Unit 3

Instance Based Learning

Instance-based learning methods are conceptually straightforward approach to approximating real-valued or discrete-valued target functions. Each learning algorithm mainly comprises two phases- training phase and testing phase.

Instance based Learning algorithms do not perform any processing on training data during training phase, instead simply store the presented training data. In testing phase (classification or regression) when a new query instance is encountered, a set of similar related training instances is retrieved from memory and used to classify or predict the target function value of the new query instance.

Instance-based methods are sometimes referred to as **"lazy"** learning methods because they delay processing until a new instance must be classified.

Advantages of Instance based learning-

1. 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.
2. Instance-based methods can also use more complex, symbolic representations for instances.

Disadvantages of Instance based Learning-

1. The cost of classifying new instances can be high.
2. Many instance-based approaches, especially nearest-neighbor approaches, typically consider all attributes of the instances when attempting to retrieve similar training examples from memory.

K-NEAREST NEIGHBOR LEARNING :

KNN can be used for approximating discrete valued target function and continuous valued target function for a test instance. Here, this algorithm assumes all instances correspond to points in the n-dimensional space ‘R’. Let an arbitrary instance x be described by the feature vector-

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

Where, n is the total number of attributes and a_r (x) denotes the value of the rth attribute of instance x. Then the Euclidean distance is used to define similarity between two instances xi and xj which is given by d(xi, xj), 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 (classification) or real-valued (Regression).

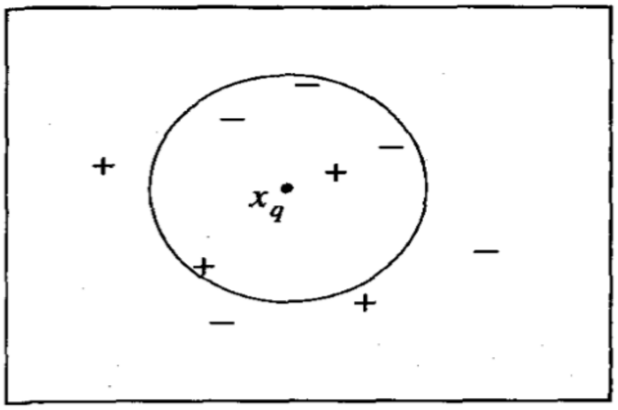
Classification :

Let us first consider learning discrete-valued target functions of the form f : R →V, where V is the finite set {v_1, . . . v_n}.

