Course: Machine Learning (17CS73)

Module 5

Evaluating Hypothesis, Classification using instance-based learning, Reinforcement Learning

Prof. Mahesh G. Huddar

Instance-based Learning

- **Key idea**: In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance-based learning constructs the target function only when 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 Learning

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

Instance-based Learning

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

Advantages of Instance-based learning

1. Training is very fast
2. Learn complex target function
3. Don’t lose information

Disadvantages of Instance-based learning

- The 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.
- In many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

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^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- The arbitrary instance ***x*** be described by the feature vector

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

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

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

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

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

k-NEAREST NEIGHBOR LEARNING

- Let us first consider learning ***discrete-valued target functions*** of the form

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

- Where, V is the finite set $\{v_1, \ldots v_s\}$
- The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:

Training algorithm:

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

Classification algorithm:

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

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

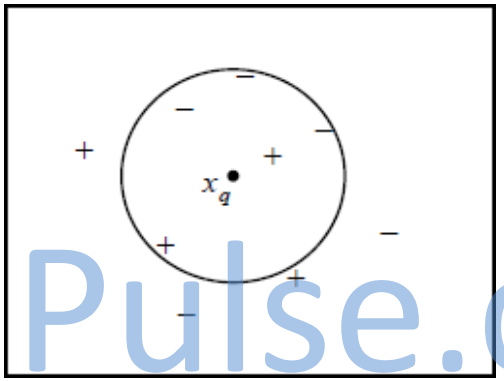
where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

k-NEAREST NEIGHBOR LEARNING

- The value $\hat{f}(xq)$ returned by this algorithm as its estimate of $f(xq)$ is just the most common value of f among the k training examples nearest to xq.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(xq)$ the value $f(x_i)$.
- Where xi is the training instance nearest to xq.
- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

k-NEAREST NEIGHBOR LEARNING

- Below figure illustrates the operation of the k-Nearest Neighbor 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.
- The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.

k-NEAREST NEIGHBOR LEARNING

- The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given below $f : \Re^n \rightarrow \Re$

Training algorithm:

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

Classification algorithm:

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

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

Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point xq, giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from xq.
- Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions

Distance-Weighted Nearest Neighbor Algorithm

Training algorithm:

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

Classification algorithm:

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

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

where

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

Distance-Weighted Nearest Neighbor Algorithm

- Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

---

Training algorithm:

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

Classification algorithm:

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

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

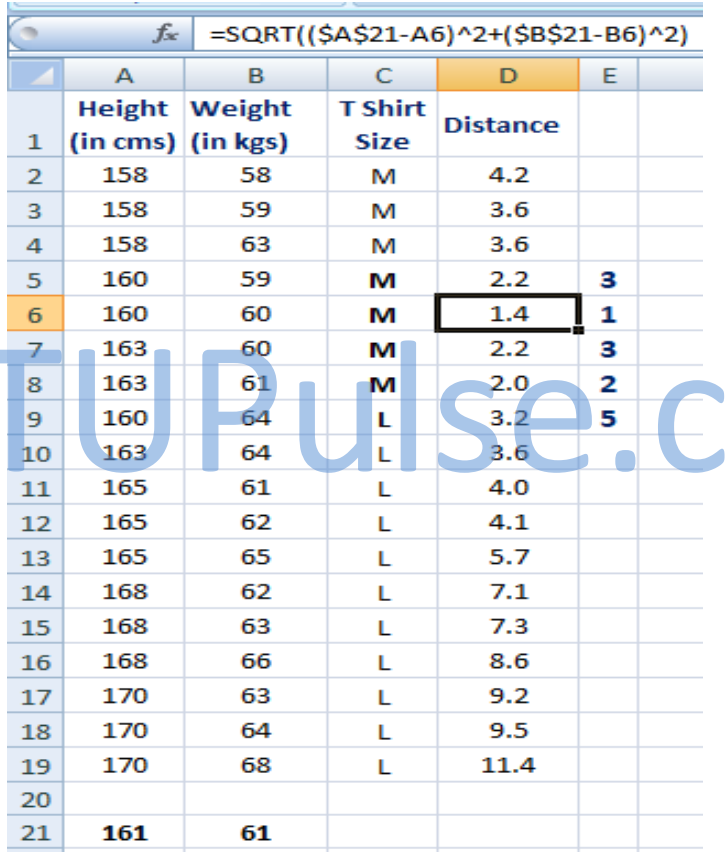
where

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

Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have. Data including height, weight and T-shirt size information is shown below

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	L
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

New customer named 'Monica' has height 161cm and weight 61kg.



