MODULE – 5

EVALUATING HYPOTHESIS, INSTANCE-BASED LEARNING, REINFORCEMENT LEARNING

PART 1- EVALUATING HYPOTHESIS

5.1. MOTIVATION

Importance to evaluate the performance,

1. To understand whether to use the hypothesis.
    - For instance, when learning from a limited-size database indicating the effectiveness of different medical treatments, it is important to understand as precisely as possible the accuracy of the learned hypotheses.
2. Evaluating hypotheses is an integral component of many learning methods.
    - For example, in post-pruning decision trees to avoid overfitting, we must evaluate resultant trees

- ➢ Data is plentiful → Accuracy is straightforward.
- ➢ Difficulties arise given limited set of data. They are

    1. Bias in the estimate.
        - ▪ The observed accuracy of the learned hypothesis over the training examples is often a poor estimator of its accuracy over future examples.
        - ▪ It is a biased estimate of hypothesis accuracy over future examples.
        - ▪ To obtain an unbiased estimate of future accuracy, we typically test the hypothesis on some set of test examples chosen independently of the training examples and the hypothesis.
    2. Variance in the estimate.
        - ▪ Even if the hypothesis accuracy is measured over an unbiased set of test examples, independent of the training examples, the measured accuracy can still vary from the true accuracy, depending on the makeup of the particular set of test examples.
        - ▪ The smaller the set of test examples, the greater the expected variance

5.2. ESTIMATING HYPOTHESIS ACCURACY

Setting:

There is some space of possible instances X (e.g., the set of all people) over which various target functions may be defined (e.g., people who plan to purchase new skis this year.

X→ different instances may be encountered with different frequencies.

Assumption: there is some unknown probability distribution $\mathcal{D}$ that defines the probability of encountering each instance in X. For example, $\mathcal{D}$ might assign a higher probability to encountering 19-year-old people than 109-year-old people.

$\mathcal{D}$→ only determines the probability that ***x*** will be encountered.

Learning Task→ to learn the target concept or target function *f* by considering a space *H* of possible hypotheses.

Trainer → Provides Training examples of the target function *f* to the learner

→ Draws instances independently according to the distribution $\mathcal{D}$

→ Forwards the instance x along with its correct target value *f(x)* to the learner

Illustration: Consider,

Target function to be learnt → people who plan to purchase new skis this year.

Given→ sample of training data collected by surveying people as they arrive at a ski resort.

X→ instance space of all people, who might be described by attributes such as their age, occupation, how many times they skied last year, etc.

$\mathcal{D}$ → Distribution → for each person ***x*** the probability that x will be encountered as the next person arriving at the ski resort.

Therefore,

Target function → *f: X: {0,1}* → for each person ***x*** the probability that x will be encountered as the next person arriving at the ski resort.

Questions to be explored

1. Given a hypothesis h and a data sample containing n examples drawn at random according to the distribution D, what is the best estimate of the accuracy of h over future instances drawn from the same distribution?
2. What is the probable error in this accuracy estimate?

5.2.1. Sample Error and True Error

These are the two notations of accuracy or equivalently error.

Sample Error → error rate of the hypothesis over the sample of data that is available

True Error→ error rate of the hypothesis over the entire unknown distribution $\mathcal{D}$ of examples.

<u>Sample Error</u>

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

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→ number of examples in S

$\delta(f(x),h(x))$→ is 1 if $f(x) \neq h(x)$ otherwise 0.

<u>True Error</u>

The ***true error*** of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution $\mathcal{D}$**.**

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

$$error_{\mathrm{D}}(h) \equiv Pr_{x\in\mathrm{D}} f(x) \neq h(x)]$$

Where, $Pr_{x\in\mathrm{D}}$→ denotes that the probability is taken over the instance distribution $\mathcal{D}$.

<u>Requirement</u>

We need to Calculate *error$_\mathcal{D}$(h)* but we have *error$_s$(h)* in hand.

Question→ How good an estimate of *error$_\mathcal{D}$(h)* is provided by *error$_s$(h)*?

5.2.2. Confidence Intervals for Discrete-Valued Hypothesis

Answers the question How good an estimate of *error$_\mathcal{D}$(h)* is provided by *error$_s$(h)*?

Here, h→ discrete-valued function.

<u>To estimate the true error for some discrete-valued hypothesis **h**, based on its observed sample error over a sample S</u>

Given,

- Sample S→ ***n*** examples drawn independent of one another and of *h*, according to the probability distribution $\mathcal{D}$.
- n≥30
- *h* → commits ***r*** errors over these ***n*** examples (i.e., ***error$_s$(h)=r/n)***

The statistical theory allows us to make the following assertions:

1. Given no other information, the most probable value of *error$_\mathcal{D}$(h)* is *error$_s$(h).*
2. With approximately ***95%*** probability, the true error *error$_\mathcal{D}$(h)* lies in the interval

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1-error_S(h))}{n}} \qquad \text{--- (i)}$$

Example Problem for Illustration

Given:

- S contains 40 examples → n=40
- *h* commits 12 errors → *r* =12

Therefore,

Sample Error→ $error_S(h)$=12 / 40 = 0.30

Since no other information is given, use $error_S(h)$ to calculate $error_{\mathcal{D}}(h)$.

From experiments it is noticed that, on drawing independent samples drawing 40 examples every time resulted that **approximately for 95% of these experiments, the calculated interval would contain the true error**. → 95% confidence interval estimate for $error_{\mathcal{D}}(h)$.

For current example, *h* commits r = 12 errors over a sample of n = 40 independently drawn examples, the confidence interval is given by

$$\text{Confidence Interval} \rightarrow \; error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

$$\rightarrow 0.30 \pm 1.96\sqrt{\frac{0.30\times(1-0.30)}{40}}$$

$$\rightarrow 0.30 \pm 1.96\sqrt{\frac{0.30\times0.70}{40}}$$

$$\rightarrow 0.30 \pm 1.96\sqrt{\frac{0.21}{40}}$$

$$\rightarrow 0.30 \pm 1.96\sqrt{0.00525}$$

$$\rightarrow 0.30 \pm 1.96 \times 0.07$$

Confidence Interval → $\mathbf{0.30 \pm 0.14}$

The Eq. (i) given for 95% confidence interval can be generalized any desired confidence level. Let Z_N be used to calculate *N%* confidence intervals for $error_{\mathcal{D}}(h)$.

There Eq. (i) can be rewritten as

$$error_S(h) \pm Z_N\sqrt{\frac{error_S(h)(1-error_S(h))}{n}} \qquad \text{--- (5.1)}$$

Where, Z_N → constant →chosen depending on the desired confidence level (Table 5.1)

Table 5.1. Values of Z_N for two-sided *N%* confidence intervals

Confidence Level *N%* :	50%	68%	80%	90%	95%	98%	99%

Constant Z_N :	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Therefore, for the above example, if 68% is the confidence level, then we get the confidence interval as $\mathbf{0}0.30 \pm 1.00 \times 0.07$

Eq. (5.1.) → describes how to calculate the confidence intervals, or error bars, for estimates of *error$_{\mathcal{D}}$(h)* that are based on *error$_S$(h)***.**

