S. J. P. N. TRUST'S

# HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI

Accredited at 'A' Grade by NAAC

Programmes Accredited by NBA: CSE, ECE, EEE & ME.

## Department of Computer Science & Engineering

**Course: Artificial Intelligence and Machine Learning (18CS71)**



## Module 5

# Classification using instance-based learning, Reinforcement Learning

Dr. Mahesh G. Huddar

Asst. Prof. , Dept. of Computer Science & Engg.,

Hirasugar Institute of Technology, Nidasoshi

# 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

| Sl. No. | Height | Weight | Target |
|---|---|---|---|
| 1 | 150 | 50 | Medium |
| 2 | 155 | 55 | Medium |
| 3 | 160 | 60 | Large |
| 4 | 161 | 59 | Large |
| 5 | 158 | 65 | Large |
| 6 | 157 | 54 | ? |

# k-NEAREST NEIGHBOR LEARNING

| Sl. No. | Height | Weight | Target | Distance |
|---|---|---|---|---|
| 1 | 150 | 50 | Medium | 8.06 |
| 2 | 155 | 55 | Medium | 2.24 |
| 3 | 160 | 60 | Large | 6.71 |
| 4 | 161 | 59 | Large | 6.40 |
| 5 | 158 | 65 | Large | 11.05 |
| 6 | 157 | 54 | ? | |

# k-NEAREST NEIGHBOR LEARNING

| Sl. No. | Height | Weight | Target | Distance | Nearest Points |
|---|---|---|---|---|---|
| 1 | 150 | 50 | Medium | 8.06 | |
| 2 | 155 | 55 | Medium | 2.24 | 1 |
| 3 | 160 | 60 | Large | 6.71 | 3 |
| 4 | 161 | 59 | Large | 6.40 | 2 |
| 5 | 158 | 65 | Large | 11.05 | |
| 6 | 157 | 54 | ? | | |

# k-NEAREST NEIGHBOR LEARNING

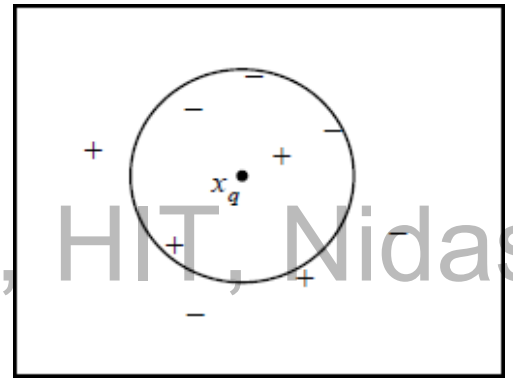
- The value $\hat{f}(xq)$ returned by this algorithm as its estimate of $f(xq)$ is just the most common value of $f$ among the $k$ training examples nearest to $xq$.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to $\hat{f}(xq)$ the value $f(x_i)$.



- 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.

# 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

| Sl. No. | Height | Weight | Target |
|---|---|---|---|
| 1 | 150 | 50 | Medium |
| 2 | 155 | 55 | Medium |
| 3 | 160 | 60 | Large |
| 4 | 161 | 59 | Large |
| 5 | 158 | 65 | Large |
| 6 | 157 | 54 | ? |

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance |
|---|---|---|---|---|
| 1 | 150 | 50 | Medium | 8.06 |
| 2 | 155 | 55 | Medium | 2.24 |
| 3 | 160 | 60 | Large | 6.71 |
| 4 | 161 | 59 | Large | 6.40 |
| 5 | 158 | 65 | Large | 11.05 |
| 6 | 157 | 54 | ? | |

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance | Nearest Points |
|---|---|---|---|---|---|
| 1 | 150 | 50 | Medium | 8.06 | |
| 2 | 155 | 55 | Medium | 2.24 | 1 |
| 3 | 160 | 60 | Large | 6.71 | 3 |
| 4 | 161 | 59 | Large | 6.40 | 2 |
| 5 | 158 | 65 | Large | 11.05 | |
| 6 | 157 | 54 | ? | | |



