**Module-5**

**Chapter-1**

**Instance based learning**

Instance-based methods are frequently referred to as “lazy” learning methods because they defer processing until a new instance must be classified.

The training examples are simply stored in the methods. Beyond these examples, generalization is postponed until a new instance has to be classified.

When a new query instance is encountered, its relationship to previously-stored examples is investigated in order to assign a target function value to the new instance.

Nearest neighbor and locally weighted regression approaches are examples of instance-based learning, which presume that instances may be represented as points in a Euclidean space. Case-based reasoning approaches, which use more elaborate, symbolic representations for cases, are also included.

Instead of estimating the target function once for the entire instance space, these approaches can estimate it locally and differently for each new case to be categorized, which is a significant benefit of delayed, or lazy, learning.

Basically, Instance-Based Learning follows the approach of memorizing and learning instead of performing explicit generalization, it compares the new instances with the training data present in the memory.

For example, Any email that looks like spam will be automatically labeled as spam. They aren’t looked into every time.

**Advantage:**

* Instead of estimating the target function for the full instance set, smaller approximations might be made.
* This technique is easily adaptable to new data, which is collected as we go.

**Disadvantage:**

* The costs of classification are substantial.
* Large amounts of memory are required to hold the data, and each query necessitates the creation of a new local model.

The size of the training data determines the time complexity of this technique. This algorithm’s worst-case time complexity is O (n), where n is the number of training cases.

 Approaches to approximating real-valued or discrete-valued target functions using instance-based learning methods such as closest neighbor and locally weighted regression are theoretically simple.

In these algorithms, learning consists of just storing the training data. When a new query instance is encountered, a set of related instances similar to the current query instance is obtained from memory and used to classify it.

 Many strategies merely create a local approximation to the goal function that applies in the immediate vicinity of the new query instance, rather than an approximation that performs well across the entire instance space.

When the target function is exceedingly complicated but can still be described by a set of less complex local approximations, this has substantial advantages.

Some of the Instance-based Learning algorithms are:

* Lazy Learners (KNN algorithm)
* Radial Based Functions (RBF)
* Case-Based Reasoning (CBR)

1. **k- Nearest Neighbor Learning**

The k-NEAREST NEIGHBOR algorithm is the simplest basic instance-based technique. All instances in this algorithm are assumed to correspond to points in n-dimensional space.

The assumption K-NN uses is that the new cases are similar to the previous already trained and classified cases, and hence classifies the new data in the category it is most similar to.

The K-NN method stores all available data and separates the new data point based on its similarity to existing data. This means that new data can be quickly processed into a well-defined phase using the K-NN method.

Although the K-NN method can be used for both regression and classification, it is most typically employed for classification.

The K-NN algorithm is a non-parametric algorithm, meaning it doesn’t make any assumptions about the data.

K-NN is known as a lazy learner algorithm because it doesn’t learn from the training set right away, but it saves the dataset and classifies it when it’s time

During the training phase, the KNN algorithm automatically stores the data, and when it receives new data, it puts it in the exact same category as the new data.

**Example:** Let’s say we have a picture of a creature that looks like a cat or a dog, but we don’t know whether it’s a cat or a dog. We can utilize the KNN method for this identification because it is based on a similarity measure.

Our KNN model will compare the new data set to the cats and dogs photographs and label it as a cat or a dog based on the most similar qualities.

The standard Euclidean distance is used to define an instance’s closest neighbors.

Allow the feature vector to describe any arbitrary instance x.

where signifies the value of instance x’s  property. Then d(xi, xj) is defined as the distance between two instances

The target function in nearest-neighbor learning might be discrete or real-valued

Consider learning discrete-valued target functions of the form https://d1zx6djv3kb1v7.cloudfront.net/wp-content/media/2022/01/k2.jpg, where V denotes the finite collection .

This algorithm’s estimate of is simply the most common value of f among the k training samples closest to x. When k is set to 1, the 1-NEAREST method assigns the value to where is the closest training instance to x. The approach assigns the most common value among the k closest training instances for greater values of k.

**Training Algorithm:**

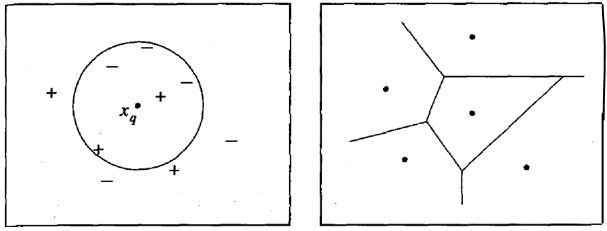
* Add the example to the list of training examples for each training example (x, f(x)).