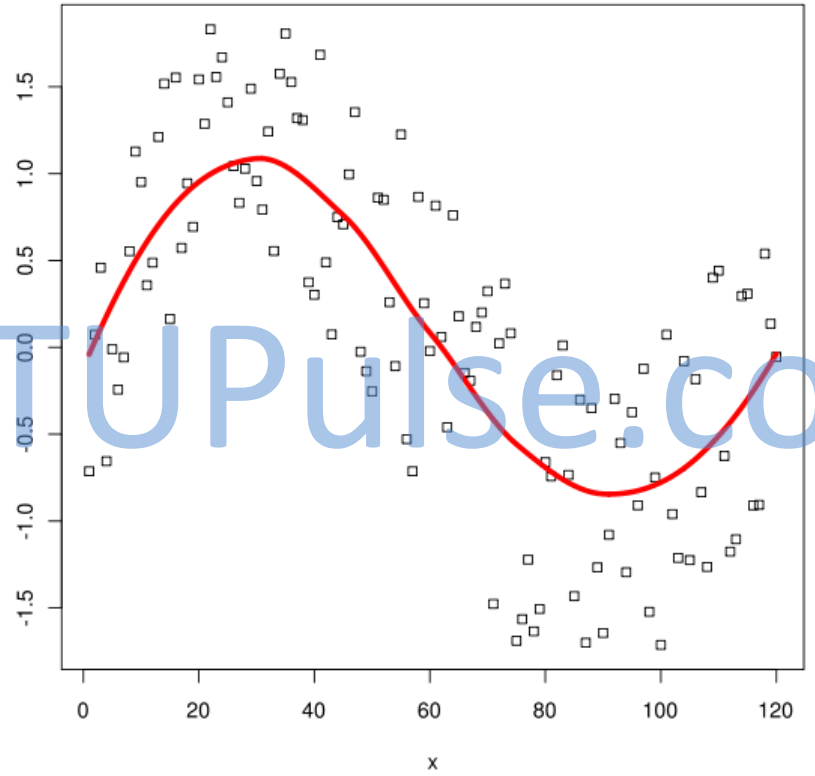
=SQRT((A21-A6)^2+(B21-B6)^2)

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	**M**	2.2	**3**
6	160	60	**M**	1.4	**1**
7	163	60	**M**	2.2	**3**
8	163	61	**M**	2.0	**2**
9	160	64	**L**	3.2	**5**
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	**161**	**61**			

