INSTANCE-BASED LEARNING

Dr. Nagaratn P Hegde
Professor, Dept of CSE

- **Instance-based Learning:** Introduction, k-Nearest Neighbor-Distance Weighted Nearest Neighbor Algorithm, Locally Weighted Regressions, Radial Basis Functions, Case – based learning.
- **Reinforcement Learning:** Introduction, Learning Task, Q Learning.

instance-based learning

- methods simply store the training examples. Generalizing beyond these examples is postponed until a new instance must be classified.
- Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Instancebased 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.
- Many techniques construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire instance space. This has significant advantages when the target function is very complex, but can still be described by a collection of less complex local approximations.

case-based learning

- Instance-based methods can also use more complex, symbolic representations for instances. In case-based learning, instances are represented in this fashion and the process for identifying "neighboring" instances is elaborated accordingly.
- Case-based reasoning has been applied to tasks such as storing and reusing past experience at a help desk, reasoning about legal cases by referring to previous cases, and solving complex scheduling problems by reusing relevant portions of previously solved problems.
- 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.

- Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time.
- 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 LEARNING

- The most basic instance-based method is the k-NEARESNTE IGHBOaRlg orithm. This algorithm assumes all instances correspond to points in the n-dimensional space $\Re^n$
- *The nearest neighbors of an instance are defined in terms of the standard* Euclidean distance.

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

- let an arbitrary instance *x be described by the* feature vector

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

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

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

k-NEAREST NEIGHBOR LEARNING

consider learning discrete-valued target functions of the form $f : \Re^n \rightarrow V$ where V is the finite **set *{vl, . . v,}.***

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

k-NEAREST NEIGHBOR LEARNING

Training algorithm:

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

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ 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.

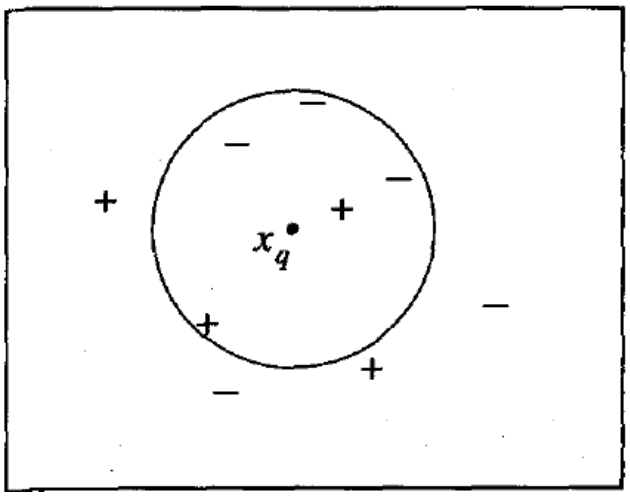
TABLE 8.1

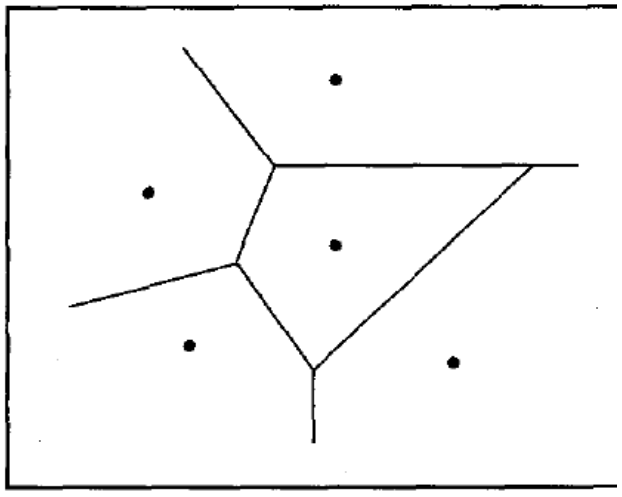
The *k*-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

KNN

- 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
- 1-Nearest learning algorithm classifies *x, as a positive example in this figure,* whereas the 5-NEARESTE learning algorithm classifies it as a negative example.
- the k- 1-Nearest learning algorithm never forms an explicit general hypothesis $\hat{f}$ *regarding the target function f . It simply* computes the classification of each new query instance as needed.