Classification algorithm:

* Given a classification query instance ,
  + Let’s  say examples of k from training\_examples near .
  + Return,

Where

**kNN and Voroni diagram**



The examples are points in a two-dimensional space with a Boolean value for the target function. “+” and “-” represent good and negative training instances, respectively.

In this image, the 1-NEAREST learning algorithm identifies x as a positive example, while the 5-NEAREST learning method classifies it as a negative example.

The k- 1-Nearest learning technique never generates a general hypothesis about the target function f. It just computes each new query instance’s classification as needed.

The geometry of this decision surface induced by 1-Nearest Neighbor throughout the full instance space is depicted in the diagram on the right side of Figure.

Each of the training examples is surrounded by a convex polyhedron on the decision surface. The polyhedron represents the collection of query points whose categorization will be totally controlled by that training example for each training example.

Outside the polyhedron, query locations are closer to another training example. The Voronoi diagram of the collection of training examples is a common name for this type of diagram.

Approximating continuous-valued target functions is simple with the KNN algorithm. We do this by having the algorithm calculate the mean value of the k closest training samples rather than the most common value. To be more specific, to approximate a target function https://d1zx6djv3kb1v7.cloudfront.net/wp-content/media/2022/01/k5.jpg with an actual value.

**KNN Algorithm Benefits:**

* It is straightforward to implement.
* It can withstand noisy training data.
* If the training data is large, it may be more effective.

**KNN Algorithm Disadvantages:**

* The value of K must constantly be determined, which might be challenging at times.
* Because the distance between data points for all training samples must be calculated, the calculation costs are high.

The amount of the training dataset increases the computational cost of KNN. KNN may be made stochastic for very large training sets by choosing a sample from the training dataset from which to determine the K-most similar occurrences.

KNN has been examined extensively and has been around for a long time. Resulting, many instructions use various terms to explain it, such as:

* **Instance-Based Learning:** Predictions are made using only the raw training instances. As a result, KNN is frequently referred to as case-based learning or instance-based learning (where each training instance is a case from the problem domain).
* **Lazy Learning:** The model does not need to be learned, and all of the work is done when a prediction is needed. As a result, KNN is known as a sluggish learning algorithm.
* KNN is **non-parametric**, which means it makes no assumptions about the problem’s functional form.

**Distance- Weighted Nearest Neighbor Algorithm**

A modified form of k closest neighbors is weighted kNN. The choice of the hyperparameter k is one of the numerous factors that influence the performance of the kNN algorithm.

The method will be more sensitive to outliers if k is too small. If k is set too high, the neighborhood may contain too many points from different classes.

For many practical applications, the distance-weighted k-NN is a highly effective inductive inference approach. When given a sufficiently big collection of training data, it is resilient to noisy training data and extremely successful.

It can smooth out the influence of isolated noisy training samples by calculating the weighted average of the k neighbors closest to the query location.

The inductive bias is the belief that the classification of an instance  will be most similar to the classification of other examples within Euclidean distance.

The contribution of each of the k neighbors is weighted according to their distance from the query point  in the Nearest Neighbor algorithm, with closer neighbors receiving more weight.

* This may be performed by changing the algorithm’s last line with,

To accommodate he case where the query point exactly matches one of the training instances is therefore zero we assign to be in this case. If there are several such training examples we assign majority classification among them.

* for target functions with real values,

If for all training cases, then ← c

The sole drawback of taking into account all samples is that our classifier will run slower. While all training instances are taken into account when categorizing a new query instance, the algorithm is referred to as a global method.

We call it a local technique if we just consider the training examples that are closest to us. Shepard’s method is the name given to a rule that is applied as a global method to all training samples.

**CURSE of Dimensionality:**

The difficulty produced by the exponential rise in volume associated with adding more dimensions to Euclidean space is known as the curse of dimensionality.

The curse of dimensionality states that as the number of characteristics grows, the error grows as well. It refers to the fact that high-dimensional algorithms are more difficult to build and typically have a running duration that is proportional to the dimensions.

A larger number of dimensions theoretically allows for more information to be stored, but in practice, it seldom helps since real-world data contains more noise and redundancy.

Consider a k-NN situation in which each instance is defined by 20 qualities, but only two of them are essential to deciding the classification for the specific target function.