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

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance | 1/distance$^2$ | Nearest Points |
|---|---|---|---|---|---|---|
| 1 | 150 | 50 | Medium | 8.06 | | |
| 2 | 155 | 55 | Medium | 2.24 | 0.45 | 1 |
| 3 | 160 | 60 | Large | 6.71 | 0.15 | 3 |
| 4 | 161 | 59 | Large | 6.40 | 0.16 | 2 |
| 5 | 158 | 65 | Large | 11.05 | | |
| 6 | 157 | 54 | ? | | | |

# 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

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

# k-NEAREST NEIGHBOR LEARNING

| Sl. No. | Height | Weight | Target |
|---|---|---|---|
| 1 | 150 | 50 | 1.5 |
| 2 | 155 | 55 | 1.2 |
| 3 | 160 | 60 | 1.8 |
| 4 | 161 | 59 | 2.1 |
| 5 | 158 | 65 | 1.7 |
| 6 | 157 | 54 | ? |

# k-NEAREST NEIGHBOR LEARNING

| Sl. No. | Height | Weight | Target | Distance |
|---|---|---|---|---|
| 1 | 150 | 50 | 1.5 | 8.06 |
| 2 | 155 | 55 | 1.2 | 2.24 |
| 3 | 160 | 60 | 1.8 | 6.71 |
| 4 | 161 | 59 | 2.1 | 6.40 |
| 5 | 158 | 65 | 1.7 | 11.05 |
| 6 | 157 | 54 | ? | |

# k-NEAREST NEIGHBOR LEARNING

| Sl. No. | Height | Weight | Target | Distance | Nearest Points |
|---|---|---|---|---|---|
| 1 | 150 | 50 | 1.5 | 8.06 | |
| 2 | 155 | 55 | 1.2 | 2.24 | 1 |
| 3 | 160 | 60 | 1.8 | 6.71 | 3 |
| 4 | 161 | 59 | 2.1 | 6.40 | 2 |
| 5 | 158 | 65 | 1.7 | 11.05 | |
| 6 | 157 | 54 | ? | | |

# 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

- 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

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



---

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target |
|---|---|---|---|
| 1 | 150 | 50 | 1.5 |
| 2 | 155 | 55 | 1.2 |
| 3 | 160 | 60 | 1.8 |
| 4 | 161 | 59 | 2.1 |
| 5 | 158 | 65 | 1.7 |
| 6 | 157 | 54 | ? |

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance |
|---|---|---|---|---|
| 1 | 150 | 50 | 1.5 | 8.06 |
| 2 | 155 | 55 | 1.2 | 2.24 |
| 3 | 160 | 60 | 1.8 | 6.71 |
| 4 | 161 | 59 | 2.1 | 6.40 |
| 5 | 158 | 65 | 1.7 | 11.05 |
| 6 | 157 | 54 | ? | |

# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance | Nearest Points |
|---|---|---|---|---|---|
| 1 | 150 | 50 | 1.5 | 8.06 | |
| 2 | 155 | 55 | 1.2 | 2.24 | 1 |
| 3 | 160 | 60 | 1.8 | 6.71 | 3 |
| 4 | 161 | 59 | 2.1 | 6.40 | 2 |
| 5 | 158 | 65 | 1.7 | 11.05 | |
| 6 | 157 | 54 | ? | | |

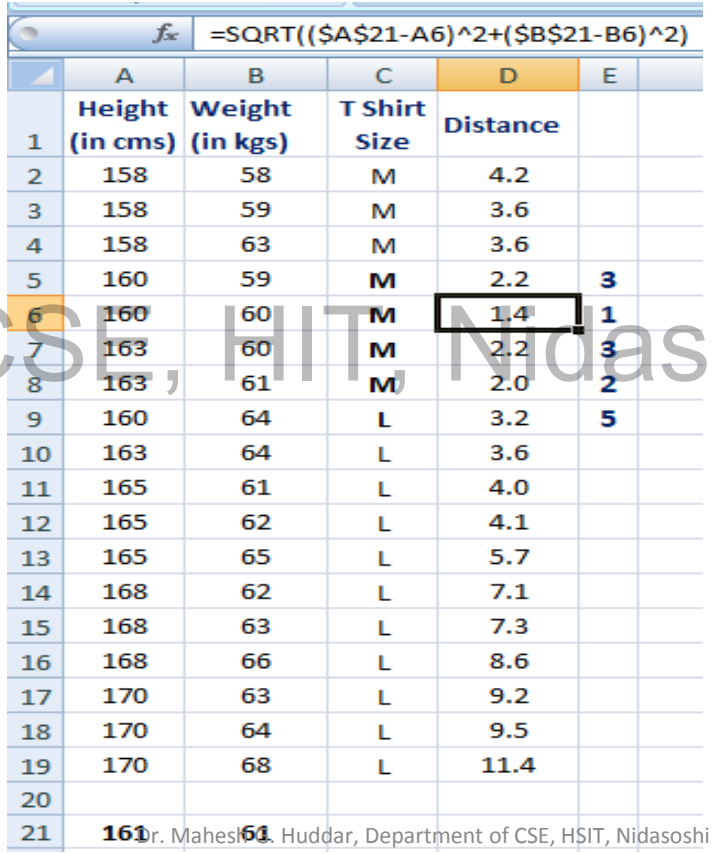
# Distance-Weighted Nearest Neighbor Algorithm