KNN





The diagram on the right side of Figure shows the shape of this decision surface induced by 1-Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples. For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the **Voronoi *diagram of the set of training examples***

KNN

The KNN algorithm is easily adapted to approximating continuous-valued target functions. To accomplish this, we have the algorithm 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 $f : \Re^n \rightarrow \Re$

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

Distance-Weighted Nearest Neighbor algorithm

- Refinement to the Nearest Neighbor algorithm is to weight the contribution of each of the k neighbors according to their distance to the query pointx_q, ***giving greater weight to closer neighbors***
- This can be accomplished by replacing the final line of the algorithm by $\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$

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

To accommodate the case where the query point x_q exactly matches one of the training instances x_i and the denominator $d(x_q, x_i)^2$ is therefore zero, we assign $\hat{f}(x_q)$ to be $f(x_i)$ in this case. If there are several such training examples, we assign the majority classification among them.

Distance-Weighted Nearest Neighbor algorithm

- for real-valued target functions

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

it assures that if $f(x_i) = c$ for all training examples, then $f(x_q) \leftarrow c$ as well

The only disadvantage of considering all examples is that our classifier will run more slowly. If all training examples are considered when classifying a new query instance, we call the algorithm a *global method.* If only the nearest training examples are considered, we call it a *local method.*

When the rule is applied as a global method, using all training examples, it is known as Shepard's method (Shepard 1968).

KNN

- The distance-weighted k-NN is a highly effective inductive inference method for many practical problems. It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data.
- By taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples
- The inductive bias corresponds to an assumption that the classification of an instance ***xq, will be most similar to the classification of other instances that are nearby in*** Euclidean distance.

KNN

- One practical issue in applying NN is that the distance between instances is calculated based on *all attributes of the instance* (i.e., on all axes in the Euclidean space containing the instances). This lies in contrast to methods such as rule and decision tree learning systems that select only a subset of the instance attributes when forming the hypothesis.

curse of dimensionality.

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

curse of dimensionality.

- One interesting approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances. This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes. The amount by which each axis should be stretched can be determined automatically using a cross-validation approach

curse of dimensionality

- To see how, first note that we wish to stretch (multiply) the jth axis by some factor *zj, where* the values zl . . . z, are chosen to minimize the true classification error of the learning algorithm. Second, note that this true error can be estimated using crossvalidation.
- Hence, one algorithm is to select a random subset of the available data to use as training examples, then determine the values of zl . . . z, that lead to the minimum error in classifying the remaining examples. By repeating this process multiple times the estimate for these weighting factors can be made more accurate. This process of stretching the axes in order to optimize the performance of k-NN provides a mechanism for suppressing the impact of irrelevant attributes.

curse of dimensionality

- An even more drastic alternative is to completely eliminate the least relevant attributes from the instance space. This is equivalent to setting some of the zi scaling factors to zero. Moore and Lee (1994) discuss efficient cross-validation methods for selecting relevant subsets of the attributes
- explore methods based on leave-one-out crossvalidation, in which the set of *m training instances is repeatedly divided into a* training set of size m - 1 and test set of size 1, in all possible ways. This leave-oneout approach is easily implemented in k-NEARESNT EIGHBOaRl gorithms because no additional training effort is required each time the training set is redefined.

curse of dimensionality

- Stretch each axis by a value that varies over the instance space. However, as we increase the number of degrees of freedom available to the algorithm for redefining its distance metric in such a fashion, we also increase the risk of overfitting. Therefore, the approach of locally stretching the axes is much less common.