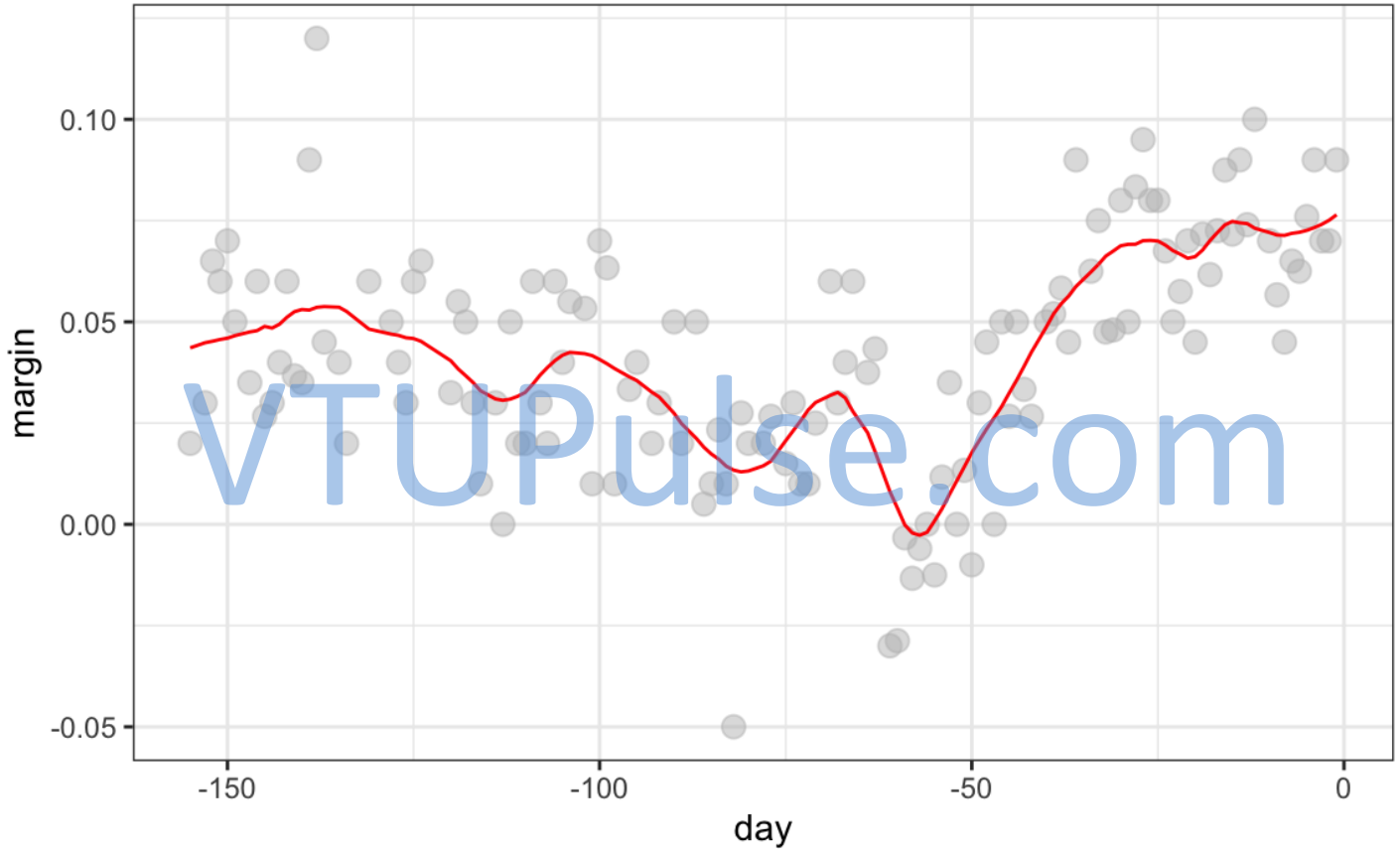
LOCALLY WEIGHTED REGRESSION

- The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.
- Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding xq. This approximation is then used to calculate the value $\hat{f}(xq)$, which is output as the estimated target value for the query instance.

LOCALLY WEIGHTED REGRESSION



LOCALLY WEIGHTED REGRESSION



LOCALLY WEIGHTED REGRESSION

- Consider locally weighted regression in which the target function ***f*** is approximated near xq using a linear function of the form

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

- Where, ai(x) denotes the value of the i^{th} attribute of the instance x

$$w_i \leftarrow w_i + \Delta w_i$$

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

LOCALLY WEIGHTED REGRESSION

- Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

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

- Which led us to the gradient descent training rule

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

- Where, η is a constant learning rate

LOCALLY WEIGHTED REGRESSION

Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

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

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

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 xq :

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

LOCALLY WEIGHTED REGRESSION

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \, K(d(x_q, x)) \qquad \text{equ(3)}$$

- If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

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

- The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples.

RADIAL BASIS FUNCTIONS

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

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

- Where, each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases.
- Here k is a user provided constant that specifies the number of kernel functions to be included.
- $\hat{f}$ is a global approximation to f (x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

RADIAL BASIS FUNCTIONS

- Choose each function Ku(d(x_u, x)) to be a Gaussian function centred at the point x_u with some variance σ_u^2

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

- The functional form of equ(1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width
- σ^2 of each kernel can be separately specified
- The function given by equ(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

Example: Radial basis function (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, are trained to maximize the fit of the network to the training data, using the global error criterion given by

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

- Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently

REINFORCEMENT LEARNING

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.

- Consider 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.
- Its task is to learn a control strategy, or ***policy***, for choosing actions that achieve its goals.
- The goals of the agent can be defined by a ***reward function*** that assigns a numerical value to each distinct action the agent may take from each distinct state.

