UNIT- IV

Instance-based Learning

- Instance-based learning methods simply store the training examples instead of learning explicit description of the target function.
  - Generalizing the examples is postponed until a new instance must be classified.
  - When a new instance is encountered, its relationship to the stored examples is examined in order to assign a target function value for the new instance.
- Instance-based learning includes nearest neighbor, locally weighted regression and case-based reasoning methods.
- 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 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.

Sample Complexity For Finite Hypothesis Space

- Sample Complexity
  - The growth in the number of required training examples with problem size is called the sample complexity of the learning problem
- We will consider only ***consistent learners***, which are those that maintain a training error of 0.
  - A learner is consistent if it outputs
  - hypotheses that perfectly fit the training data, whenever possible
- We can derive a bound on the number of training examples required by *any* consistent learner!

- The version space, $VS_{H,D}$ is defined as the set of all hypotheses h ϵ H that correctly classify the training examples D

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

- Every consistent learner outputs a hypothesis belonging to the version space, regardless of the instance space X, hypothesis space H, or training data D.
  - The reason is simply that by definition the version space $VS_{H,D}$ contains every consistent hypothesis in H.
- Therefore, to bound the number of examples needed by any consistent learner, we need only bound the number of examples needed to assure that the version space contains no unacceptable hypotheses.

Instance-based Learning vs Model-based Learning

- The main difference in these models is how they generalize information.
  - Instance-based learning will memorize all the data in a training set and then set a new data point to the same or average output value of the most common data point or similar data points it has memorized.
  - In model-based learning, the model would create a prediction line or prediction sections based on the different attributes of the data it trained on. A new data point would then fall along this line or within certain sections based on the attributes it possessed

Disadvantages

- One disadvantage of instance-based approaches is that the cost of classifying new instances can be high.
  - This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered
- A second disadvantage to many instance-based approaches, especially nearestneighbor 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 algorithm

- k-nearest neighbor algorithm is the most basic instance-based method.
- k-Nearest Neighbor Learning 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 Euclidean distance
- Euclidean distance between the instances xi = <xi1,…,xin> and xj = <xj1,…,xjn> are:

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

- For a given query instance x_q, $f(x_q)$ is calculated the function values of k-nearest neighbor of x_q
- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.
- Consider learning discrete-valued target functions of the form $f : R^n \rightarrow V$, where V is the finite set $\{v1, \ldots v_s\}$.
  - Store all training examples <xi,f(xi)>
  - Calculate f(xq) for a given query instance xq using k-nearest neighbor
  - The value 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 x,
  - If we choose k = 1, then the 1-NEAREST NEIGHBOR algorithm assigns to f^(xq) the value f (xi) 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.

- For Continuous-valued target functions:
  - calculate the mean value of the k nearest training examples rather than calculate their most common value.
  - More precisely, to approximate a real-valued target function f : R^n -> R we replace the final line of the above algorithm by:

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

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.

Distance-Weighted Nearest Neighbor Algorithm

- One 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 x_q, giving greater weight to closer neighbors.
- we might weight the vote of each neighbor according to the inverse square of its distance from x_q
- This can be accomplished by replacing the final line of the algorithm by

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

- where

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

- We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

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

Locally Weighted Regression

- The basic assumption for a linear regression is that the data must be linearly distributed.
- But what if the data is not linearly distributed. Can we still apply the idea of regression?
  - And the answer is ‘ *yes’* … we can apply regression and it is called as locally weighted regression
- Regression is a statistical tool used to understand and quantify the relation between two or more variables.
  - Regression range from simple models to complex equations.

- Most of the algorithms such as classical feedforward neural network, support vector machines, nearest neighbor algorithms etc. are global learning systems or global function approximations where it is used to minimize the global loss functions such as sum squared error.
- In contrast, local learning systems will divide the global learning problem into multiple smaller/simpler learning problems and this is usually achieved by dividing the cost function into multiple independent local cost functions.
- The disadvantage of global methods is that sometimes no parameter values can provide a sufficiently good approximation.