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
              $\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$
          where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.
```

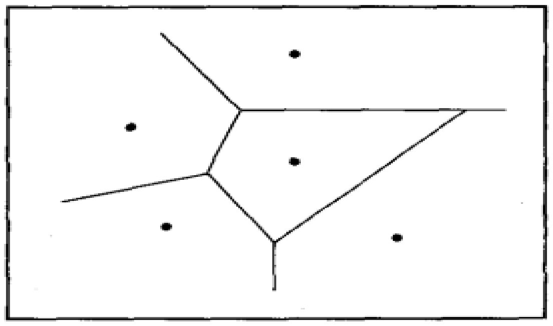
The value f (x_q) returned by this algorithm as its estimate of f (x_q) is just the most common value of f among the k training examples nearest to x_q. If we choose k = 1, then the 1-NEAREST NEIGHBOR algorithm assigns to f(x_q) the value f (x_i) where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the most common value among the k nearest training examples.



Here, for 1NN the target function value for test instance x_q is +, for 5NN the target function value for test instance x_q is -.

VORONAI DIAGRAM:

When we consider 1NN, what is the decision surface induced 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.



Regression:

The k-NEAREST NEIGHBOR 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, we have-

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

Distance-Weighted NEAREST NEIGHBOR Algorithm:

One refinement is added to k-NN algorithm, that is to weight the contribution of each of the k neighbors according to their distance to the query point x, giving greater weight to closer neighbours. The weight to the vote of each neighbour can be assigned according to the inverse square of its distance from x_q.

To approximate a discrete-valued target function, we have-

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

where

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

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.

To approximate real-valued target functions in a similar fashion, we have

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

Note the denominator is a constant that normalizes the contributions of the various weights (e.g., it assures that if f (xi) = c for all training examples then value of target function returned will also be c.

Important Remarks on k-NN Algorithm:

1.The distance-weighted k-NEAREST NEIGHBOR algorithm is a highly effective inductive inference method for many practical problems.

2.It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data.

3.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.

4. **The inductive bias** of KNN corresponds to an assumption that the classification of an instance x_q, will be most similar to the classification of other instances that are nearby in Euclidean distance.

5. One practical issue is when many irrelevant attributes are also present in feature vector of training instances, is sometimes referred to as the **curse of dimensionality**. Nearest-neighbor approaches are especially sensitive to such problem.

6. Here true error can be estimated efficiently using cross- validation. Hence, one algorithm is to select a random subset of the available data to use as training examples, then determine the values of z1 . . . z n, 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.

7. Another practical issue in applying k-NEAREST NEIGHBOR is the requirement of efficient memory indexing.

IMPORTANT TERMS:

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

LOCALLY WEIGHTED REGRESSION:

Locally weighted regression is a generalization approach of KNN (approximation of target function at a single query point). Locally weighted regression uses nearby or distance-weighted training examples to form the local approximation to f. It constructs an explicit approximation to f over a local region surrounding x_q for all neighbouring training instances.

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 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 $\hat{f}$ that fits the training examples in the neighborhood surrounding 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. The description of $\hat{f}$ may then be deleted, because a different local approximation will be calculated for each distinct query instance. (f: target function, output: approximation of f: $\hat{f}$)

<u>Locally Weighted Linear Regression:</u>

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

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

As before, $a_i(x)$ denotes the value of the ith attribute of the instance x.

To find the global approximation to the target function, methods are derived (as in gradient descent) to choose weights that minimize the squared error summed over the set D of training examples

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

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, and where the training rule has been re-expressed from the notation of Chapter 4 to fit our current notation (i.e., $t \rightarrow f(x)$, $o \rightarrow \hat{f}(x)$, and $x_j \rightarrow a_j(x)$).

How shall we 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. Note we write the error $E(x_q)$ to emphasize the fact that now the error is being defined as a function of the query point x_q.

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

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

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