REINFORCEMENT LEARNING

- 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 is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

REINFORCEMENT LEARNING

Example:

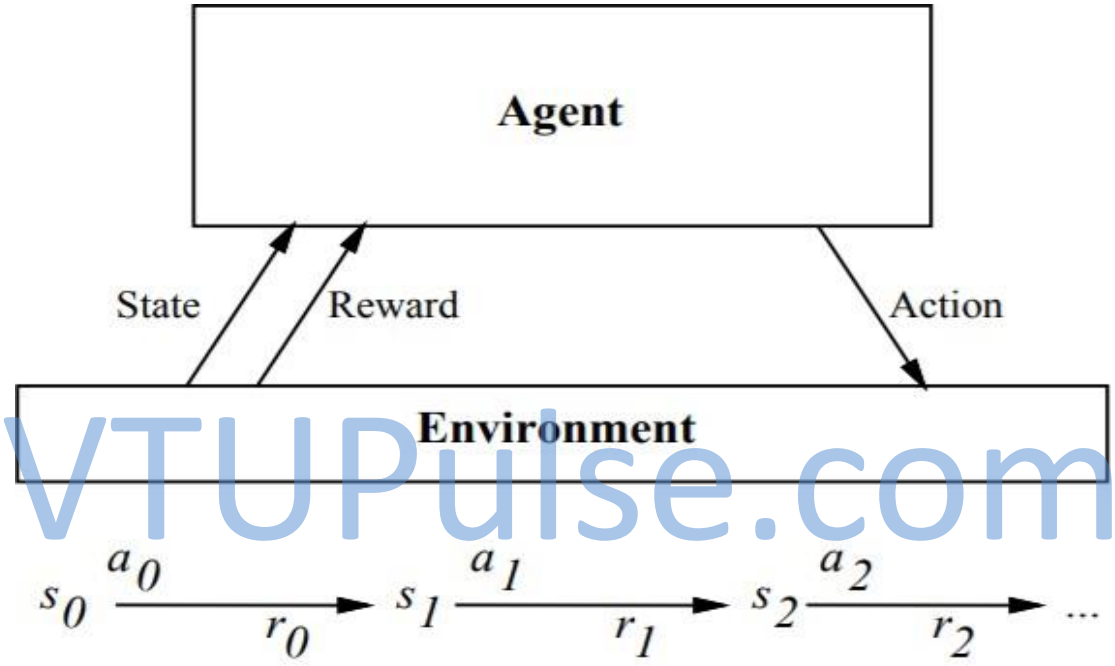
- A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn."
- The robot may have a goal of docking onto its battery charger whenever its battery level is low.
- The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

REINFORCEMENT LEARNING

Reinforcement Learning Problem

- An agent interacting with its environment.
- The agent exists in an environment described by some set of possible states S.
- Agent perform any of a set of possible actions A.
- Each time it performs an action A, in some state st the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition.
- This produces a sequence of states si, actions ai, and immediate rewards ri 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.

REINFORCEMENT LEARNING



Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \text{ , where } 0 \leq \gamma < 1$$

Reinforcement learning problem characteristics

1. **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 = π (s). In reinforcement learning, training information is not available in (s, π (s)). 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.

2. **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 trade-off in choosing whether to favor exploration of unknown states and actions, or exploitation of states and actions that it has already learned will yield high reward.

Reinforcement learning problem characteristics

3. **Partially observable states:** 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. In such cases, the agent needs 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.
4. **Life-long learning:** Robot requires to 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.

Reinforcement learning

THE LEARNING TASK

- Consider Markov decision process (MDP) where 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 st, chooses a current action at, and performs it.
- The environment responds by giving the agent a reward rt = r(st, at) and by producing the succeeding state st+1 = δ(st, at).
- Here the functions δ(st, at) and r(st, at) depend only on the current state and action, and not on earlier states or actions.
- The task of the agent is to learn a policy, π**: S → A**, for selecting its next action a, based on the current observed state st; that is, π**(st) = at**.

Reinforcement learning

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

1. One approach is to require the policy that produces the greatest possible ***cumulative reward*** for the robot over time.

   – To state this requirement more precisely, define the cumulative value V^{π} (st) achieved by following an arbitrary policy π from an arbitrary initial state st as follows:

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

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

Reinforcement learning

- Where, the sequence of rewards rt+i is generated by beginning at state st and by repeatedly using the policy π to select actions.
- Here $0 \leq \gamma \leq 1$ is a constant that determines the relative value of delayed versus immediate rewards. 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^{π} (st) is called the ***discounted cumulative reward*** achieved by policy π from initial state ***s***. It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

Reinforcement learning

2. Other definitions of total reward is ***finite horizon reward,***

$$\sum_{i=0}^{h} r_{t+i}$$

Considers the undiscounted sum of rewards over a finite number ***h*** of steps

3. Another approach is ***average reward***

$$\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

Considers the average reward per time step over the entire lifetime of the agent.