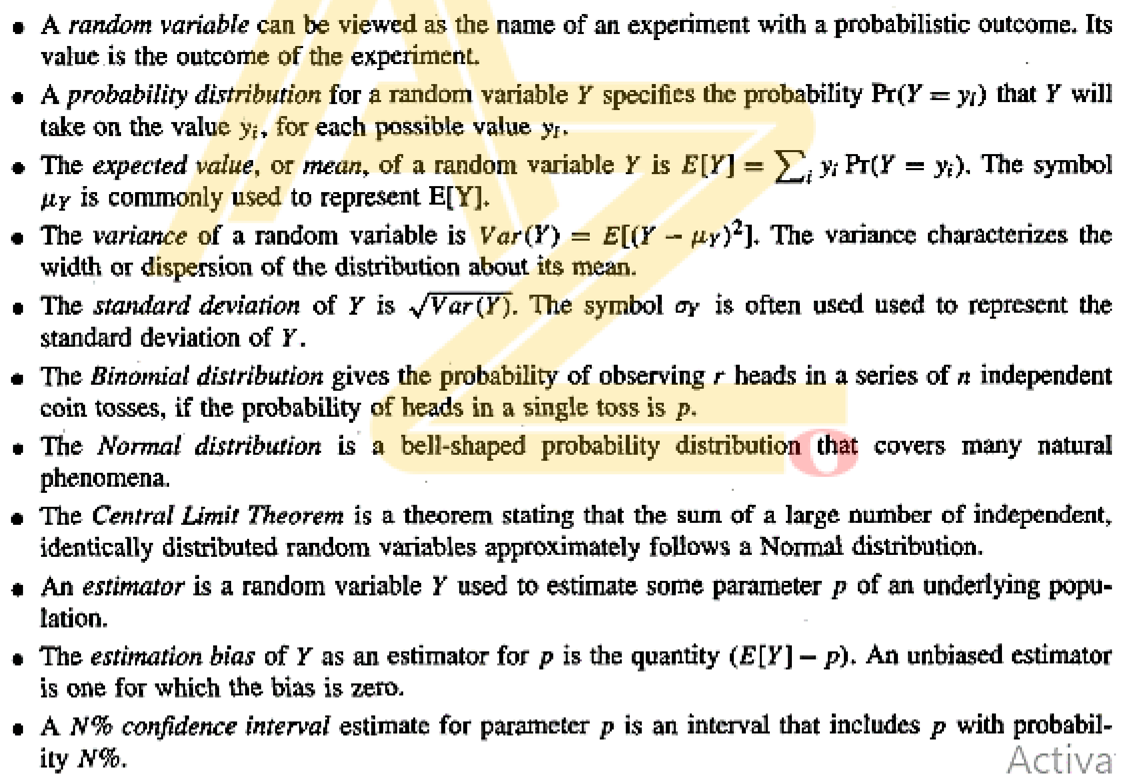
→ provides only an approximate confidence interval, though the approximation is quite good when the sample contains at least ***30*** examples, and *error$_S$(h)* is not too close to ***0*** or ***1***

Rule of thumb → The above approximation works well when

$$n.\, error_S(h)(1 - error_S(h)) \geq 5$$

5.3. BASICS OF SAMPLING THEOREM

Summary



- 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%.

5.3.1. Error Estimation and Estimating Binomial Proportions

Precisely how does the deviation between sample error and true error depend on the size of the data sample? → The property of interest is that *h* misclassifies the example.

To calculate sample error

- First, collect a random sample **S** of ***n*** independently drawn instances from the distribution ***D***
- Then measure the sample error $error_s(h)$

Each time drawing a different random sample S_i of size n, → we observe different values for the various $error_{s_i}(h)$ depending on random differences in the makeup of the various S_i.

Therefore, $error_{s_i}(h)$→ the outcome of the ith such experiment, is a random variable.

Example,

To run k random experiments measuring the random variables, $error_{s_1}(h), error_{s_2}(h)$,…, $error_{s_k}(h)$. As k grows, the histogram approach the form of distribution (Figure 5.1).

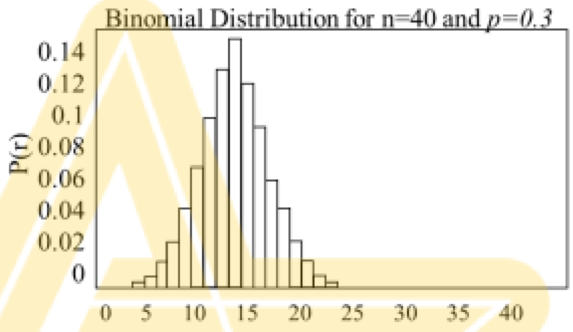


Figure 5.1. The Binomial Distribution

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 with the same mean and variance.

Note: Use the Normal approximation only when $np(1-p) \geq 5$.

5.3.2. The Binomial Distribution

Consider the following **example**.

Given: a worn and bent coin

Task→ to estimate the probability that the coin will turn up heads when tossed→ *p*

Process→ Toss the coin *n* times and record the number of times *r* that it turns up heads.

Reasonable estimate of *p* is *r/n*.

When the experiment is rerun for new set of *n* coin tosses the value *r* may vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for *p*.

Binomial Distribution → for each possible value of *r* (i.e., from 0 to n), the probability of observing exactly *r* heads given a sample of *n* independent tosses of a coin whose true probability of heads is *p*.

Estimating *p* from a random sample of coin tosses is equivalent to estimating $error_D(h)$ from testing h on a random sample of instances.

- A single toss of the coin corresponds to drawing a single random instance from $\mathcal{D}$ and determining whether it is misclassified by *h*.
- The probability p that a single random coin toss will turn up heads corresponds to the probability that a single instance drawn at random will be misclassified → *p* corresponds to $error_D(h)$.
- *r* → number of heads observed over a sample of n coin tosses
  → number of misclassifications observed over n randomly drawn instances
- *r/n* →$error_S(h)$

Here, the problem of estimating *p* for coins is identical to the problem of estimating $error_D(h)$ for hypotheses.

Binomial Distribution → general form of the probability distribution for the random variable *r*, whether it represents the number of heads in *n* coin tosses or the number of hypothesis errors in a sample of *n* examples.

<u>The general setting to which the Binomial distribution applies is:</u>

- There is a base, or underlying, experiment (e.g., toss of the coin) whose outcome can be described by a random variable, say *Y* . The random variable *Y* can take on two possible values (e.g., *Y* = 1 if heads, Y = 0 if tails)
- The probability that *Y* = 1 on any single trial of the underlying experiment is given by some constant p, independent of the outcome of any other experiment. The

probability that $Y = 0$ is therefore (1- p). Typically, p is not known in advance, and the problem is to estimate it.

- **A** series of n independent trials of the underlying experiment is performed (e.g., n independent coin tosses), producing the sequence of independent, identically distributed random variables Y_1 , Y_2,...,Y_n. Let R denote the number of trials for which $\boldsymbol{Y_i}$= 1 in this series of n experiments

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

- The probability that the random variable $\boldsymbol{R}$ will take on a specific value r (e.g., the probability of observing exactly r heads) is given by the Binomial distribution (Figure 5.1.)

$$Pr(R = r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \qquad \text{--- (5.2)}$$

5.3.3. Mean and Variance

<u>Mean → Expected value</u>

The expected value is the average of the values taken on by repeatedly sampling the random variable.

Definition: Consider a random variable Y that takes on the possible values $y_1, . . . y_n$. The expected value of Y , $E[Y]$,is

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i) \qquad \text{--- (5.3)}$$

Example,

Consider, Y = 1 with probability 0.7 and 2 with probability 0.3.

Then, Expected value is $1 \cdot 0.7 + 2 \cdot 0.3 = 1.3$

Note: If random variable Y is governed by a Binomial distribution, then it can be shown that $E[Y] = np$ ---- (5.4)

<u>Variance</u>

It captures the "width or "spread" of the probability distribution → how far the random variable is expected to vary from its mean value.

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

$$Var[Y] \equiv E[(Y - E[y])^2] \qquad \text{---(5.5)}$$

Variance→ describes the expected squared error in using a single observation of Y to estimate its mean $E [Y]$

<u>Standard Deviation</u>

The square root of the variance is called the *standard deviation* of Y , denoted by σ_Y.

Definition: The **standard deviation** of a random variable Y, σ_Y, is

$$\sigma_Y \equiv \sqrt{E[(Y - E[y])^2} \qquad \text{---(5.6)}$$

For instance, if the random variable Y is governed by a Binomial distribution, then the variance and standard deviation are given by

$$Var[Y] = np(1-p)$$

$$\sigma_Y \equiv \sqrt{np(1-p)} \qquad \text{---(5.7)}$$

5.3.4. Estimators, Bias, and Variance

Question →What is the likely difference between *error$_S$(h)* and the true error *error$_D$(h)***?**

Rewrite *error$_S$(h)* and *error$_D$(h)* using terms in Eq.(5.2) defining the Binomial distribution:

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

$$errror_D(h) = p$$

Where,

n→ number of instances in the sample S

r → number of instances from S misclassified by *h*

p→ probability of misclassifying a single instance drawn from *D*.

<u>Estimation Bias</u>

error$_S$(h) → an estimator for true error *error$_D$(h)*.

An estimator is any random variable used to estimate some parameter of the underlying population from which the sample is drawn.