- An alternative to global function approximation is Locally Weighted Learning .
- Locally Weighted Learning methods are non-parametric and the current prediction is done by local functions.
  - The basic idea behind LWL is that instead of building a global model for the whole function space, for each point of interest a local model is created based on neighboring data of the query point.
  - For this purpose, each data point becomes a weighting factor which expresses the influence of the data point for the prediction.
  - In general, data points which are in the close neighborhood to the current query point are receiving a higher weight than data points which are far away.
- LWL is also called lazy learning because the processing of the training data is shifted until a query point needs to be answered

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

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

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

  - ai(x) denotes the value of the ith attribute of the instance x

- We know that gradient descent method can be used to find the coefficients wo . . . wn to minimize the error in fitting linear functions to a given set of training examples

- Derived methods used to chose the weights to minimize the squared error summed over set of D 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
-

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

- We 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:

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

  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 xq :

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

3. Combine 1 and 2:

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

Criterion three is a good approximation to criterion two and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number k of neighbors considered.

Rederive the gradient descent rule using the same style of argument we obtain the following training rule

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

Radial Basis Functions

- Is one approach for function approximation that is closely related to distance-weighted regression and also to artificial neural networks
- In this approach, the learned hypothesis is a function of the form:

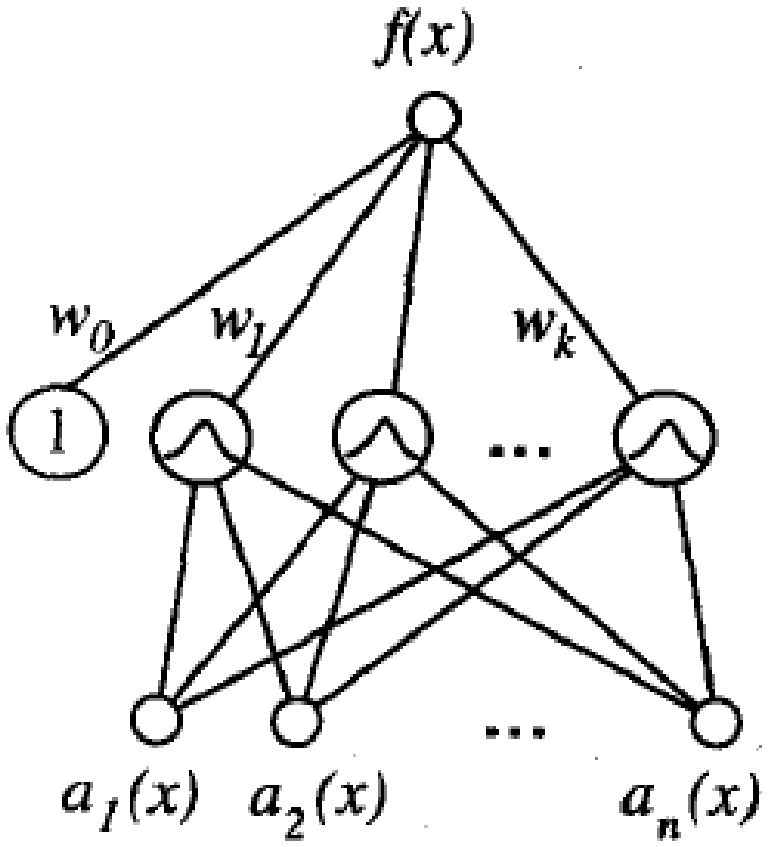
$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

- 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.

- Even though f(x) is a global approximation to f (x), the contribution from each of the Ku(d (xu, x)) terms is localized to a region nearby the point xu.
- It is common to choose each function Ku (d (xu, x)) to be a Gaussian function centered at the point xu with some variance $\sigma^2{}_u$

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

- The function given by can be viewed as describing a two layer network where the first layer of units computes the values of the various Ku(d(xu, x)) and where the second layer computes a linear combination of these first-layer unit values.