Reinforcement learning

Q learning algorithm

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 state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

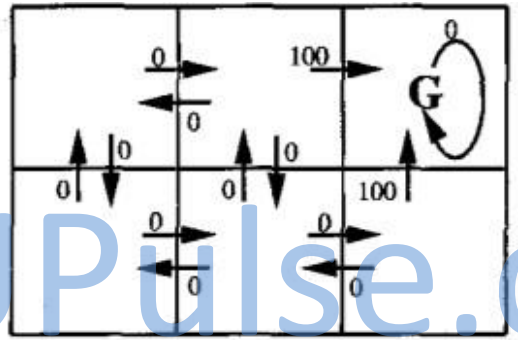
$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

Reinforcement learning

Example:

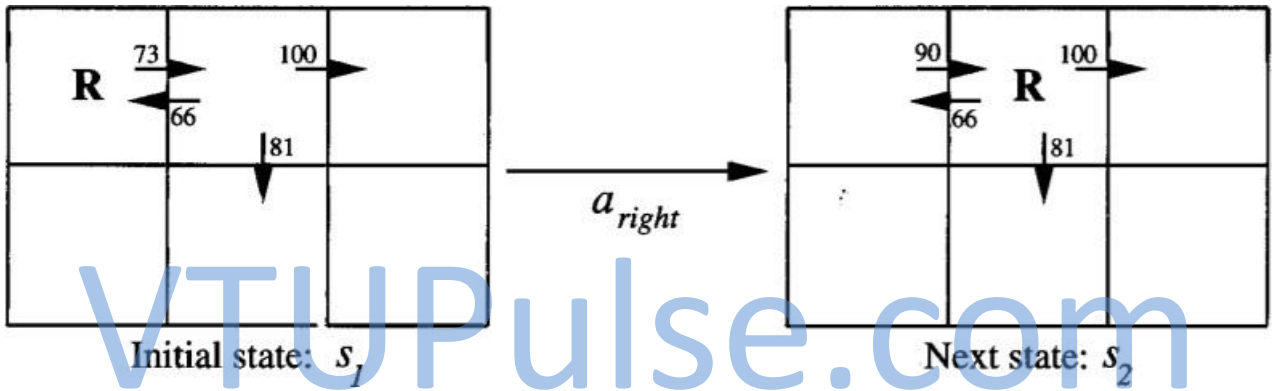
- A simple grid-world environment is depicted in the diagram



$r(s, a)$ (immediate reward) values

Reinforcement learning

- To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to 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.

Reinforcement learning

- Apply the training rule of Equation

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

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

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

EVALUATING HYPOTHESES

MOTIVATION

- It is important to evaluate the performance of learned hypotheses as precisely as possible.
  - One reason is simply to understand whether to use the hypothesis.
  - A second reason is that evaluating hypotheses is an integral component of many learning methods.

EVALUATING HYPOTHESES

Two key difficulties arise while learning a hypothesis and estimating its future accuracy given only a limited set of data:

- **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. Because the learned hypothesis was derived from these examples, they will typically provide an optimistically biased estimate of hypothesis accuracy over future examples. This is especially likely when the learner considers a very rich hypothesis space, enabling it to overfit the training examples. To obtain an unbiased estimate of future accuracy, test the hypothesis on some set of test examples chosen independently of the training examples and the hypothesis.
- **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.

EVALUATING HYPOTHESES

ESTIMATING HYPOTHESIS ACCURACY

- **Sample Error –**
- 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 (***$error_S(h)$***) 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 is the number of examples in S, and the quantity δ(f(x), h(x)) is 1 if f (x) ≠ h(x), and 0 otherwise.

EVALUATING HYPOTHESES

ESTIMATING HYPOTHESIS ACCURACY

- **True Error –**
- The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution **D**.
- ***Definition:*** The true error ($error_D(h)$) 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**

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

EVALUATING HYPOTHESES

- **Confidence Intervals for Discrete-Valued Hypotheses**
- Suppose we wish to estimate the true error for some discrete valued hypothesis h, based on its observed sample error over a sample S, where
  - The sample S contains n examples drawn independent of one another, and independent of h, according to the probability distribution D
  - $n \geq 30$
  - Hypothesis h commits r errors over these n examples (i.e., $error_s(h) = r/n$).

