedback provided to the agent is correct set of actions for performing a task, reinforcement learning uses rewards and punishment as signals for positive and negative behaviour.

As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in reinforcement learning the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. The figure below represents the basic idea and elements involved in a reinforcement learning model.

Some key terms that describe the elements of a RL problem are:

Environment: Physical world in which the agent operates

State: Current situation of the agent

Reward: Feedback from the environment

Policy: Method to map agent's state to actions

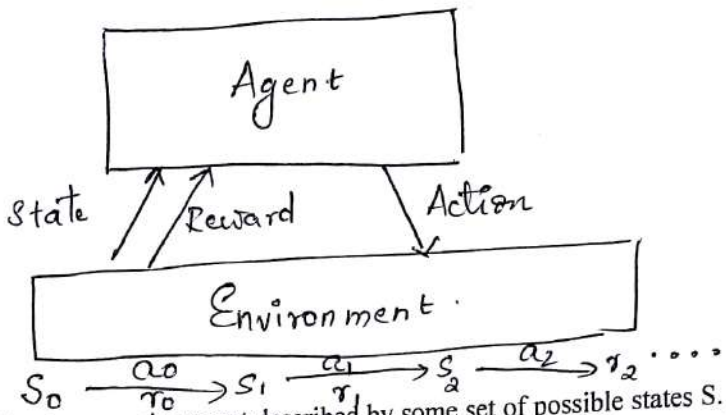
Value: Future reward that an agent would receive by taking an action in a particular state

Example:

Consider the environment of building a learning robot. The robot, or agent, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state. For example, a mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn." Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals. For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low. The agents can learn successful control policies by experimenting in their environment. We assume that the goals of the agent can be defined by a reward function that assigns a numerical value-an immediate payoff-to each distinct action the agent may take from each distinct state. For example, the goal of docking to the battery charger can be captured by assigning a positive reward (e.g., +loo) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition. This reward function may be built into the robot, or

known only to an external teacher who provides the reward value for each action performed by the robot. The task of the robot is to perform sequences of actions, observe their consequences, and learn a control policy. The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agents.

Figure:An agent interacting with its environment



The agent exists in an environment described by some set of possible states S. It can perform any of a set of possible actions A. Each time it performs an action a, in some state s_t the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure. The agent's task is to learn a control policy, P : S A, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals. This very generic problem covers tasks such as learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games. Each time the agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state. For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states. The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.

Differences between Reinforcement Learning and other function approximation tasks

Delayed reward. The task of the agent is to learn a target function π that maps from the current state s to the optimal action $a = \pi(s)$. In earlier learning tasks, each training example would be a pair of the form $(s, \pi(s))$. In reinforcement learning, however, training information is not available in this form. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of temporal credit assignment: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

Exploration. In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a tradeoff in choosing whether to favor exploration of

unknown states and actions (to gather new information), or exploitation of states and actions that it has already learned will yield high reward (to maximize its cumulative reward).

Partially observable states. Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. For example, a robot with a forward-pointing camera cannot see what is behind it. In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

Life-long learning. Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how-to pick-up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

The learning task in Reinforcement Learning

Various assumptions of the agent

- Agent's actions are deterministic or nondeterministic
- Agent can predict the next state that will result from each action or it cannot
- Agent is trained by an expert who shows it examples of optimal action sequences, or that it must train itself by performing actions of its own choice

The learning in this case is based on Markov Decision process.MDP are mathematical frameworks to describe an environment in reinforcement learning consisting of a set of states, actions, rewards, and a transition model from one state to another based on the action. Almost all RL problems can be formalized using MDPs.

- In a Markov decision process (MDP), the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
- At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t and performs it.
- The environment responds by giving the agent a reward $r_t=r(s_t, a_t)$ and by producing the succeeding state $s_{t+1} = \partial(s_t, a_t)$
- Here the functions ∂ and r are part of the environment and are not necessarily known to the agent.
- In an MDP, the functions $\partial(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.
- We consider only the case in which S and A are finite. In general, $\partial$and r may be nondeterministic functions, but we begin by considering only the deterministic case
- The task of the agent is to learn a policy, $\pi : S \ A$, for selecting its next action a, based on the current observed state s_t; that is, $\pi (s_t) = a_t$.
- The agent chooses the policy that produces the greatest possible cumulative reward for the robot over time. To state this requirement more precisely, we define the cumulative value $V^{\pi}(s_t)$ achieved by following an arbitrary policy π from an arbitrary initial state s_t as follows:

where the sequence of rewards r_{t+i} is generated by beginning at state s_t, and by repeatedly using the policy π to select actions as described above (i.e., $a_t = \pi(s_t)$, $a_{t+1} = \pi(s_{t+1})$, etc.). Here $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards. In particular, rewards received i time steps into the future are discounted exponentially by a factor of γ_i. Note if we set $\gamma = 0$, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.

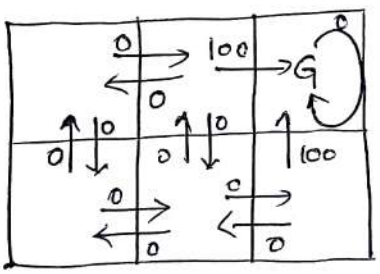
The quantity $V^{\pi}(s_t)$ is often called the discounted cumulative reward achieved by policy π from initial state s.

We are now in a position to state precisely the agent's learning task. We require that the agent learn a policy π that maximizes $V^{\pi}(s_t)$ for all states s. We will call such a policy an optimal policy and denote it by π^*.