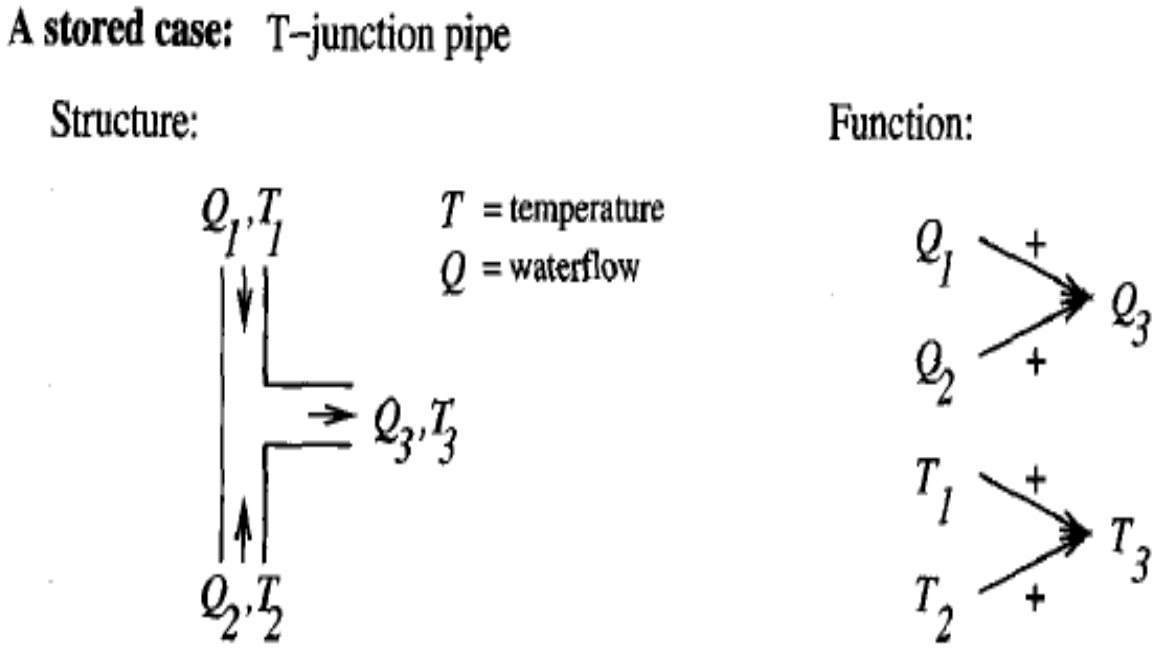
Case-based Reasoning(CBR)

- 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 (CBR) is a learning paradigm based on the first two of these principles, but not the third.
- In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.