EVALUATING HYPOTHESES

Under these conditions, statistical theory allows to make the 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 $error_D$ (h) lies in the interval

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

Example:

- Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.
  - The ***sample error*** is $error_s$(h) = r/n = 12/40 = 0.30
  - Given no other information, ***true error*** is $error_D$ (h) = $error_s$(h), i.e., $error_D$ (h) = 0.30
  - With the 95% confidence interval estimate for $error_D$ (h).

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

= 0.30 ± (1.96 * 0.07) = 0.30 ± 0.14

EVALUATING HYPOTHESES

3. A different constant, ***ZN***, is used to calculate the **N% confidence interval**. The general expression for approximate N% confidence intervals for $error_D$ (h) is

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

- Where,

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

- The above equation describes how to calculate the confidence intervals, or error bars, for estimates of $error_D$ (h) that are based on $error_s$(h)

EVALUATING HYPOTHESES

- **Example:**
- Suppose the data sample S contains n = 40 examples and that hypothesis h commits r = 12 errors over this data.
  - The ***sample error*** is errors(h) = r/n = 12/40 = 0.30
  - With the 68% confidence interval estimate for $error_D(h)$.

$$error_S(h) \pm 1.00\sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

= 0.30 ± (1.00 * 0.07)

= 0.30 ± 0.07

EVALUATING HYPOTHESES

The Binomial Distribution

Consider the following problem for better understanding of Binomial Distribution

- Given a worn and bent coin and estimate the probability that the coin will turn up heads when tossed.
- Unknown probability of heads ***p***. Toss the coin ***n*** times and record the number of times ***r*** that it turns up heads.

Estimate of **p = r / n**

- If the experiment were *rerun*, generating a new set of ***n*** coin tosses, we might expect the number of heads ***r*** to vary somewhat from the value measured in the first experiment, yielding a somewhat different estimate for ***p***.
- The Binomial distribution describes 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***.

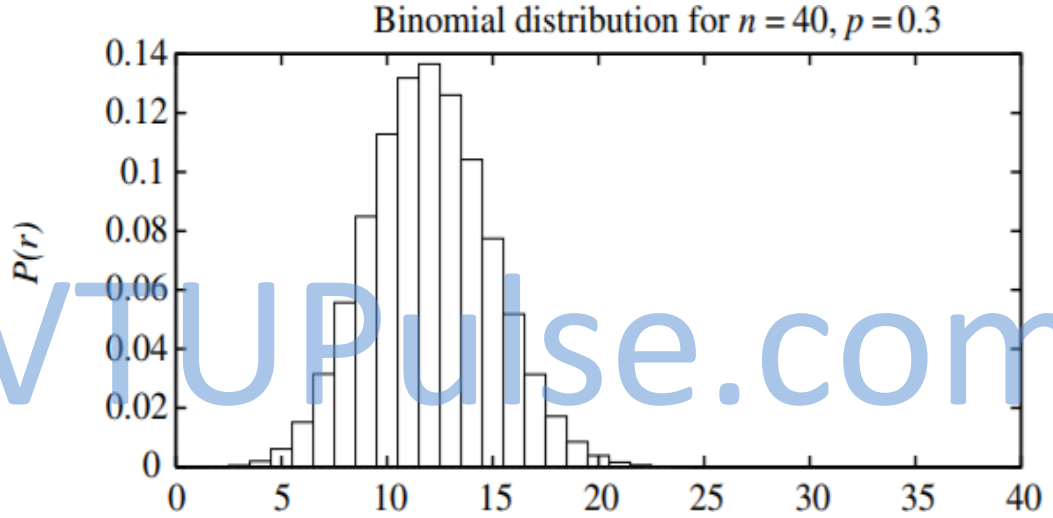
EVALUATING HYPOTHESES

BASICS OF SAMPLING THEORY

- **Error Estimation and Estimating Binomial Proportions**
- Collect a random sample S of *n* independently drawn instances from the distribution *D*, and then measure the sample error $error_s(h)$. Repeat this experiment many times, each time drawing a different random sample Si of size *n*, we would expect to observe different values for the various $error_{si}(h)$, depending on random differences in the makeup of the various Si. We say that $error_{si}(h)$, the outcome of the i^{th} such experiment, is a ***random variable***.
- Imagine that we were to run k random experiments, measuring the random variables $error_{s1}(h)$, $error_{s2}(h)$ . . . $errors_{sk}(h)$ and plotted a histogram displaying the frequency with which each possible error value is observed.