In this situation, instances with identical values for the two important properties might nonetheless be far apart in the 20-dimensional instance space. As a result, the k-NEAREST NEIGHBOR similarity score, which is based on all 20 traits, will be deceptive.

The vast amount of irrelevant qualities will influence the distance between neighbors. As the curse of dimensionality refers to the problem that comes when a large number of unnecessary qualities are present. Nearest-neighbor techniques are particularly vulnerable to this issue.

**Why is it difficult to analyze high-dimensional data?**

The combination of two factors causes difficulty in analyzing high-dimensional data.

* High-dimensional spaces exhibit geometrical qualities that are counterintuitive and different from those seen in two- or three-dimensional spaces.
* Data analysis tools are frequently created with understandable features and examples in low-dimensional spaces in mind, and they are typically best explained in 2- or 3-dimensional environments.

The problem is that those tools are also utilized when data is high-dimensional and complicated, which increases the risk of losing intuition about the tool’s behavior and drawing inaccurate conclusions.

**How to Overcome the Dimensionality Curse**

**1. Weigh each property differently:**

When determining the distance between two instances, one intriguing technique to solving this problem is to weigh each property differently.

Stretching the axes in Euclidean space, shortening the axes that correspond to less significant traits, and extending the axes that belong to more relevant attributes is equivalent to this.

A cross-validation technique may be used to automatically calculate the amount by which each axis should be extended.

To demonstrate how, consider that we want to stretch (multiply) the axis by some factor with the values selected to minimize the learning algorithm’s real classification error. Second, cross-validation may be used to estimate this genuine error.

As a result, one approach is to choose a random portion of the available data as training examples, then calculate the values of that results in the least amount of error when categorizing the remaining cases.

The estimate for these weighting factors can be improved by repeating this approach several times. Stretching the axes to improve the performance of k-NN gives a way for reducing the influence of irrelevant qualities.

**2. Remove the attributes that aren’t important**

An even more severe option is to remove the least relevant properties from the instance space entirely. This is the same as turning off some of the zi scaling factors.

Moore and Lee (1994) demonstrate how to identify meaningful subsets of the characteristics using efficient cross-validation methods.

In all feasible approaches, investigate strategies based on leave-one-out cross-validation, in which a collection of m training examples is regularly divided into a training set of size m – 1 and a test set of size 1.

Because no extra training effort is required each time the training set is redefined, this leave-one-out strategy is simple to apply in k-NEAREST NEIGHBOR algorithms.

Stretch each axis by a different amount over the instance space. However, as the number of degrees of freedom available to the algorithm for redefining its distance measure in this way grows, the danger of overfitting grows as well. As a result, expanding the axes locally is a considerably less usual strategy.

1. **Locally Weighted Regression**

Nearest-neighbor techniques previously discusses may be regarded as approximating the goal function f (x) at the single query point .

A generalization of this method is locally weighted regression. Over a limited region surrounding , it creates an explicit approximation to f. To construct this local approximation to f, locally weighted regression utilizes close or distance-weighted training instances.

We may use a linear function, a quadratic function, a multilayer neural network, or any functional form to approximate the goal function in the vicinity of x.

The term “locally weighted regression” comes from the fact that the function is approximated using only data near the query point, weighted because each training example’s contribution is weighted by its distance from the query point, and regression because this is the term widely used in the statistical learning community to describe the problem of approximating real-valued functions.

* **Locally weighted linear regression:**

The main method in locally weighted regression is to generate an approximation f that matches the training instances in the neighborhood surrounding x, given a new query instance x.

This approximation is then used to produce the value f”(x,), which is returned as the query instance’s estimated target value.

Because a separate local approximation will be produced for each individual query instance, the description of f can be removed.

Consider the example of locally weighted regression, which uses a linear function of the type to approximate the target function f near x.

The value of the ith attribute of the instance x is denoted by

Using a global approximation to the target function, discover the coefficients wo…w to minimize the error in fitting such linear functions to a given collection of training instances.

As a result, we developed techniques for selecting weights that minimize the total squared error over the set D of training instances.

Gradient descent training rule,

where  *is a constant learning rate*

* Local Approximation:
  1. Minimize the squared error over just the k nearest neighbors:
  2. Minimize the squared error over the entire set of D training examples, while weighting the error of each training example by some decreasing function K of its distance from
  3. Combine 1 and 2:

The distance penalty is now increased by the contribution of instance x to the weight update, and the error is now averaged across just the k closest training examples.