KNN

- One additional practical issue in applying k-NN efficient memory indexing. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query.
- Various methods have been developed for indexing the stored training examples so that the nearest neighbors can be identified more efficiently at some additional cost in memory. One such indexing method is the kd-tree (Bentley 1975; Friedman et al. 1977), in which instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query *xq, to the relevant leaf by testing selected attributes of xq,.*

Terminology

Much of the literature on nearest-neighbor methods and weighted local regression uses a terminology that has arisen from the field of statistical pattern recognition.

it is useful to know the following terms:

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

LOCALLY WEIGHTED REGRESSION

- The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function f ***(x) at the single query point x = xq.*** Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to f over a local region surrounding x***q,. Locally weighted*** regression uses nearby or distance-weighted training examples to form this local approximation to f
- we might approximate the target function in the neighborhood surrounding ***x, using a linear function, a quadratic function,*** a multilayer neural network, or some other functional form. The phrase "locally weighted regression" is called ***local because the function is approximated based a*** only on data near the query point, ***weighted because the contribution of each*** training example is weighted by its distance from the query point, and ***regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Regression

- Given a new query instance ***x,, the general approach in locally weighted*** regression is to construct an approximation *f^ that fits the training examples in the* neighborhood surrounding ***x,. This approximation is then used to calculate the*** value ***f"(x,), which is output as the estimated target value for the query instance.***
- The description of *f^ may then be deleted, because a different local approximation* will be calculated for each distinct query instance.

Locally Weighted Linear Regression

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

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

ai(x) denotes the value of the ith attribute of the instance x***.***

In gradient descent find the coefficients ***wo . . . w, to minimize the error in fitting such linear functions*** to a given set of training examples using a global approximation to the target function. Therefore, we derived methods to choose weights that minimize the squared error summed over the set D of training examples

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

gradient descent training rule

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

where *q is a constant learning rate*

local approximation

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

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

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:

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

3. Combine 1 and 2:

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

Locally Weighted Regression

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

contribution of instance ***x to the weight update is now*** multiplied by the distance penalty ***K(d(x,, x)) , and that the error is summed over*** only the k nearest training examples. In fact, if we are fitting a linear function to a fixed set of training examples, then methods much more efficient than gradient descent are available to directly solve for the desired coefficients ***wo . . . Urn***

More complex functional forms are not often found because (1) the cost of fitting more complex functions for each query instance is prohibitively high, and (2) these simple approximations model the target function quite well over a

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
- the learned hypothesis is a function of the form

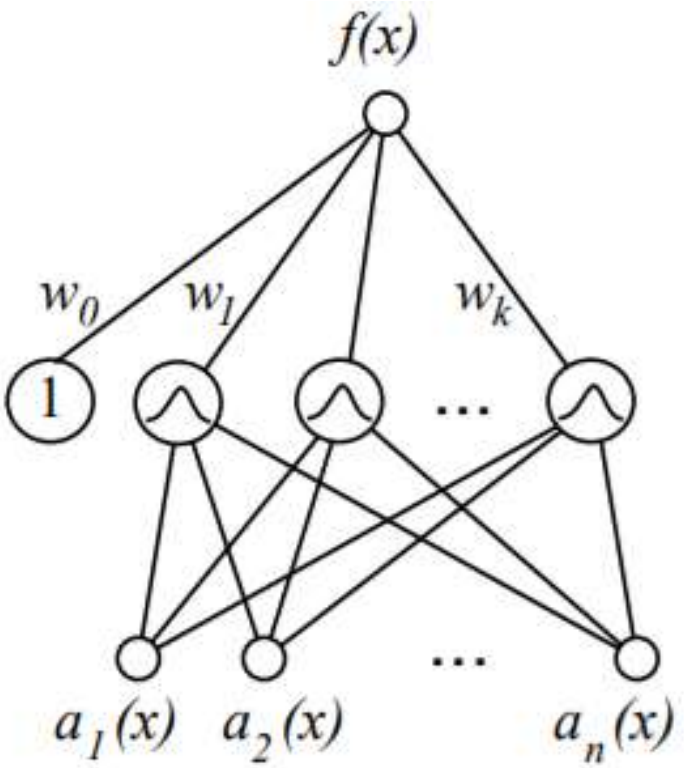
$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

where each ***xu is an instance from X and where the kernel function K,(d(xq,, x))***is defined so that it decreases as the distance ***d(xq,, x) increases. Here k is a userprovided*** constant that specifies the number of kernel

RBF

- $\hat{f}(x)$ *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 xu. It is common* to choose each function $K_u(d(x_u, x))$ ***to*** be a Gaussian function centered at the point xu with some variance σ_u^2.
- The function 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.