EVALUATING HYPOTHESES

- As ***k*** grows, the histogram would approach a particular probability distribution called the ***Binomial distribution*** which is shown in below figure.



- A Binomial distribution is defined by the probability function

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

EVALUATING HYPOTHESES

- 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)***

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

$$E[X] \equiv \sum_{i=0}^{n} iP(i) = np$$

- Variance of X is

$$Var(X) \equiv E[(X - E[X])^2] = np(1-p)$$

- Standard deviation of X, σ_X, is

$$\sigma_X \equiv \sqrt{E[(X - E[X])^2]} = \sqrt{np(1-p)}$$

EVALUATING HYPOTHESES

Estimators, Bias, and Variance

- Let us describe errors(h) and errorD(h) using the terms in the Binomial distribution. We then have

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

- Where,
  - n is the number of instances in the sample S,
  - r is the number of instances from S misclassified by h
  - p is the probability of misclassifying a single instance drawn from *D*

EVALUATING HYPOTHESES

- The variance in this estimate arises completely from the variance in *r,* because *n* is a constant.
- Because *r* is Binomially distributed, its variance is given by as *np(1 - p).*
- Given *r* errors in a sample of *n* independently drawn test examples, the standard deviation for *errors(h)* is given by

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

- which can be approximated by substituting *rln* = *errors(h)* for *p*

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

EVALUATING HYPOTHESES

Confidence Intervals

- An ***N%* confidence interval** for some parameter p is an interval that is expected with probability ***N%*** to contain ***p.***
- For example, if we observe $r = 12$ errors in a sample of $n = 40$ independently drawn examples, we can say with approximately *95%* probability that the interval 0.30 ± 0.14 contains the true error $error_D(h)$.

How can we derive confidence intervals for $error_D(h)$?

- The answer lies in the fact that we know the Binomial probability distribution governing the estimator $error_s(h)$.
- The mean value of this distribution is $error_D(h)$, and the standard deviation.
- Therefore, to derive a *95%* confidence interval, we need only find the interval centered around the mean value $error_D(h)$, which is wide enough to contain ***95%*** of the total probability under this distribution.

EVALUATING HYPOTHESES

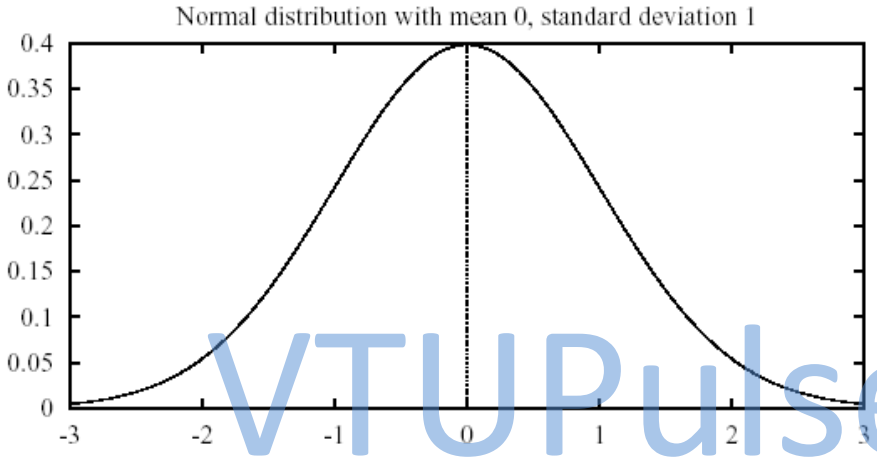
Confidence Intervals

- This provides an interval surrounding $error_D(h)$ into which $error_S(h)$ must fall ***95%*** of the time. Equivalently, it provides the size of the interval surrounding $error_S(h)$ into which $error_D(h)$ must fall ***95%*** of the time.

- For a given value of ***N*** how can we find the size of the interval that contains
- ***N%*** of the probability mass?
- Unfortunately, for the Binomial distribution this calculation can be quite tedious.
- Fortunately, however, an easily calculated and very good approximation can be found in most cases, based on the fact that for sufficiently large sample sizes the Binomial distribution can be closely approximated by the Normal distribution.