| Sl. No. | Height | Weight | Target | Distance | 1/distance$^2$ | Nearest Points |
|---|---|---|---|---|---|---|
| 1 | 150 | 50 | 1.5 | 8.06 | | |
| 2 | 155 | 55 | 1.2 | 2.24 | 0.45 | 1 |
| 3 | 160 | 60 | 1.8 | 6.71 | 0.15 | 3 |
| 4 | 161 | 59 | 2.1 | 6.40 | 0.16 | 2 |
| 5 | 158 | 65 | 1.7 | 11.05 | | |
| 6 | 157 | 54 | ? | | | |

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 XYZ' has height 161cm and weight 61kg.



=SQRT(($A$21-A6)^2+($B$21-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 | | | |

CSE, HIT, Nidasoshi

# LOCALLY WEIGHTED REGRESSION

- Locally weighted regression is instance based learning algorithm.
- 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.

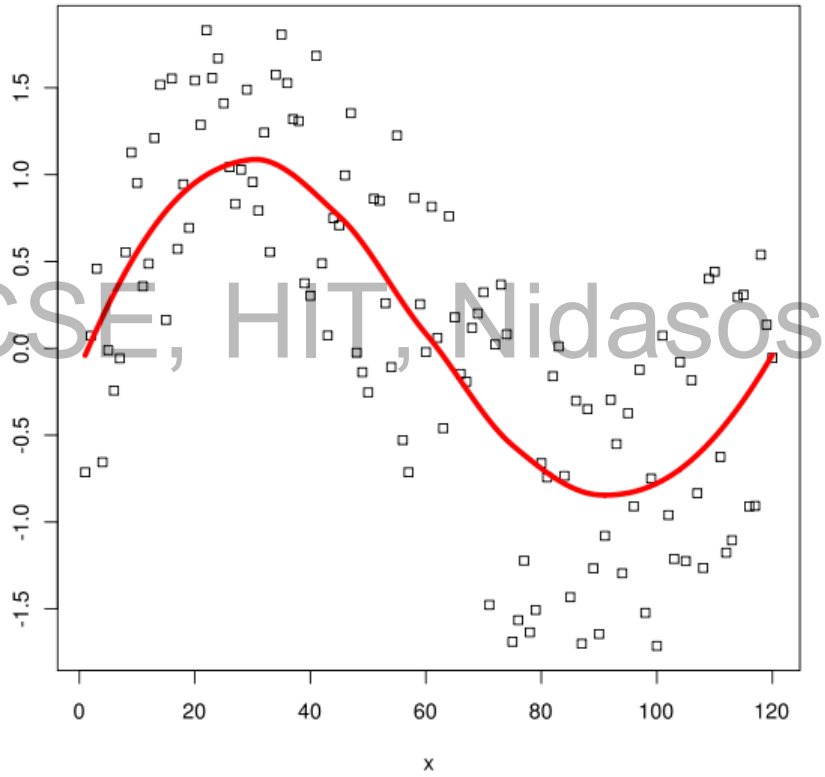
# LOCALLY WEIGHTED REGRESSION

- Given a new query instance $x_q$, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding of $x_q$.