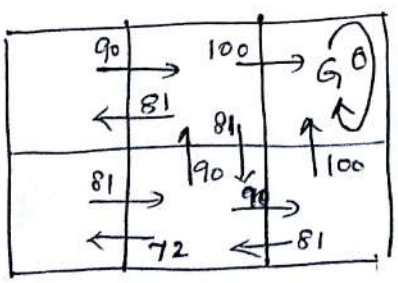
$$\pi^* = \text{argmax}_{\pi} V^{\pi}(s), (\forall s)$$

Example:

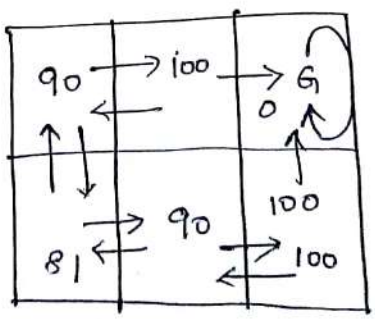
Consider a simple grid-world environment. The six grid squares in this diagram represent six possible states, or locations, for the agent. Each arrow in the diagram represents a possible action the agent can take to move from one state to another. The number associated with each arrow represents the immediate reward r(s, a) the agent receives if it executes the corresponding state-action transition. Note the immediate reward in this particular environment is defined to be zero for all state-action transitions except for those leading into the state labelled G. It is convenient to think of the state G as the goal state, because the only way the agent can receive reward, in this case, is by entering this state. Note in this particular environment, the only action available to the agent once it enters the state G is to remain in this state. For this reason, we call G an absorbing state. Once the states, actions, and immediate rewards are defined, and once we choose a value for the discount factor γ, we can determine the optimal policy π^* and its value function $V^*(s)$. In this case, let us choose $\gamma = 0.9$.



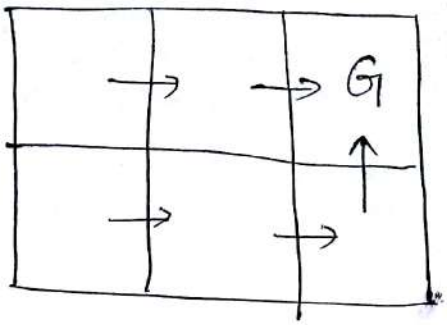
r (s, a) (Immediate reward) values.



Q (s, a) values



$V^*(s)$ values



One optimal policy.

Each grid square represents a distinct state, each arrow a distinct action. The immediate reward function, r(s, a) gives reward 100 for actions entering the goal state G, and zero otherwise. Values of V*(s) and Q(s,a) follow from r(s,a), and the discount factor $\gamma = 0.9$. An optimal policy, corresponding to actions with maximal Q values, is also shown.

Q learning

Q-learning is a reinforcement learning technique used in machine learning. The goal of Q-Learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations.

Learning an optimal policy $\pi^*: S \rightarrow A$ is difficult because the available training data does not provide training examples of the form (s,a). Instead, the only training information available to the learner is the sequence of immediate rewards $r(s_i, a_i)$ for i = 0, 1,2, . . . Given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.The easiest choice for the agent to learn the evaluation function is to choose V* .The agent should choose prefer state s1 over s2 if V*(s1)> V*(s2) The optimal action in state s is the action a that maximizes the sum of the immediate reward r(s, a) plus the value V* of the immediate successor state, discounted b γ.

$$\pi^*(s) = \text{argmax}_a[r(s, a) + \gamma V^*(\partial(s, a))] \text{ --------->(a)}$$

$\partial(s, a)$ denotes the state resulting from applying action a to state s.)

Thus, the agent can acquire the optimal policy by learning V*, provided it has perfect knowledge of the immediate reward function r and the state transition function ∂.But in many practical learning scenarios, the transitions and rewards may not be directly inferred.Hence it is required to modify the evaluation function that suits all general settings.

The Q function

Let us define the evaluation function Q(s, a) so that its value is the maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action. In other words, the value of Q is the reward received immediately upon executing

action a from state s, plus the value (discounted by γ) of following the optimal policy thereafter.

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\partial (s, a)) \text{----------} > (b)$$

Q(s, a) is exactly the quantity that is maximized inEqn(a) in order to choose the optimal action a in state s.Thus we can write (a) in terms of Q(s,a) as follows:

$$\pi^* (s) = \text{argmax}_a Q(s ,a)$$

If the agent learns the Q function instead of the V* function, it will be able to select optimal actions even when it has no knowledge of the functions r and ∂.It needs to consider each available action a in its current state s and choose the action that maximizes Q(s, a).

Q learning algorithm

For each s, a, initialize the table entry Q^(s, a) to zero.

Observe the current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for Q(s, a) as follows:

$$Q\hat{}(s,a) \leftarrow r + \gamma \max_{a'} Q\hat{}(s',a')$$

- s← s'

The key problem is finding a reliable way to estimate training values for Q, given only a sequence of immediate rewards r spread out over time. This can be accomplished through iterative approximation. To see how, notice the close relationship between Q and V*,

$V^*(S) = \max_{a'} Q(s, a')$.Hence

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\partial(s,a),a')$$

This recursive definition of Q provides the basis for algorithms that iteratively approximate Q. To describe the algorithm, we will use the symbol Q to refer to the learner's estimate, or hypothesis, of the actual Q function. In this algorithm the learner represents its hypothesis Q by a large table with a separate entry for each state-action pair. The table entry for the pair (s, a) stores the value for Q^(s, a)-the learner's current hypothesis about the actual but unknown value Q(s, a). The table can be initially filled with random values (though it is easier to understand the algorithm if one assumes initial values of zero). The agent repeatedly observes its current state s, chooses some action a, executes this action, then observes the resulting

reward r = r(s, a) and the new state s' =∂ (s, a). It then updates the table entry for Q^(s, a) following each such transition, according to the rule:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$