Estimation bias→ difference between the expected value of the estimator and the true value of the parameter.

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

$$E[Y] = p$$

If the estimation bias is zero → Y is an ***unbiased estimator*** for ***p.***

Question → Is ***error$_S$(h)*** an unbiased estimator for ***error$_D$(h)?***

Yes.

- Binomial distribution the expected value of ***r*** is equal to ***np.***
- Given that ***n*** is a constant, that the expected value of ***r/n*** is p.

Another property of an estimator is its *variance*. Given a choice among alternative unbiased estimators choose the one with least variance → yields the smallest expected squared error between the estimate and the true value of the parameter.

Example, for $r = 12$ and $n = 40$, an unbiased estimate for *error$_D$(h)* is given by

$$error_S(h) = r/n = 0.3.$$

The variance arises from r since n is a constant. Given r → Binomially distributed. Therefore, Variance is given by Eq. (5.7) → $\boldsymbol{np(1-p)}$

Here, p is unknown and can be replaced by estimator r/n. Hence, we get,

$$\text{Estimated variance} = 40 \cdot 0.30(1 - 0.30) = 8.4 \text{ and}$$

$$\text{Standard Deviation} = \sqrt{8.4} \approx 2.9$$

Therefore, standard deviation in $error_S(h) = r/n$ is approximately 2.9/40 = 0.07.

Hence, here, $error_S(h)$=0.30 with standard deviation of approximately 0.07. Given r errors in a sample of n independently drawn test examples, the standard deviation for $error_S(h)$is given by

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}} \qquad \text{--- (5.8)}$$

Eq. 5.8. can be approximated by substituting r/n = $error_S(h)$ for p.

$$\sigma_{error_S(h)} \approx \sqrt{\frac{\sigma_{error_S(h)}(1-\sigma_{error_S(h)})}{n}} \qquad \text{--- (5.9)}$$

5.3.5. Confidence Interval

To describe the uncertainty associated with an estimate, give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval → **Confidence Interval estimates**.

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

Example, Given $r = 12$ and $n = 40$ approximately *95% probability* that the interval *0.30 f0.14* contains the true error $error_D(h)$.

Normal Distribution

It is a bell-shaped distribution fully specified by its mean μ and standard deviation σ (FIgure 5.2).

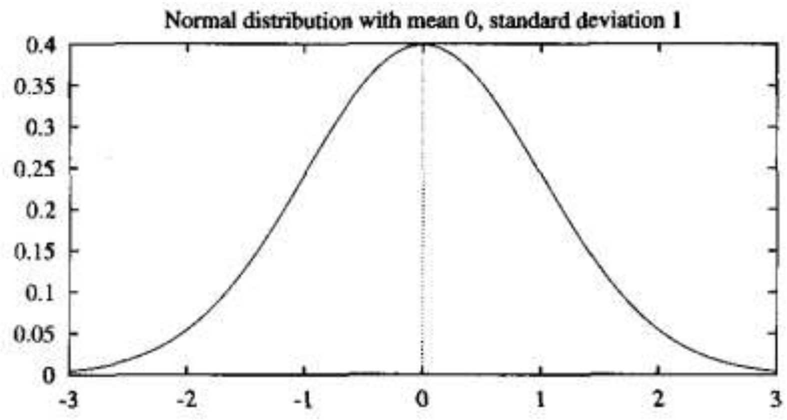


Figure 5.2: Normal Distribution

For large n, any Binomial distribution is very closely approximated by a Normal distribution with the same mean and variance.

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

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

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:

1. The probability that X will fall into the interval (a,***6)***is given by

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

2. The expected or mean value of X, *E[X]*, is

$$E[X] = \mu$$

3. The variance of X, *Var(X)*, is

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

4. The standard deviation of X, σ_X , is

$$\sigma_X = \sigma$$

<u>Why to use Normal Distribution?</u>

Most statistics references give tables specifying the size of the interval about the mean that contains ***N%*** of the probability mass under the Normal distribution → needed to calculate N% confidence interval.

From Table 5.1,

Z_N → width of the smallest interval about the mean that includes ***N%*** of the total probability mass under the bell-shaped Normal distribution.

→ gives half the width of the interval (i.e., the distance from the mean in either direction) measured in standard deviation (Figure 5.3 → $Z_{0.80}$)

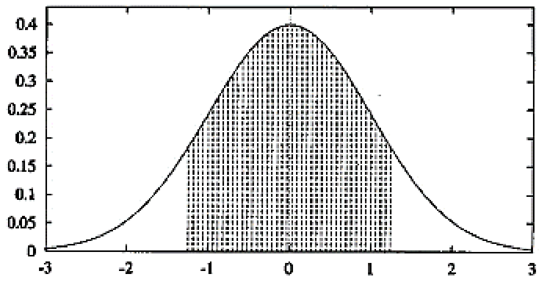


Figure 5.3: A Normal distribution with mean *0*, standard deviation 1: With 80% confidence, the value of the random variable will lie in the two-sided interval [-1.28,1.28]. Note $Z_{.80}$ = 1.28. With 10% confidence it will lie to the right of this interval, and with 10% confidence it will lie to the left.

Therefore,

If a random variable *Y* obeys a Normal distribution with mean μ and standard deviation σ, then the measured random value *y* of *Y* will fall into the following interval ***N%*** of the time

$$\mu \pm Z_N \sigma \quad \text{--- (5.10)}$$

Equivalently, the mean ***μ*** will fall into the following interval ***N%*** of the time

$$y \pm Z_N \sigma \quad \text{--- (5.11)}$$

<u>Derivation of General Expression for N% confidence interval:</u>

We know that,

- $error_S(h)$follows a Binomial distribution with mean value $error_D(h)$ and standard deviation as in Eq. 5.9.
- For sufficiently large sample size n, the Binomial distribution is well approximated by a Normal distribution.
- Eq. 5.11 is used to find the ***N%*** confidence interval for estimating the mean value of a Normal distribution.

Substituting the mean and standard deviation of $error_S(h)$ into Eq. 5.11 we get Eq. 5.1. for N% confidence intervals for discrete-valued hypotheses.

$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h)(1-error_S(h))}{n}} \qquad \text{--- (5.1)}$$

Two approximations used to derive Eq. 5.1 are

1. in estimating the standard deviation σ of $error_S(h)$ → approximated $error_d(h)$ by $error_S(h)$. [Eq. 5.8 to Eq. 5.9]
2. the Binomial distribution has been approximated by the Normal distribution.

Note:

- The two approximations are very good as long as n≥30, or when np(1-p)≥5.
- For small *n* use Binomial Distribution.

5.3.6. Two-Sided and One-Sided Bounds

Eq. 5.1 is two-sided bound→ it bounds the estimated quantity from above and from below.

One-sided bound → Example,

Question of interest → What is the probability that $error_D(h)$ is at most U?

This means it reflects bounding the maximum error of h and do not mind if the true error is much smaller than estimated.

<u>To find one-sided error bounds</u>

Normal distribution is symmetric about its mean. Therefore, any two-sided confidence interval based on a Normal distribution can be converted to a corresponding one-sided interval with twice the confidence (Figure 5.3 (b)).

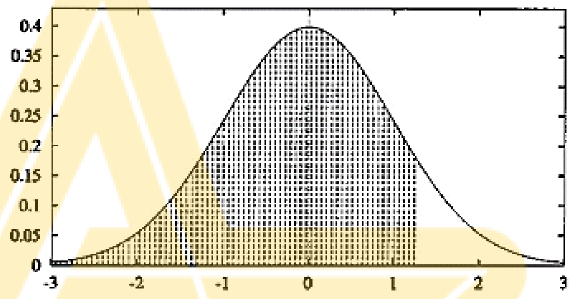


Figure 5.3: A Normal distribution with mean *0*, standard deviation 1: With 90%confidence, it will lie in the one-sided interval $[-\infty, 1.28]$.

That is,

100(1- α)% confidence interval with lower bound L and upper bound ***U***

→100(1-α/2)% confidence interval with lower bound L and no upper bound, or

→100(1-α/2)% confidence interval with lower bound U and no lower bound.