RBF



Each hidden unit produces an activation determined by a Gaussian function centered at some instance *xu. Therefore, its activation will be close to zero* unless the input x is near *xu. The output unit produces a linear* combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

RBF

- 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 xu and* σ_u^2. that define its kernel function $K_u(d(x_u, x))$ *Second, the weights* w_u*, are trained to* maximize the fit of the network to the training data, using the global error criterion. Because the kernel functions are held fixed during this second stage, the linear weight values *wu, can be trained very efficiently***.**

- One approach is to allocate a Gaussian kernel function for each training example *(xi, f (xi)), centering* this Gaussian at the point *xi. Each of these kernels may be assigned the same width* σ_u^2.
- Given this approach, the RBF network learns a global approximation to the target function in which each $\hat{f}$ training example *(xi, f (xi)) can influence the value* of only in the neighborhood of *xi. One advantage of this choice of kernel* functions is that it allows the RBF network to fit the training data exactly. That is, for any set of m training examples the weights *wo . . . w, for combining the* m Gaussian kernel functions can be set so that $\hat{f}(x_i) = f(x_i)$ *for each training*

RBF

A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large. The set of kernel functions may be distributed with centers spaced uniformly throughout the instance space X. Alternatively, we may wish to distribute the centers nonuniformly, especially if the instances themselves are found to be distributed nonuniformly over X. In this later case, we can pick kernel function centers by randomly selecting a subset of the training instances, thereby sampling the underlying distribution of instances.

RBF

- Alternatively, we may identify prototypical clusters of instances, then add a kernel function centered at each cluster. The placement of the kernel functions in this fashion can be accomplished using unsupervised clustering algorithms that fit the training instances (but not their target values) to a mixture of Gaussians.

RBF

- The EM algorithm provides one algorithm for choosing the means of a mixture of k Gaussians to best fit the observed instances. In the case of the EM algorithm, the means are chosen to maximize the probability of observing the instances xi, given the k estimated means. Note the target function value f (xi) of the instance does not enter into the calculation of kernel centers by unsupervised clustering methods. The only role of the target values f (xi) in this case is to determine the output layer weights wu,.

RBF

- radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions. The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular center and width.
- Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function. One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with BACKPROPAGATTIOhNis . follows from the fact that the input layer and the output layer of an RBF are trained separately.

CASE-BASED REASONING

- Instance-based methods such as K-NN 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 thequery.
- 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

CBR

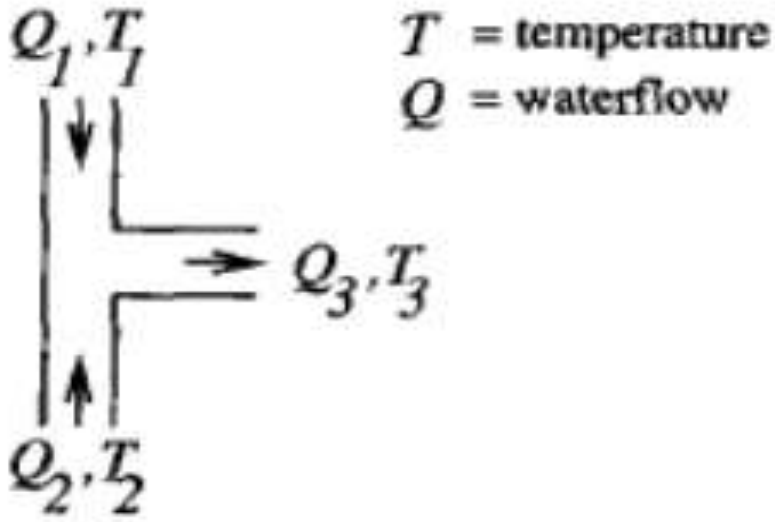
- In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs (Sycara et al. 1992), reasoning about new legal cases based on previous rulings (Ashley 1990), and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems (Veloso 1992