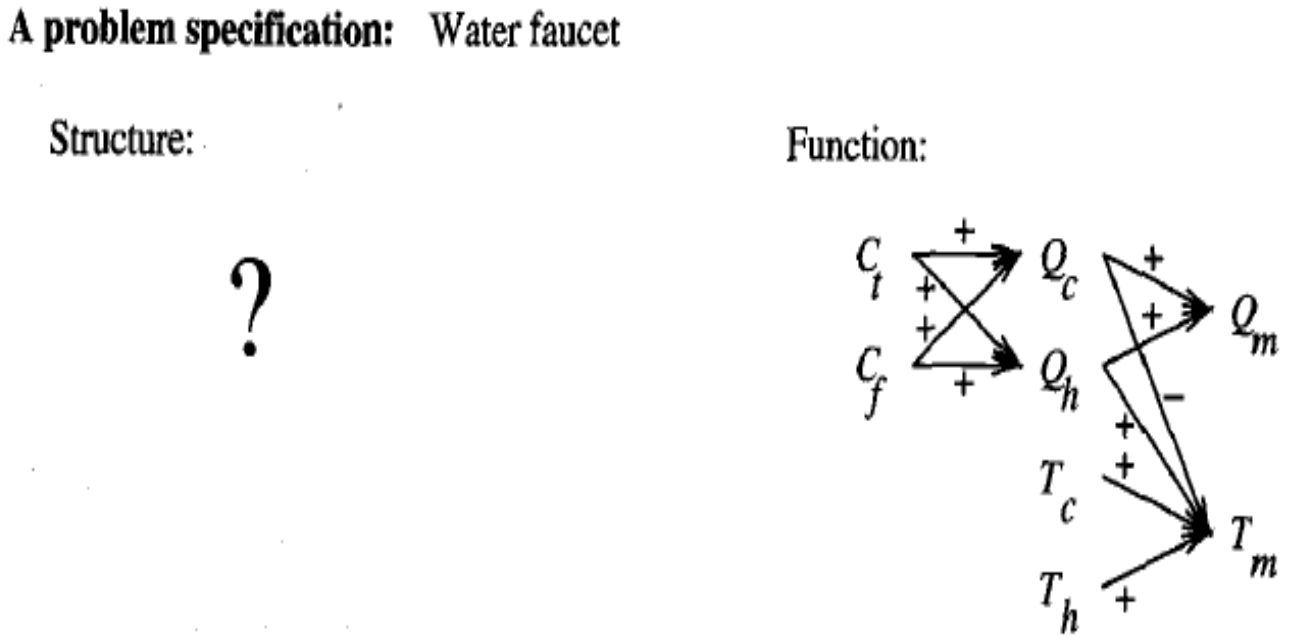
- CBR has been applied to problems such as
  - conceptual design of mechanical devices based on a stored library of previous designs,
  - reasoning about new legal cases based on previous rulings,
  - and solving planning and scheduling problems by reusing and combining portions of previous solutions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

- Consider a prototypical example of a case-based reasoning system CADET system which 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.

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



- This figure depicts a new design problem described by its desired function.
- This particular function describes the required behavior of one type of water faucet. Here 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.
- Similarly, 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.
- Note the description of the desired function specifies that these controls Ct and Cf are to influence the water flows Qc, and Qh, thereby indirectly influencing the faucet output flow Qm, and temperature Tm.



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

- More generally, CADET searches for subgraph isomorphisms between the two function graphs, so that parts of a case can be found to match parts of the design specification.
- Furthermore, the system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases.
- By retrieving multiple cases that match different subgraphs, the entire design can sometimes be pieced together.
- In general, the process of producing a final solution from multiple retrieved cases can be very complex.
  - It may require designing portions of the system from first principles, in addition to merging retrieved portions from stored cases.
  - It may also require backtracking on earlier choices of design subgoals and, therefore, rejecting cases that were previously retrieved.

- CADET has very limited capabilities for combining and adapting multiple retrieved cases to form the final design and relies heavily on the user for this adaptation stage of the process.
- The correspondence between the problem setting of CADET and the general setting for instance-based methods such as k-NEAREST NEIGHBOR:
  - In CADET each stored training example describes a function graph along with the structure that implements it. New queries correspond to new function graphs. Thus, we can map the CADET problem into our standard notation by defining the space of instances X to be the space of all function graphs.
  - The target function f maps function graphs to the structures that implement them. Each stored training example (x, f (x)) is a pair that describes some function graph x and the structure f (x) that implements x. The system must learn from the training example cases to output the structure f (xq) that successfully implements the input function graph query xq.

- Generic 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. This may require a similarity metric different from Euclidean distance, such as the size of the largest shared subgraph between two function graphs.
  - Multiple retrieved cases may be combined to form the solution to the new problem. This is similar to the k-NEAREST NEIGHBOR approach, in that multiple similar cases are used to construct a response for the new query.
  - There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving. One simple example of this is found in CADET, which uses generic knowledge about influences to rewrite function graphs during its attempt to find matching cases.

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