Where,

α→ probability that the correct value lies outside the stated interval, where

- α → probability that the value will fall into the unshaded region in Figure 5.2 and
- α/2 → probability that it will fall into the unshaded region of Figure 5.3.

<u>Illustration</u>

Example, *h* commits r = 12 errors over a sample of n = 40 independently drawn examples. We get two-sided 95% confidence interval of 0.30±0.14. Here,

100(1- α) = 95% → α = 0.05 and

100(1- α/2) = 97.5% → $error_D(h)$ is at most 0.30+0.14 = 0.44 with no assertion about the lower bound on $error_D(h)$. [100(1-(0.05/2))=97.5].

This represents one-sided error bound on *error_D(h)* with double the confidence than the corresponding two-sided bound.

5.4. GENERAL APPROACH OF DERIVING CONFIDENCE INTERVALS

The general process includes the following steps:

1. Identify the underlying population parameter p to be estimated, for example, $error_D(h)$.
2. Define the estimator Y. (E.g., $error_S(h)$). It is desirable to choose a minimum-variance, unbiased estimator.
3. Determine the probability distribution D_Y that governs the estimator Y, including its mean and variance.
4. Determine the ***N%*** confidence interval by finding thresholds L and U such that ***N%*** of the mass in the probability distribution D_Y falls between L and U.

5.4.1. Central Limit Theorem

Theorem 5.1: Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean, $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$.

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 the mean and variance of $\bar{Y}$ can be used to determine the mean and variance of the individual Y_i.
- Implies that whenever we define an estimator that is the mean of some sample (e.g., ***$error_s(h)$*** is the mean error), the distribution governing this estimator can be approximated by a Normal distribution for sufficiently large ***n.***

5.5. DIFFERENCE IN ERROR OF TWO HYPOTHESIS

Consider 2 hypothesis h_1 and h_2 for a discrete-valued target function where,

- h_1 has been tested on a sample S_1 containing n_1 randomly drawn examples, and
- h_2 has been tested on an independent sample S_2 containing n_2 randomly drawn from the same distribution.

The difference d between the true errors of these two hypotheses is given by

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

Here, to derive a confidence interval estimate for *d* is to be calculated using an estimator. That is a sample error can be an estimator and let this difference be represented as $\hat{d}$.

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

It can be shown that $\hat{d}$ gives an unbiased estimate of *d* → $E[\hat{d}]=d$.

What is the probability distribution governing the random variable $\hat{d}$?

We know that,

- For large n_1 and n_2 ,both $error_{S_1}(h_1)$ and $error_{S_2}(h_2)$follow the distributions that are approximately Normal.
- The difference of two Normal distributions is also a Normal distribution, $\hat{d}$ will also follow a distribution that is approximately Normal, with mean *d*.

It can be shown that the variance of this distribution is the sum of the variances of $error_{S_1}(h_1)$ and $error_{S_2}(h_2)$.

Use Eq. 5.9 to obtain the approximate variance of each of these distributions, to get

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)\left(1-error_{S_1}(h_1)\right)}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2} \qquad \text{--- (5.12)}$$

Next derive confidence intervals that characterize the likely error in employing $\hat{d}$ to estimate d. For a random variable $\hat{d}$ obeying a Normal distribution with mean *d* and variance σ^2, the *N%* confidence interval estimate for *d* is $\hat{d} \pm Z_N\sigma$.

Using the approximate variance $\sigma_{\hat{d}}^2$, the approximate *N%* confidence interval estimate for *d* is

$$\hat{d} \pm Z_N\sqrt{\frac{error_{S_1}(h_1)\left(1-error_{S_1}(h_1)\right)}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}} \quad \text{--- (5.13)}$$

Eq. 5.13 → general two-sided confidence interval for estimating the difference between errors of two hypotheses.

→ Acceptable to be used where h_1 and h_2 are tested on a single sample *S* (where *S* is still independent of h_1 and h_2). Hence, $\hat{d}$ can be redefined as

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

5.5.1. Hypothesis Testing

What is the probability that $error_D(h_1) > error_D(h_2)$?

Let the sample errors for h_1 and h_2 are measured using two independent samples S_1 and S_2 of size 100 and find that

$$error_{S_1}(h_1) = 0.30 \text{ and}$$

$$error_{S_2}(h_2) = 0.20$$

And observed that the difference $\hat{d}$ =0.10.

What is the probability that *error*$_D$*(h*$_1$*)* > *error*$_D$*(h*$_2$*)* given the observed difference in sample errors $\boldsymbol{\hat{d}}$ =0.10 in this case? Equivalently, **what is the probability that *d>0*, given that $\boldsymbol{\hat{d}}$ =0.10.**

The probability Pr(d>0) is equal to the probability that $\boldsymbol{\hat{d}}$ has not overestimated *d* by more than 0.10 or this is the probability that $\boldsymbol{\hat{d}}$ falls into the one-sided interval $\boldsymbol{\hat{d} < d + 0.10}$ **or** $\boldsymbol{\hat{d} < \mu_{\hat{d}} + 0.10}$ (since *d* is the distribution governing $\hat{d}$)

Re-expressing the interval $\boldsymbol{\hat{d} < \mu_{\hat{d}} + 0.10}$ in terms of number of Standard Deviations, it allows deviating from the Mean. From Eq. 5.12, it is observed that $\sigma_{\hat{d}} \approx$ 0.061. Hence on re-expressing the interval we get,

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

[NOTE for reference: Use $\sigma_{\hat{d}} \approx 0.061$ in $\boldsymbol{\hat{d} < \mu_{\hat{d}} + 0.10}$ to get $1.64\sigma_{\hat{d}}$

$\frac{0.1}{0.061} = 1.64$ therefore 0.10 can be expressed as 1.64×0.061=$1.64\sigma_{\hat{d}}$]

5.6. COMPARING LEARNING ALGORITHMS

Comparing the performance of two learning algorithms L_A and L_B rather than comparing two specific hypothesis.

What is an appropriate test for comparing learning algorithms, and how can we determine whether an observed difference between the algorithms is statistically significant?

Task: <u>Determine which of L_A and L_B is the better learning method on average for learning some particular target function *f*.</u>

"On average" → consider the relative performance of these two algorithms averaged over all the training sets of size n that might be drawn from the underlying instance distribution $\mathcal{D}$. That is, **estimate the expected value of the difference in errors**

$$E_{S\subset D}^{[error_D(L_A(S)) - error_D(L_B(S))]} \quad \text{---(5.14)}$$

Here,

L(S) → hypothesis output by learning method *L* when given the sample *S* of training data

$S \subset D$→ the expected value is taken over samples S drawn according to the underlying instance distribution $\mathcal{D}$.

Eq. 5.14 → expected value of the difference in errors between learning methods L_A and L_B.

Consider a limited sample D_0 for comparing algorithms.

- Divide D_0 into a training set S_0 and a disjoint test set T_0 .
  - The training data can be used to train both L_A and L_B.
  - The test set can be used to compare the accuracy of the two learned algorithms.

Here we measure the quantity,

$$error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0)) \text{ ---(5.15)}$$

Difference between Eq. 5.14 and Eq. 5.15

- $error_{T_0}(h)$ is used to approximate $error_D(h)$
- Difference in errors is measured for one training set S_0 rather than using the entire sample S.

To improve the estimator in Eq. 5.15

- Repeatedly partition the data D_0 into disjoint training and test sets and
- Take the mean of the test set errors for these different experiments

Result → Procedure below

Procedure: to estimate the difference in error between L_A and L_B.

```
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
```

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=0}^{k} \delta_i$$

--- (T5.1)

Steps in Algorithm above

- Partition the data into k disjoint subsets of equal size, where this size is at least 30

- Train and test the learning algorithms k times, using each of the k subsets in turn as the test set, and use all remaining data as the training set.
- After testing for all k independent test sets, return the mean difference $\bar{\delta}$ that represents an estimate of the difference between the two learning algorithms.

$\bar{\delta}$ can be taken as an estimate of the desired quantity from Eq. 5.14. Hence $\bar{\delta}$ is an estimatge of the quantity

$$E_{S\subset D_0}^{[error_D(L_A(S))-error_D(L_B(S))]} \quad \text{---(5.16)}$$

S→ a random sample of size $\frac{k-1}{k}|D_0|$ drawn uniformly from D_0.