CBR

- The CADET system (Sycara et al. 1992) employs casebased 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 (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 stored case: T–junction pipe

Structure:



Function:

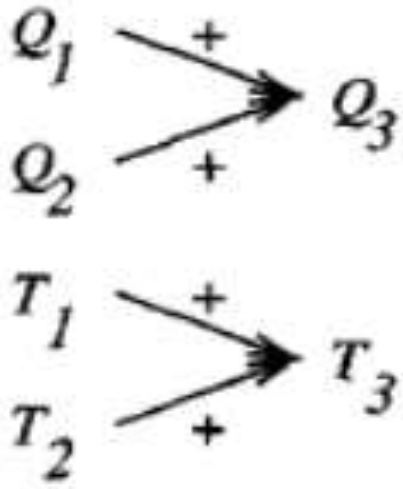


figure shows the description of a typical stored case called a T-junction pipe. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs

CBR

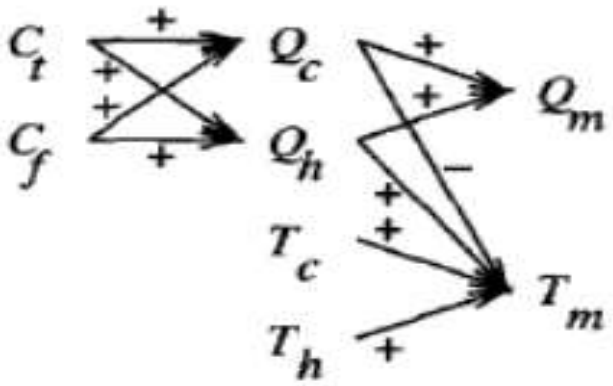
- In the functional description at its right, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. For example, the output waterflow ***Q3 increases with increasing input waterflow Ql,*** a "-" label indicates that the variable at the head decreases with the variable at the tail
- a new design problem is described by its desired function. 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,.*

A problem specification: Water faucet

Structure:

?

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 ***C, denotes the control signal*** for temperature that is input to the faucet, and Cf denotes the control signal for waterflow

CBR

- 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
- the T-junction function matches a subgraph of the water faucet function graph.

CBR

- **CADET searches for subgraph isomorphisms between the** two function graphs, so that parts of a case can be found to match parts of the design specification. Furthermore, the system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases.
- It uses general knowledge about physical influences to create these elaborated function graphs. For example, it uses a rewrite rule that allows it to rewrite the influence
- $A \xrightarrow{+} B$ as $A \xrightarrow{+} x \xrightarrow{+} B$

CBR

- This rewrite rule can be interpreted as stating that if B must increase with A, then it is sufficient to find some other quantity *x such that B increases with x,* and *x increases with A. Here* x is a universally quantified variable whose value is bound when matching the function graph against the case library
- 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.
- **CADET is** a research prototype system intended to explore the potential role of case-based reasoning in conceptual design. It does not have the range of analysis algorithms needed to refine these abstract conceptual designs into final designs.
- In CADET each stored training example describes a function graph along with the structure that implements it. New queries correspond to new function graphs

CBR

- 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,.

CBR

- 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-NN Aproach, in that multiple similar cases are used to construct a response for the new query.
- However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.

CBR

- 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 knowledgebased reasoning and search-intensive problem-solving methods

LAZY AND EAGER LEARNING

- lazy learning methods: the k-NN algorithm, locally weighted regression, and case-based reasoning.