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

3. Combine 1 and 2:

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

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. If we choose criterion three above and rederive the gradient descent rule-

$$\Delta w_j = \eta \sum_{x \in \text{ k nearest nbrs of } x_q} K(d(x_q, x))\ (f(x) - \hat{f}(x))\ a_j(x) \qquad (8.7)$$

Notice the only differences between this new rule and the rule given by Equation (8.6) 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. 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 $w_0 \ldots w_n$.

Limitation-

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

RADIAL BASIS FUNCTIONS:

Radial basis functions are used for function approximation that is closely related to distance-weighted regression and also to artificial neural networks. 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 feed forward networks trained with BACKPROPAGATION as the input layer and the output layer of an RBF are trained separately.

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 $\hat{f}(x)$ 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 x_u. It is common

to choose each function $K_u(d(x_u, x))$ to be a Gaussian function (see Table 5.4) centered 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 function given by the above Equation 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.

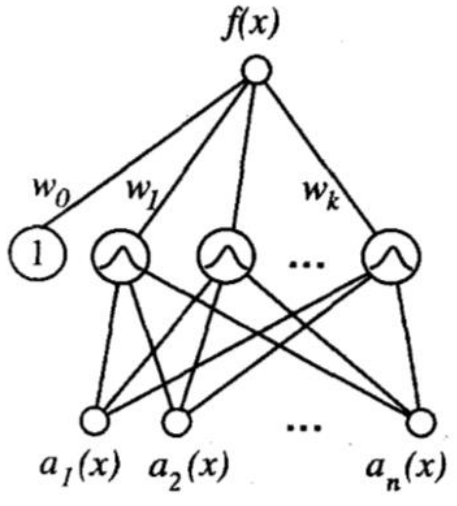


FIGURE 8.2
A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance x_u. Therefore, its activation will be close to zero unless the input x is near x_u. The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

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_u are trained to maximize the fit of the network to the training data, using the global error criterion given by Equation (8.5). Because the kernel functions are held fixed during this second stage, the linear weight values w_u can be trained very efficiently.

Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions

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. Given this approach, the RBF network learns a global approximation to the target function in which each training example (xi, f (xi)) can influence the value of $\hat{f}$ 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.

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 uni- formly 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. 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.

Case-based reasoning:

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.

Case-based reasoning (CBR) is a learning paradigm based on the following two principles-

1. 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.
2. Second, they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query.

In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.

Applications of CBR:

1. Conceptual design of mechanical devices based on a stored library of previous designs (Sycara et al. 1992),
2. Reasoning about new legal cases based on previous rulings (Ashley 1990),
3. Solving, planning and scheduling problems by reusing and combining portions of previous solutions to similar problems (Veloso 1992).

Example : The CADET system (Sycara et al. 1992) 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 top half of 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. Similarly a "-" label indicates that the variable at the head decreases with the variable at the tail.

The bottom half of 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.

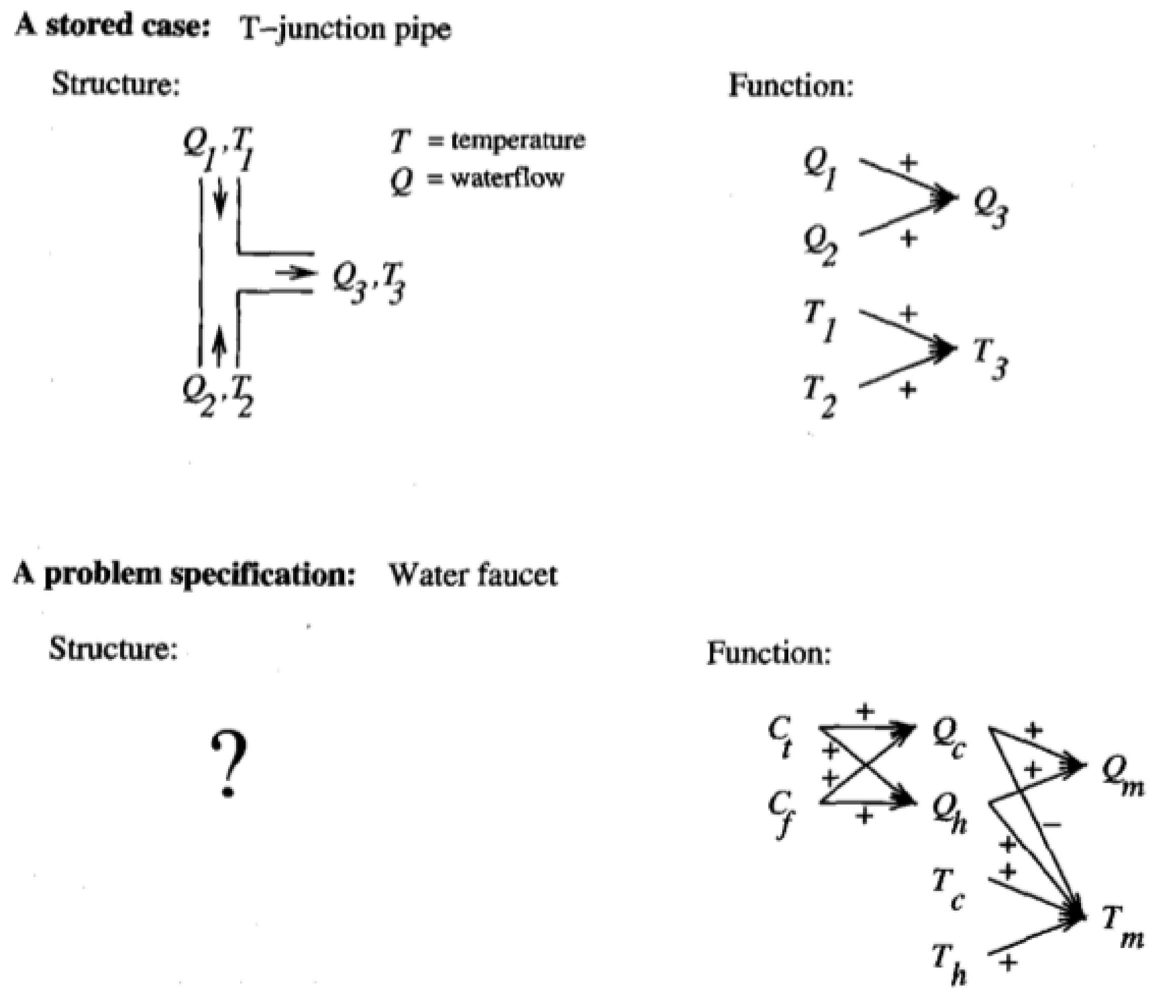


FIGURE 8.3
A stored case and a new problem. The top half of the figure describes a typical design fragment in the case library of CADET. The function is represented by the graph of qualitative dependencies among the T-junction variables (described in the text). The bottom half of the figure shows a typical design problem.

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 sub-graphs of the desired functional specification.

In Figure 8.3, 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.

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.

$$A \xrightarrow{+} B$$

$$A \xrightarrow{+} x \xrightarrow{+} B$$

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 re-trieved portions from stored cases.

It may also require backtracking on earlier choices of design subgoals and, therefore, rejecting cases that were previously retrieved.

Limitations of CADET:

- 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.

CADET system illustrates several generic properties of case-based reasoning systems that distinguish them from approaches such as k-NEAREST NEIGHBOR:

1. 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.
2. 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. However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.
3. 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. Other systems have been

developed that more fully integrate case-based reasoning into general search- based problem-solving systems.