The approximate ***N%*** confidence interval for estimating the quantity in Eq. 5.16 using $\bar{\delta}$ is given by

$$\bar{\delta} \pm t_{N,k-1}S_{\bar{\delta}} \quad \text{--- (5.17)}$$

Where,

$t_{N,k-1}$→constant→ analogous to Z_N

$S_{\bar{\delta}}$→estimate of the Standard Deviation of the distribution governing $\bar{\delta}$. It is derfined as

$$S_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)}\sum_{i=1}^{k}\left(\delta_i-\bar{\delta}\right)^2} \quad \text{---(5.18)}$$

In $t_{N,k-1}$, N → desired confidence level

k-1→ number of ***degrees*** of ***freedom*** (also denoted by v)

→ related to the number of independent random events that go into producing the value for the random variable $\bar{\delta}$

Table 5.2: Values of $t_{N,v}$ two-sided confidence intervals. As v→∞, $t_{N,v}$ approaches Z_N.

	Confidence Interval			
	90%	95%	98%	99%
v = 2	2.92	4.30	6.96	9.92
v = 5	2.02	2.57	3.36	4.03
v = 10	1.81	2.23	2.76	3.27
v = 20	1.72	2.09	2.53	2.84
v = 30	1.70	2.04	2.46	2.75

v = 120	1.66	1.96	2.36	2.62
v = ∞	1.64	1.98	2.33	2.58

Paired Tests

Tests where the hypotheses are evaluated over identical samples are called ***paired tests.*** Paired tests typically produce tighter confidence intervals because any differences in observed errors in a paired test are due to differences between the hypotheses.

Note: when the hypotheses are tested on separate data samples, differences in the two sample errors might be partially attributable to differences in the makeup of the two samples.

5.6.1. Paired *t* Tests

To understand the justification for the confidence interval estimate given by Eq. 5.17, consider,

- Given, the observed values of a set of independent, identically distributed random variables $Y_1, Y_2, \ldots, Y_k$.
- Mean μ of the probability distribution governing Y_i.
- The estimator used is sample mean $\bar{Y}$

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

The *t* test represented by Eq. 5.17 and Eq. 5.18 applies to the case in which Y_i follow a Normal Distribution. The *t* test applies to the situation in which task is to estimate the sample mean of a collection of independent, identically and Normally distributed random variables. Using Eq. 5..17 and Eq. 5.18 we get,

$$\mu = \bar{Y} \pm t_{N,k-1} S_{\bar{Y}}$$

Where, $S_{\bar{Y}}$ is the estimated standard deviation of the sample mean

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

The t distribution is a bell-shaped distribution similar to the Normal distribution, but wider and shorter to reflect the greater variance introduced by using $S_{\bar{Y}}$ to approximate the true standard deviation $\sigma_{\bar{Y}}$.

PART 2- INSTANCE-BASED LEARNING

5.7. INTRODUCTION

Examples → nearest neighbor and locally weighted regression

Learning →

Storing the presented training data

When a new query instance is encountered

Retrieve set of similar related instances from memory

Classify the new instances using the retrieved instances

Key Difference→ Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified

Advantage → can use more complex, symbolic representations for instances→ case-based reasoning

→ case-based reasoning

- storing and reusing past experience at a help desk
- reasoning about legal cases by referring to previous cases
- solving complex scheduling problems by reusing relevant portions of previously solved problems

Disadvantage

i. The cost of classifying new instances can be high.

Reason→ nearly all computation takes place at classification time rather than when the training examples are first encountered

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

5.8. *k*-NEAREST NEIGHBOR LEARNING

Assumes all instances correspond to points in the n-dimensional space $\Re^n$

The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.

Consider,

Let instance *x* be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$$

Where,

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

The distance between two instances x_i and x_j being $d(x_i, x_j)$ is given by

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

Target function may be either discrete-valued or real-valued.

Consider learning discrete-valued target functions of the form

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

Where,

$V \rightarrow$ finite set $\{v_1, \dots, v_s\}$

The *k*-Nearest Neighbor Algorithm for approximating a discrete-valued target function

Training algorithm:

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

Classification Algorithm:

- Given a query instance x_q to be classified
    - Let $x_1 \dots x_k$ 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 $a == b$

$\delta(a, b) = 0$ otherwise

Operation of *k*-Nearest Neighbor

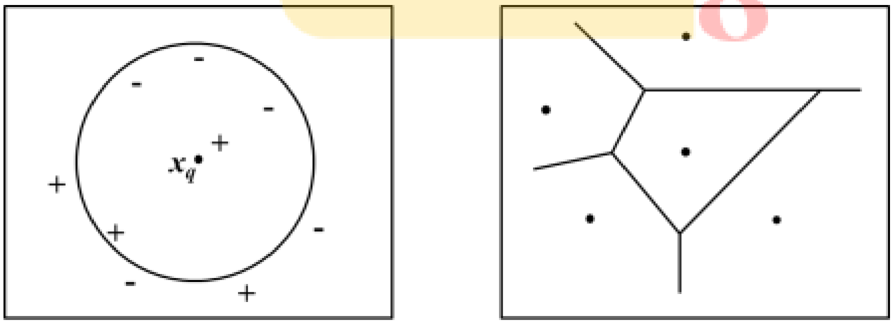


Figure 5.3: k-NEAREST NEIGHBOR.

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

What is the nature of the hypothesis space H implicitly considered by the *k*-Nearest Neighbor algorithm?

k-Nearest Neighbor algorithm never forms an explicit general hypothesis $\hat{f}$ regarding the target function *f*.

What does it do?

Just computes the classification of each new query instance as needed .

From Figure, *k*-Nearest Neighbor represents the decision surface as required by classification

The decision surface 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.→ the ***Voronoi diagram*** of the set of training examples.

The *k*-Nearest Neighbor algorithm is easily adapted to approximating continuous-valued target function.

Update Algorithm to

- Calculate the mean value of the k nearest training examples rather than calculate their most common value.
- replace the final line of the algorithm by

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

5.8.1. Distance-Weighted Nearest Neighbor Algorithm

Refinement to *k*-Nearest Neighbor algorithm 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.

Replace the final line in algorithm by

$$\boldsymbol{\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} w_i \delta(v, f(x_i))} \qquad \text{---(2)}$$

Where,

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \qquad \text{---(3)}$$

In case the distance is zero, $d(x_q,x_i)^2$ becomes zero.

Solution→ assign $\hat{f}(x_q)$ to be $f(x_i)$

It is possible to distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \quad \text{---(4)}$$

5.8.2. Remarks on *k*-Nearest Neighbor Algorithm

The distance-weighted *k*-Nearest Neighbor algorithm is a highly effective inductive inference method for many practical problems

- Robust to noisy training data
- Quite effective when it is provided a sufficiently large set of training data

What is the inductive bias of *k*-Nearest Neighbor?

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.

Practical issue!!

- The distance between instances is calculated based on *all* attributes of the instance.
- The decision is affected because of ***curse of dimensionality*** → The distance between neighbors will be dominated by the large number of irrelevant attributes

Solution → weight each attribute differently when calculating the distance between two instances.

→ 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→ use cross-validation approach.

Cross-validation approach

Consider to stretch (multiply) the jth axis by some factor z_j, where the values $z_1 \ldots z_n$ are chosen to minimize the true classification error of the learning algorithm.

This true error can be estimated using cross validation.

Therefore,

Algorithm Step 1→ select a random subset of the available data to use as training examples

Algorithm Step 2→ determine the values of $z_1 \ldots z_n$ that lead to the minimum error in classifying the remaining examples.

Repeat the above process multiple times the estimate for these weighting factors can be made more accurate.

Alternative

Completely eliminate the least relevant attributes from the instance space.

- Equivalent to setting some of the z_i scaling factors to zero.

Moore and Lee (1994)

- Leave-one-out cross validation → the set of *m* training instances is repeatedly divided into a training set of size m-1 and test set of size 1, in all possible ways.
- Can be easily applied by *knn*.

Another issue: **efficient memory indexing.**

Reason: The algorithm delays all processing until a new query is received, significant computation can be required to process each new query.

5.8.3. A Note on Terminology

Uses a terminology that has arisen from the field of statistical pattern recognition.