EVALUATING HYPOTHESES

Normal Probability Distribution



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

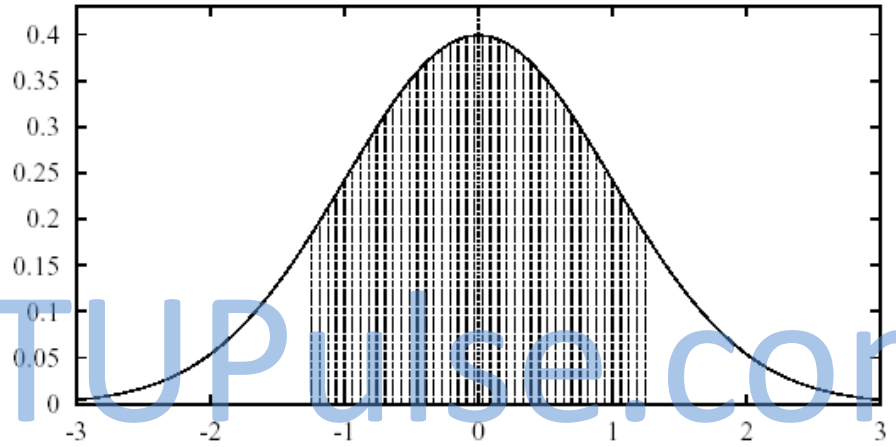
The probability that X will fall into the interval (a, b) is given by

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

- Expected, or mean value of X, $E[X]$, is $E[X] = \mu$
- Variance of X is $Var(X) = \sigma^2$
- Standard deviation of X, σ_X is $\sigma_X = \sigma$

EVALUATING HYPOTHESES

Normal Probability Distribution



Probability that mean μ falls in N% of area is $\mu \pm z_N\sigma$

N%	50%	68%	80%	90%	95%	98%	99%
z_N	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Confidence Intervals

- First, we know that *errors(h)* follows a Binomial distribution.
- Second, we know that for sufficiently large sample size n, this Binomial distribution is well approximated by a Normal distribution.
- Third, Equation $\mu \pm z_N \sigma$ tells us how to find the ***N%*** confidence interval for estimating the mean value of a Normal distribution

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

Central Limit Theorem

- Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$, all governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean,

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

- **Central Limit Theorem.** As n $\rightarrow \infty$, the distribution governing $\bar{Y}$ approaches a Normal distribution, with mean μ and variance σ^2 / n .

Calculating Confidence Intervals

1. Pick parameter p to be estimated for example
   - $error_D(h)$
2. Choose an estimator
   - $error_S(h)$
3. Determine probability distribution that governs estimator
   - $error_S(h)$ governed by Binomial distribution, approximated by Normal when $n \geq 30$
   - Find mean and standard deviation
4. Find interval (L, U) such that N% of the mass in the probability falls in the interval
   - Use table of z_N values

Difference Between Hypotheses

Test h_1 on sample S_1, test h_2 on S_2

1. Pick parameter to estimate

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

2. Choose an estimator

$$\hat{d} \equiv error_{S1}(h_1) - error_{S2}(h_2)$$

3. Determine probability distribution that governs estimator

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

4. Find interval (L, U) such that N% of probability mass falls in the interval

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

Paired *t* test to compare h_A, h_B

1. Partition data D into *k* disjoint test sets T_1, T_2, . . ., T_k of equal size, where size is at least 30.

2. For *i* from 1 to *k*, do

$$\delta_i \leftarrow error_{Ti}(h_A) - error_{Ti}(h_B)$$

3. Return the value $\bar{\delta}$, where

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

4. *N*% confidence interval estimate for *d*:

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

Paired *t* test to compare h_A, h_B

- The approximate ***N%*** confidence interval for estimating the quantity in Equation,

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

where $t_{N,k-1}$ is a constant that plays a role analogous to that of z_N in our earlier confidence interval expressions, and where $s_{\bar{\delta}}$ is an estimate of the standard deviation of the distribution governing $\bar{\delta}$. In particular, $s_{\bar{\delta}}$ is defined as

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

$t_{N,k-1}$

- *N: desired confidence level*
- *K-1: degrees of freedom*

Comparing learning algorithms L_A and L_B

1. Partition data D_0 into k disjoint test sets $T_1, T_2, \ldots, T_k$ 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

   - $S_i \leftarrow \{ D_0 - T_i \}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{Ti}(h_A) - error_{Ti}(h_B)$
3. Return the value $\bar{\delta}$, where $\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$
4. N% confidence interval estimate for d:

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