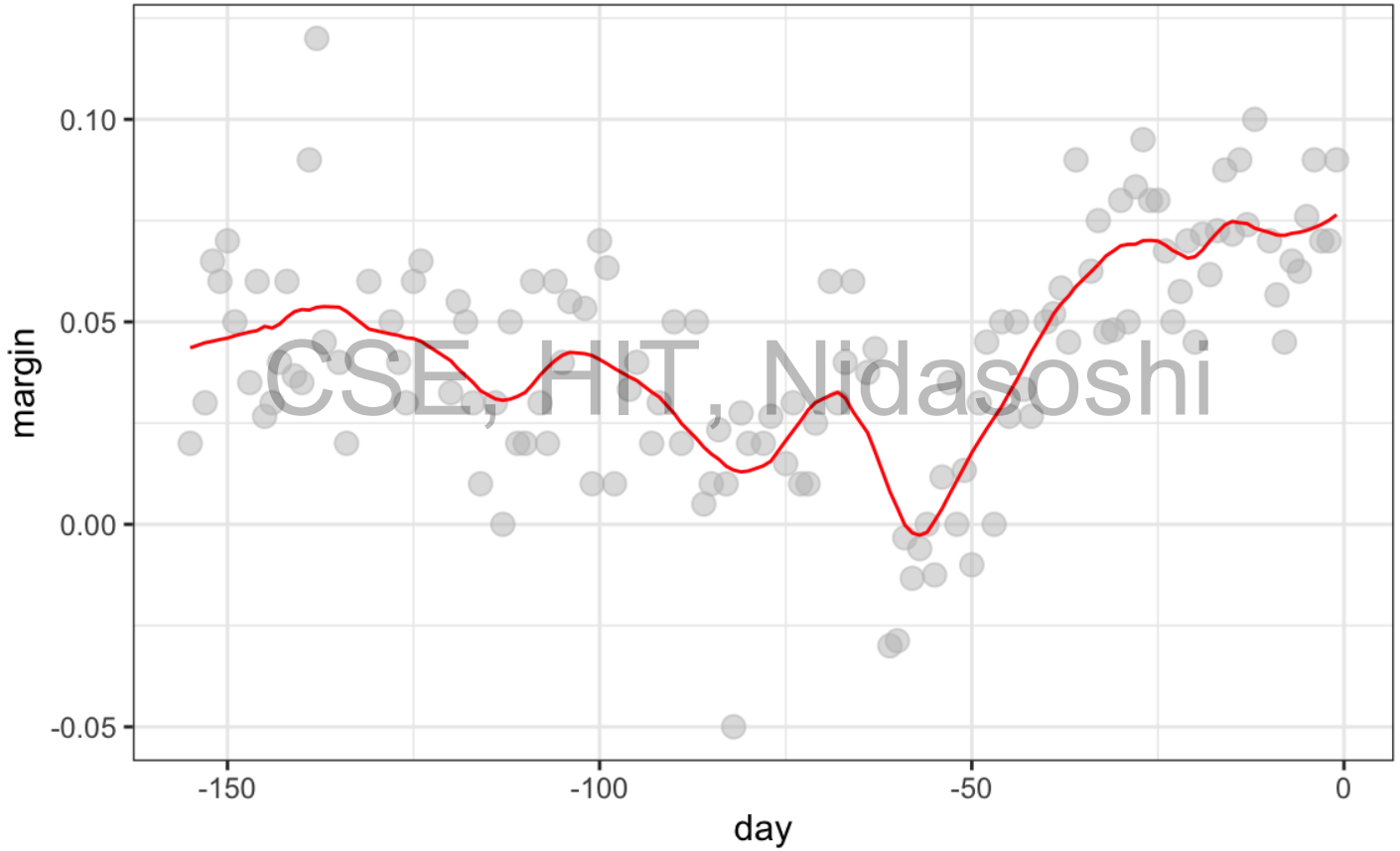


- This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.

# LOCALLY WEIGHTED REGRESSION



# LOCALLY WEIGHTED REGRESSION



# LOCALLY WEIGHTED REGRESSION

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

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

- Where, $a_i(x)$ 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 $x_q$ :

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

# 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 **$K(d(x_q, x))$,** 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 $\sigma_u^2$

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

- 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
- $\sigma^2$ 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

CSE, HIT, Nidasoshi

# CASE-BASED REASONING

- Instance-based methods such as k-NEAREST NEIGHBOR and locally weighted regression share three key properties.
- **First**, 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.
- **Second**, they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.
- **Third**, they represent instances **as** real-valued points in an n-dimensional Euclidean space.