- *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.
  - It is a function K such that $w_i = K(d(x_i, x_q))$

5.9. LOCALLY WEIGHTED REGRESSION

Locally Weighted Regression (LWR) is the generalization of nearest-neighbor approaches.

Nearest-neighbor approaches→ approximate the target function *f(x)* at the single query point $x = x_q$.

LWR → constructs an explicit approximation to *f* over a local region surrounding x_q.

→ uses nearby or distance-weighted training examples to form the local approximation to *f* .

f can be approximated using a linear function or quadratic function or multilayer NN or any form.

LWR is

- ***LOCAL*** because nearby or distance-weighted training examples are used to form the local approximation to *f*
- ***WEIGHTED*** because the contribution of each training example is weighted by its distance from the query point
- ***REGRESSION*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

The general approach in LWR!!

Given,

x_q → new query instance

Approach→ construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q.

→ use the approximation to calculate $\hat{f}(x_q)$

Here,

$\hat{f}(x_q)$ → output as the estimated target value for the query instance.

$\hat{f}$ need to be retained as a different local approximation will be calculated for each distinct query instance

5.9.1. Locally Weighted Linear Regression

Consider,

A case of locally weighted regression in which the target function f is approximated near x_q using a linear function of the form

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

From Gradient Descent rule,

$$E \equiv \frac{1}{2}\sum_{x\in D}(f(x) - \hat{f}(x))^2 \quad \text{---(5)}$$

And

$$\Delta w_j = \eta \sum_{x\in D}(f(x) - \hat{f}(x))a_j(x)) \quad \text{---(6)}$$

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

Ans: Redefine the error criterion ***E*** to emphasize fitting the local training examples

There are **3 possible criteria:**

Let E(x$_q$) → error is being defined as a function of the query point x$_q$.

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

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

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

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

3. Combine 1 and 2

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

Criteria 3 is the best approach. If criteria 3 is used and gradient descent in Eq. (6) is re-derived, we get the training rule as follows:

$$\Delta w_j = \eta \sum\nolimits_{x \in k \; nearest \; nbrs \; of \; x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) a_j(x)) \quad \text{---(7)}$$

5.9.2. Remark son Locally Weighted Regression

In most cases, the target function is approximated by a constant, linear, or quadratic function.

More complex functional forms are not often found because

(1) the cost of fitting more complex functions for each query instance is prohibitively high, and

(2) these simple approximations model the target function quite well over a sufficiently small sub-region of the instance space

5.10. RADIAL BASIS FUNCTION

Radial Basis Function (RBF) is closely related to distance-weighted regression and also to artificial neural networks.

By Powell 1987; Broomhead and Lowe 1988; Moody and Darken 1989

The learned hypothesis is a function of the form

$$\hat{f}(x) \equiv w_0 + \sum\nolimits_{u=1}^{k} w_u K_u(d(x_u, x)) \quad \text{---(8)}$$

$K_u(d(x_u, x))$→ Gaussian function centered at the point x_u with some variance σ_μ^2.

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

Eq. 8 can represent 2-layer network where

- the first layer of units computes the values of the various $K_u\big(d(x_u, x)\big)$.
- the second layer computes a linear combination of these first-layer unit values

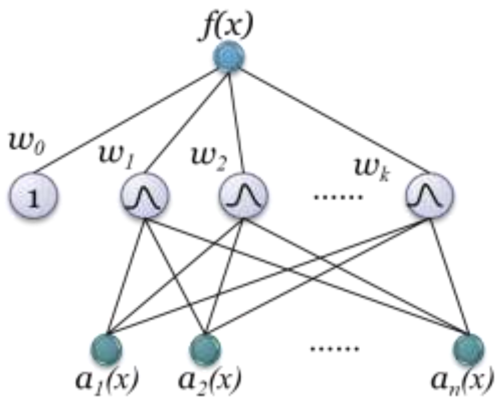


Figure 5.4: A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u.

Therefore, its activation will be close to zero unless the input x is near x_u. The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included

RBF networks are trained in 2-stage process:

1. the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_μ^2 that define its kernel function $K_u(d(x_u, x))$.
2. the weights w_u are trained to maximize the fit of the network to the training data, using the global error criterion given by Eq. 5

Alternative methods to choose an appropriate number of hidden units or, equivalently, kernel functions

- **Allocate a Gaussian kernel function for each training example $\langle x_i, f(x_i)\rangle$ centering this Gaussian at the point x_i.**

  Each kernels may be assigned the same width σ^2.

  Result: RBF network learns a global approximation to the target function in which each training example $\langle x_i, f(x_i)\rangle$ can influence the value of $\hat{f}$ only in the neighborhood of x_i.

  Advantage:

  This Kernel function allows the RBF network to fit the training data exactly.

  In other words,

  For any set of m training examples the weights w_o . ..w_m for combining the m Gaussian kernel functions can be set so that $\hat{f}(x) = f(x)$ for each training example $\langle x_i, f(x_i)\rangle$.

- **Choose a set of kernel functions that is smaller than the number of training examples.**

  This is efficient if the number of training examples is large.

  The set of kernel functions may be distributed with centers spaced uniformly throughout the instance space *X*.

  Or

  Distribute the centers non-uniformly, especially if the instances themselves are found to be distributed non-uniformly over *X*.

  Or

  Identify prototypical clusters of instances, then add a kernel function centered at each cluster.

5.11. CASE-BASED REASONING

3 key properties of kNN and LWR:

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 analyzing 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.

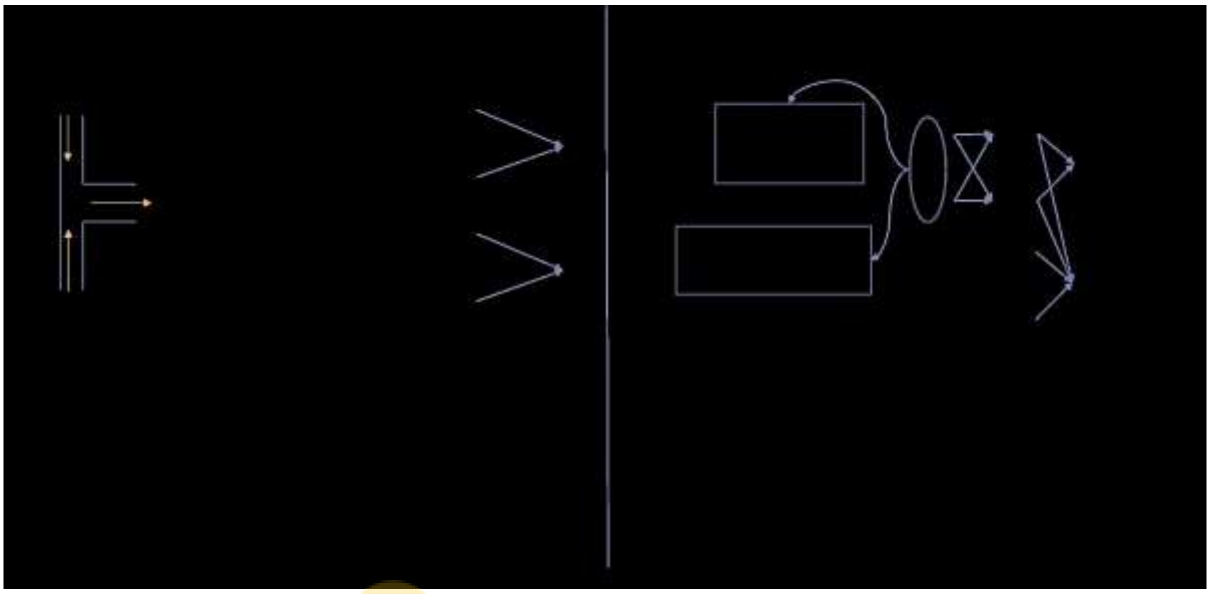
CASE-BASED REASONING

It is a learning paradigm based on the 1 and 2 but not 3.

Example,

CADET System-Sycara et al. 1992

- Uses CBR to assist in the conceptual design of simple mechanical devices such as water faucets.
- It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
- Each instance stored in memory is represented by describing both its structure and its qualitative function.
- Example, Water Pipes



Therefore,

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 sub-graphs of the desired functional specification

The system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases.

Example,

It uses a rewrite rule that allows it to rewrite the influence

$$A \rightarrow B$$

As

$$A \rightarrow x \rightarrow B$$

<u>Correspondence between the problem setting of CADET and the general setting for instance-based methods such as *k*-Nearest Neighbor.</u>

CADET

→ Each stored training example describes a function graph along with the structure that implements it.

→ New queries correspond to new function graphs.

Therefore, Consider

X→space of all function graphs

f→ Target function→ Maps function graphs to the structures that implement them

$\langle x, f(x)\rangle$→ stored training example

→ describes some function graph *x* and the structure *f (x)* that implements *x*.

Hence,

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

Properties of case-based reasoning systems that distinguish them from approaches such as *k*-Nearest Neighbor

- Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET
- Multiple retrieved cases may be combined to form the solution to the new problem → relies on knowledge-based reasoning rather than statistical methods

- There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving.

PART 3- REINFORCEMENT LEARNING

5.12. INTRODUCTION

What is addressed?

- How an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals?
- 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.

Task of the agent!!

- To learn from indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.

Example:

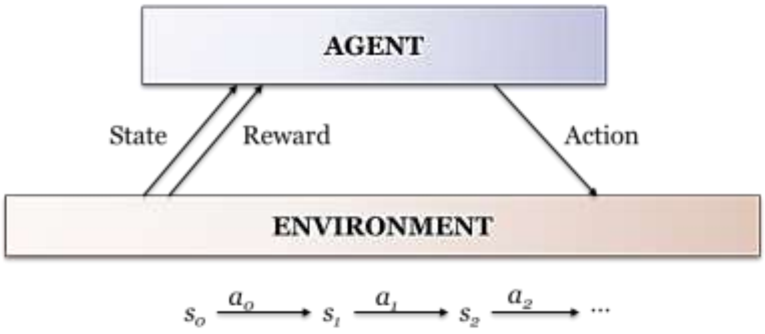
Problem: Building a Learning Robot

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.

Example: mobile robot

- Has sensors such as a camera and sonars, and actions such as “move forward” and “turn”.
- Task→ to learn a control strategy, or policy, for choosing actions that achieve its goals.
    - Example→ robot has a goal of docking onto its battery charger whenever its battery level is low.


AGENT
State
Reward
Action
ENVIRONMENT
$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots$

Goal→ Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

Where, $0 \leq \gamma \leq 1$

Task→ Learn a control policy $\pi : S \rightarrow A$ that maximizes the expected sum of rewards

Figure 5.5: 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_t, 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, π:S→*A*, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Example,

Manufacturing optimization problems in which a sequence of manufacturing actions must be chosen

- Reward → value of the goods produced minus the costs involved

Therefore, what do we need?

Type of agent that must learn to choose actions that alter the state of its environment and where a cumulative reward function is used to define the quality of any given action sequence

The problem of learning a control policy to choose actions is similar in some respects to the function approximation problems. Therefore, Target Function→ Control Policy : $\pi: S \rightarrow A$

→ outputs an appropriate action ***a*** from the set ***A,*** given the current state ***s*** from the set ***S* .**

The reinforcement learning problem differs from other function approximation tasks in several important respects

i. Delayed Reward
ii. Exploration
iii. Partially Observable States
iv. Life-Long Learning

i. <u>Delayed Reward</u>

- Task→ earn a target function π that maps from the current state *s* to the optimal action $a = \pi(s)$.
- Approach→ not as $<s, \pi(s)>$

  → trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions

- Problem→ temporal credit assignment → determining which of the actions in its sequence are to be credited with producing the eventual rewards.

ii. Exploration

- The agent influences the distribution of training examples by the action sequence it chooses.
- Tradeoff→ 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).

iii. Partially Observable States

- In many practical situations sensors provide only partial information.
- Example→ a robot with a forward-pointing camera cannot see the rear environment.
- Solution→ agent should consider its previous observations together with its current sensor data to choose action
  →actions to improve the observation of the environment

iv. Life-Long Learning

- Example,
  - Robot learning → learn several related tasks within the same environment, using the same sensors.
  - Result→ possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

5.13. LEARNING TASK

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 $s_{t+1} = \delta(s_t,a_t)$.
- The functions $\delta(s_t,a_t)$ and $r(s_t,a_t)$ depend only on the current state and action, and not on earlier states or actions.

Task of the agent → To learn a *policy*, $\pi: S \rightarrow A$, for selecting its next action a_t, based on the current observed state s_t → $\pi(s_t) = a_t$.

How shall we specify precisely which policy π we would like the agent to learn?

Require the policy that produces the greatest possible cumulative reward for the robot over time.

Therefore,

Define the cumulative value $V\pi(s_t)$ achieved by following an arbitrary policy π , from an arbitrary initial state s_t as:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ..$$

$$V^{\pi}(s_t) \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad \text{---(1)}$$

Where, $0 \leq \gamma \leq 1$

$V^{\pi}(s_t)$→ Discounted Cumulative Reward

Variants

- Finite Horizon Reward→ $\sum_{i=0}^{h} r_{t+i}$→ undiscounted sum of rewards over a finite number h of steps.
- Average Reward → $\lim_{h\rightarrow\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$→ the average reward per time step over the entire lifetime of the agent.

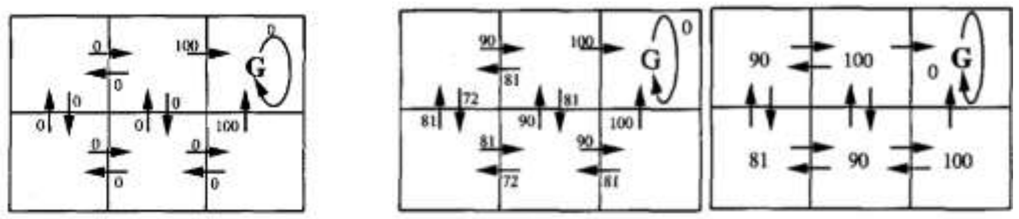
Learning Task→ learn a policy π that maximizes $V^{\pi}(s)$for all states s→ Optimal **Policy** → π^*

$$\pi^* = \underset{\pi}{argmax} V^{\pi}(s), (\forall s) \quad \text{--(2)}$$

Let, $V^{\pi^*}(s)$ be $V^*(s)$ → maximum discounted cumulative reward that the agent can obtain starting from state *s*.

Illustration

To illustrate these concepts, a simple grid-world environment is depicted in the topmost diagram of Figure 5.6.



$r(s,a)$ (immediate reward) values $\qquad Q(s,a)$ values $\qquad V^*(s)$ values

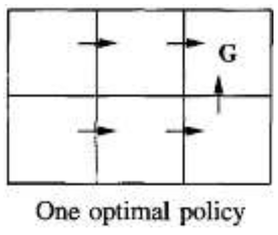


Figure 5.6: A simple deterministic world to illustrate the basic concepts of *Q*-learning.

- Each grid square represents a distinct state, each arrow a distinct action.
- An **optimal policy**, corresponding to actions with maximal Q values, is also shown
  - specifies exactly one action that the agent will select in any given state.
  - Directs the agent along the shortest path toward the state G
- The number associated with each arrow represents the immediate reward r(s,a) the agent receives if it executes the corresponding state-action transition.
- Here G→ Goal State→ agent can receive reward by entering this state.
- Also, G→ Absorbing State → the only action available to the agent once it enters the state G is to remain in this state.
- Example, consider that, an 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 γ = 0.9.
- In bottom right grid, *V*=100* because the optimal policy in this state selects the "move up" action that receives immediate reward 100.
- V* in bottom center state is 90 since the optimal policy will move the agent from this state to the right (generating an immediate reward of zero), then upward (generating an immediate reward of 100).

  Therefore, the discounted future reward from the bottom center state is

$$0 + \gamma^1 100 + \gamma^2 0 + \gamma^3 0 + \cdots = 90$$

5.14. *Q* LEARNING

How can an agent learn an optimal policy π * for an arbitrary environment?

The 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.

What evaluation function should the agent attempt to learn?

One obvious choice is V*. The agent should prefer state s_l over state s_2 whenever V*(s_l) > V*(s_2), because the cumulative future reward will be greater from s_l.