# CASE-BASED REASONING

- In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation and the methods used to retrieve similar instances are correspondingly more elaborate.
- CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems

# CASE-BASED REASONING

- In **case-based reasoning**, the training examples, the **cases**, are stored and accessed to solve a new problem.
- To get a prediction for a new example, those cases that are similar, or close to, the new example are used to predict the value of the target features of the new example.
- This is at one extreme of the learning problem where, unlike decision trees and neural networks, relatively little work must be done offline, and virtually all of the work is performed at query time.

# CASE-BASED REASONING

Case-based reasoning consists of a cycle of the following four steps:

1. **Retrieve -** Given a new case, retrieve similar cases from the case base.
2. **Reuse -** Adapt the retrieved cases to fit to the new case.
3. **Revise -** Evaluate the solution and revise it based on how well it works.
4. **Retain -** Decide whether to retain this new case in the case base.

# CASE-BASED REASONING – Example

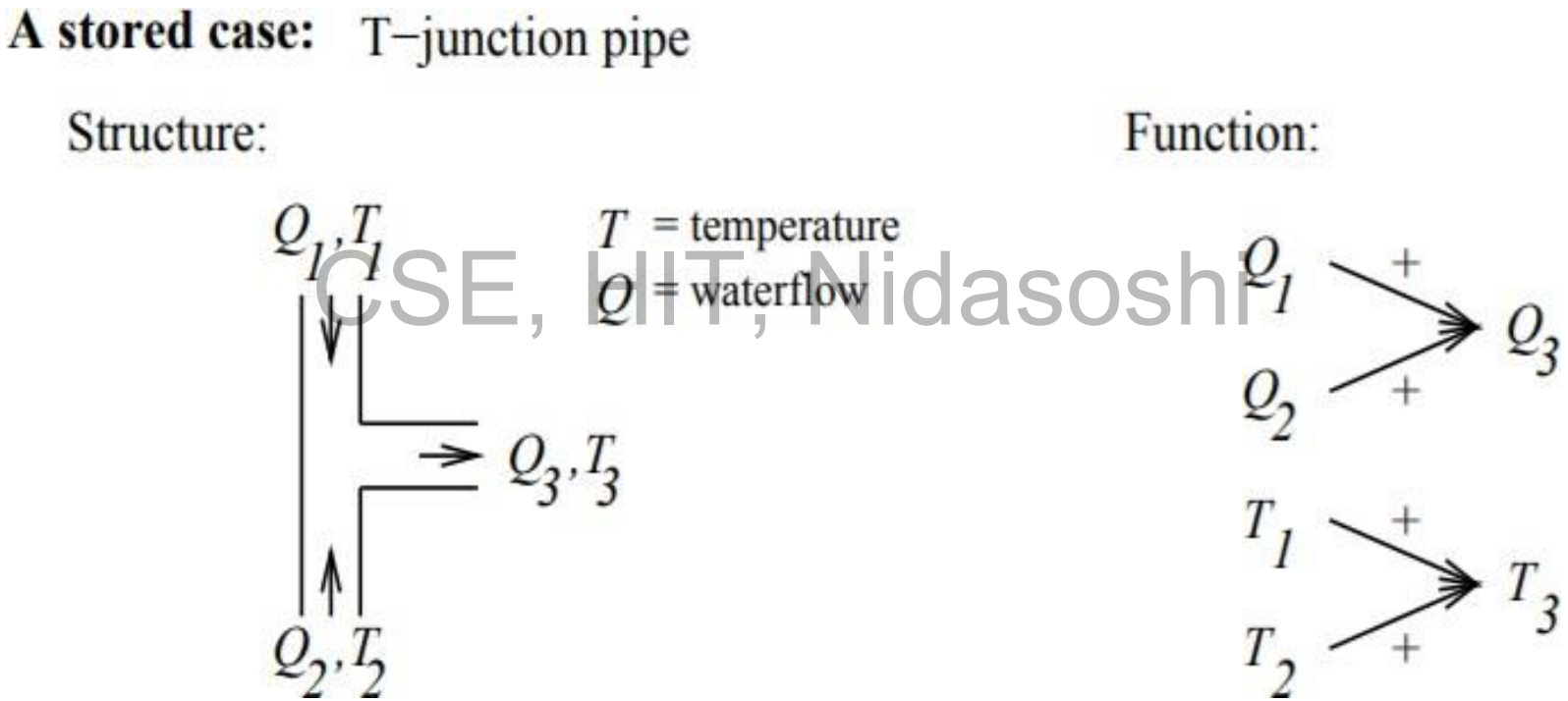
- A common example of a case-based reasoning system is a help desk that users call with problems to be solved.
- Case-based reasoning could be used by the diagnostic assistant to help users diagnose problems on their computer systems.
- When users give a description of a problem, the closest cases in the case base are retrieved. The diagnostic assistant could recommend some of these to the user, adapting each case to the user's particular situation.
- An example of adaptation is to change the recommendation based on what software the user has, what method they use to connect to the Internet, and the model of the printer.
- If one of the adapted cases works, that case is added to the case base, to be used when another user asks a similar question.
- In this way, all of the common different cases will eventually be in the case base.
- If none of the cases found works, some other method is attempted to solve the problem, perhaps by adapting other cases or having a human help diagnose the problem.
- When the problem is finally solved, the solution is added to the case base.