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 by **γ.**

$$\pi^*(s) = \underset{a}{argmax}[r(s,a) + V^*(\delta(s,a))] \qquad \text{---(3)}$$

5.14.1. The *Q* Function

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by γ) of following the optimal policy thereafter

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \text{---(4)}$$

Q(s,a)→ quantity that is maximized in Eq. 3 to choose the optimal action *a* in state *s*

Re-write Eq. (3) in terms of *Q(s,a)* to get,

$$\pi^*(s) = \underset{a}{argmax}\, Q(s,a) \qquad \text{---(5)}$$

That is, it needs to only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.

Why is this rewrite important?

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 δ* .

Figure 5.6 shows *Q* values for every state and action in the simple grid world.

- *Q* value for each state-action transition equals the *r* value for this transition plus the *V** value for the resulting state discounted by γ.
- Optimal policy shown in the figure corresponds to selecting actions with maximal *Q* values

5.14.2. An Algorithm for Learning *Q*

- Learning the Q function corresponds to learning the **optimal policy**.
- 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***

$$V^*(s) = max_{a'}^{Q(s,a')}$$

Re-write the above equation to get

$$Q(s,a) = r(s,a) + \gamma max_{a'}^{Q(\delta(s,a),a')} \text{---(6)}$$

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' = δ(s,a)*. It then updates the table entry for $\hat{Q}(s,a)$ following each such transition, according to the rule:

$$\hat{Q}(s,a) = r + \gamma max_{a'}^{\hat{Q}(s',a')} \text{---(7)}$$

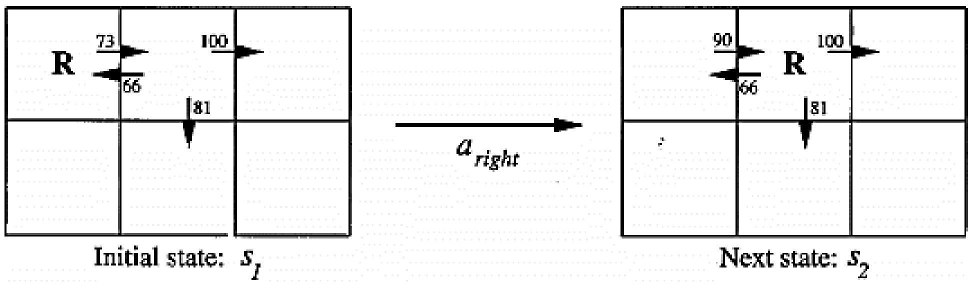
<u>Q Learning Algorithm</u>

```
For each s,a initialize the table entry $\hat{Q}(s,a)$ to zero.
Observe the current state s
Do forever:
    • Select an action a and execute it
    • Receive immediate reward r
    • Observe the new sate s′
    • Update the table entry for $\hat{Q}(s,a)$ as follows:
          $\hat{Q}(s,a) = r + \gamma max_{a'}^{\hat{Q}(s',a')}$
    • $s \leftarrow s'$
```

- Q learning algorithm assuming deterministic rewards and actions. The discount factor γ **may be any constant such that 0 ≤ γ < 1**
- $\hat{Q}$ to refer to the learner's estimate, or hypothesis, of the actual Q function

5.14.3. An Illustrative Example

To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to $\hat{Q}$ shown in below figure:



- The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition.
- Apply the training rule of Equation

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

to refine its estimate Q for the state-action transition it just executed. According the training rule, the new $\hat{Q}$ estimate for this transition is the sum of the received reward (zero) and the highest $\hat{Q}$ value associated with the resulting state (100), discounted by **γ(.9).**

$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \ \max\{66, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

5.14.4. Convergence

Will the Q Learning Algorithm converge toward a Q equal to the true Q function?

Yes, under certain conditions.

i. Assume the system is a deterministic MDP.

ii. Assume the immediate reward values are bounded; that is, there exists some positive constant c such that for all states s and actions a, **| r(s, a)| < c**

iii. Assume the agent selects actions in such a fashion that it visits every possible state-action pair infinitely often

Theorem 5.1: Convergence of *Q* learning for deterministic Markov decision processes.

Consider a *Q* learning agent in a deterministic MDP with bounded rewards$(\forall s, a|r(s,a)| \leq c$. The *Q* learning agent uses the training rule of Eq. 7 initializes its table $\hat{Q}(s,a)$ to arbitrary finite values, and uses a discount factor γ such that **0** $\leq \gamma <$ 1. Let $\hat{Q}_n(s,a)$ denote the agent's hypothesis $\hat{Q}(s,a)$ following the nth update. If each state-action pair is visited infinitely often, then $\hat{Q}_n(s,a)$ converges to *Q(s,a)* as $n \rightarrow \infty$, for all *s, a* .

Prooof: Since each state-action transition occurs infinitely often, consider consecutive intervals during which each state-action transition occurs at least once.

To prove that: The maximum error over all entries in the $\hat{Q}$ table is reduced by at least a factor of γ during each such interval. $\hat{Q}_n$**,** is the agent's table of estimated $\hat{Q}_n$ values after n updates.

Let Δ_n be the maximum error in $\hat{Q}_n$; That is,

$$\Delta_n \equiv max_{s,a}^{|\hat{Q}_n(s,a)-Q(s,a\,)|}$$

Now use $\boldsymbol{s'}$ to denote δ*(s,a)*. Hence for any table entry $\hat{Q}_n(s,a)$ that is updated on iteration *n+1*, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s,a)$ is

$$\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| = |\left(r + \gamma max_{a'}^{\hat{Q}_n\left(s',a'\right)}\right) - (r + \gamma max_{a'}^{Q(s',a')})|$$

$$= \gamma |max_{a'}^{\hat{Q}_n\left(s',a'\right)} - max_{a'}^{Q(s',a')}| \quad \text{---(ii)}$$

$$\leq \gamma max_{a'}^{|\hat{Q}_n(s',a')-Q(s',a')|} \quad \text{---(iii)}$$

$$\leq \gamma max_{s'',a'}^{|\hat{Q}_n(s'',a')-Q(s'',a')|} \quad \text{---(iv)}$$

$$\left|\hat{Q}_{n+1}(s,a) - Q(s,a)\right| \leq \gamma \Delta_n$$

Expression (iii) follows from Expression (ii) because for any two functions f_1 and f_2 the following inequality holds

$$|max_a^{f_1(a)} - max_a^{f_2(a)}| \leq max_a^{|f_1(a)-f_2(a)|}$$

s'' is introduced in Expression (iv) over which the maximization is performed. It allowed to obtain an expression that matches the definition of Δ_n**.**

The updated $\hat{Q}_{n+1}(s,a)$ for any *s*, *a* is at most γ times the maximum error in the $\hat{Q}_n$ table, Δ_n**.** The largest error in the initial table, Δ_0, is bounded because values of $\hat{Q}_0(s,a)$ and *Q(s,a)* are bounded for all *s*, *a*.

- After 1st interval during which each **s**,a is visited, the largest error in the table will be at most $\gamma\Delta_0$.
- After k such intervals, the error will be at most $\gamma^k\Delta_0$
- Since each state is visited infinitely often, the number of such intervals is infinite, and $\Delta_n \rightarrow 0$ as $n \rightarrow \infty$.

Hence proved.

5.14.5. Experimentation Strategies

The Q learning algorithm does not specify how actions are chosen by the agent

- One obvious strategy would be for the agent in state ***s*** to select the action ***a*** that maximizes $\hat{Q}(s,a)$, thereby exploiting its current approximation $\hat{Q}$.
- The agent runs the risk that it will overcommit to actions that are found during early training to have high $\hat{Q}$ values, while failing to explore other actions that have even higher values.

- For this reason, Q learning uses a probabilistic approach to selecting actions. Actions with higher $\hat{Q}$ values are assigned higher probabilities, but every action is assigned a nonzero probability.
- One way to assign such probabilities is

$$\boldsymbol{P(a_i|s)} = \frac{\boldsymbol{k}^{\hat{Q}(s,a_i)}}{\sum_{\boldsymbol{j}} \boldsymbol{k}^{\hat{Q}(s,a_j)}}$$

Where,

P(a_i|*s*) → probability of selecting action a_i, given that the agent is in state s

k>0 → constant that determines how strongly the selection favors actions with high $\hat{Q}$ values.