# A prototypical example of a case-based reasoning

- The CADET system employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.
- It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems.
- Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.
- New design problems are then presented by specifying the desired function and requesting the corresponding structure.

# A prototypical example of a case-based reasoning

- The problem setting is illustrated in below figure



# A prototypical example of a case-based reasoning

- The function is represented in terms of the qualitative relationships among the water-flow levels and temperatures at its inputs and outputs.
- In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable at the head decreases with the variable at the tail.
- Here Qc refers to the flow of cold water into the faucet, Qh to the input flow of hot water, and Qm to the single mixed flow out of the faucet.
- Tc, Th, and Tm refer to the temperatures of the cold water, hot water, and mixed water respectively.
- The variable Ct denotes the control signal for temperature that is input to the faucet, and Cf denotes the control signal for waterflow.
- The controls Ct and Cf are to influence the water flows Qc and Qh, thereby indirectly influencing the faucet output flow Qm and temperature Tm.

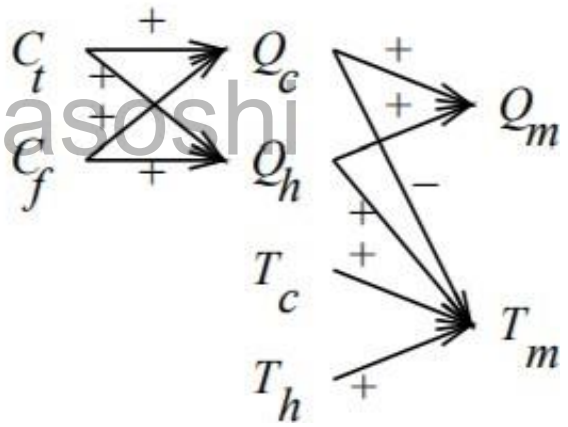
# A prototypical example of a case-based reasoning

**A problem specification:** Water faucet

Structure:

?

Function:



# A prototypical example of a case-based reasoning

- CADET searches its library for stored cases whose functional descriptions match the design problem.
- If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem.
- If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification.

CSE, HIT, Nidasoshi

# 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, $\pi$: **S → A**, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

# REINFORCEMENT LEARNING




$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Goal: Learn to choose actions that maximize

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

# Reinforcement learning problem characteristics

1. **Delayed reward**: The task of the agent is to learn a target function $\pi$ that maps from the current state s to the optimal action a = $\pi$ (s). In reinforcement learning, training information is not available in (s, $\pi$ (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, $\pi$**: S → A**, for selecting its next action a, based on the current observed state st; that is, $\pi$**(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 $V^{\pi}$ (st) achieved by following an arbitrary policy π from an arbitrary initial state st as follows:



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

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

# 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 $V^{\pi}(st)$ 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:

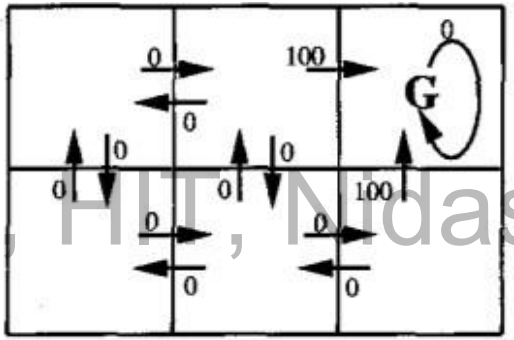
$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

# Reinforcement learning

**Example:**

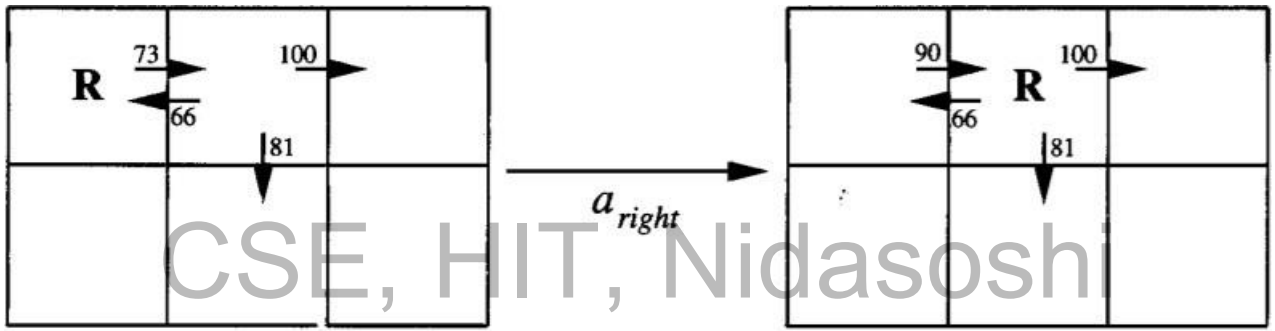
- A simple grid-world environment is depicted in the diagram




$r(s, a)$ (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).



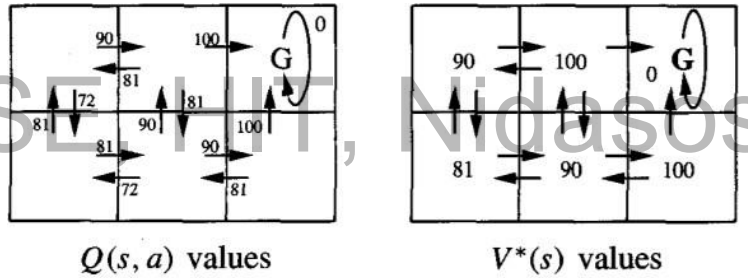
$$\begin{aligned}\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\ &\leftarrow 0 + 0.9 \max\{66, 81, 100\} \\ &\leftarrow 90\end{aligned}$$

# Reinforcement learning

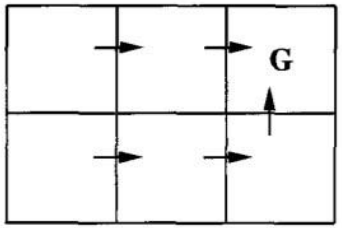
- The six grid squares in this diagram represent six possible states, or locations, for the agent.
- Each arrow in the diagram represents a possible action the agent can take to move from one state to another.
- The number associated with each arrow represents the immediate reward ***r(s, a)*** the agent receives if it executes the corresponding state-action transition
- The immediate reward in this environment is defined to be zero for all state-action transitions except for those leading into the state labelled G.
- The state G as the goal state, and the agent can receive reward by entering this state.

# Reinforcement learning

- Once the states, actions, and immediate rewards are defined, choose a value for the discount factor γ, determine the optimal policy π * and its value function V*(s).
- Let's choose γ = 0.9. The diagram at the bottom of the figure shows one optimal policy for this setting.




$Q(s, a)$ values $V^*(s)$ values




One optimal policy

# Reinforcement learning

- Values of V*(s) and Q(s, a) follow from r(s, a), and the discount factor γ = 0.9.
- An optimal policy, corresponding to actions with maximal Q values, is also shown.
- The discounted future reward from the bottom centre state is
- 0+ γ 100+ γ2 0+ γ3 0+... = 90