In reality, if we’re fitting a linear function to a given number of training samples, there exist methods for directly solving for the required coefficients that are far more efficient than gradient descent…

Because

(1) the cost of fitting more complicated functions for each query instance is prohibitively costly, and

(2) these basic approximations mimic the target function very effectively across a sufficiently small subregion of the instance space, more complex functional forms are rarely identified.

1. **Radial Basis Function**

Radial basis functions are a function approximation method that is strongly connected to distance-weighted regression and artificial neural networks.

This is the learned hypothesis function,

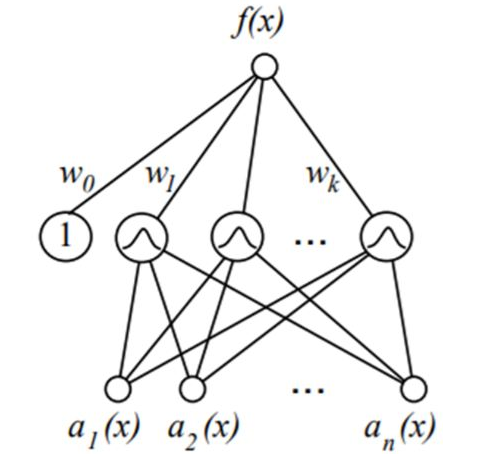
If each is an X instance and the kernel K function, is defined to decrease as increases. The user-supplied constant k specifies the number of kernel functions to be included.

The contribution from each of the elements is confined to an area around the point  in , which is a global approximation to f (x).

It is normal to choose each function ku(d(xu,x)) individually to be a Gaussian function centered at the point xu with some variance.

The function may be thought of as defining a two-layer network, with the first layer computing the values of the different and the second layer computing a linear combination of these first-layer units values.

* A Gaussian function centered at some instance determines the activation of each hidden unit. As a result, unless the input x is close to xu, its activation will be close to zero.
* The hidden unit activations are combined in a linear fashion by the output unit. Although the network illustrated here only has one output, it is possible to incorporate numerous output units.



Where are the attributes describing instance x, and

One common choice for is

RBF networks are generally trained in two stages, given a collection of target function training samples.

The number of hidden units k is chosen first, and each hidden unit u is specified by the values of and  that form its kernel function

Second, using the global error criterion, the weights are trained to maximize the network’s fit to the training data. The linear weight values wu can be learned relatively efficiently since the kernel functions are kept constant throughout this second step.

One method is to assign a Gaussian kernel function to each training example and center it at the point xi. The width  for each of these kernels might be the same.

The RBF network learns a global approximation to the target function using this strategy, in which each training sample may only impact the value of in the region of

This choice of kernel functions has the advantage of allowing the RBF network to precisely fit the training data. That is, the weights for merging the m Gaussian kernel functions may be adjusted so that  for each training example for any collection of m training instances.

Another option is to select a smaller set of kernel functions than the number of training instances. This method is far more efficient than the first, especially when dealing with a large number of training samples.

The collection of kernel functions might be dispersed throughout the instance space X, with centers evenly separated. Alternatively, we may want to distribute the facilities consistently, especially if the conditions are still distributed more than X.

In this scenario, we may choose kernel function centers by sampling the underlying distribution of examples by randomly picking a portion of the training instances.

Alternatively, we may find archetypal clusters of instances and then apply a kernel function to each one. Unsupervised clustering algorithms that fit the training instances (but not their target values) to a mixture of Gaussians can be used to place the kernel functions in this manner.

The EM algorithm is a method for selecting the means of a group of k Gaussians that best fits the observed data. Given the k estimated means, the means in the EM method are chosen to maximize the likelihood of witnessing the occurrences xi.

 Unsupervised clustering algorithms do not take the instance’s target function value into account while calculating kernel centers. The output layer weights , are determined only by the goal values in this example.

Function with a radial basis A global approximation to the target function is provided by networks, which are represented by a linear combination of several local kernel functions.

Only when the input x falls inside the zone indicated by the kernel function’s specific center and width is the value for that kernel function non-negligible.

As a result, the network may be thought of as a smooth linear combination of a number of local approximations to the goal function. RBF networks have the advantage of being much more efficient to train than feed forward networks trained with BACKPROPAGATION.

This is due to the fact that the input and output layers of an RBF are trained separately.

**Training RBF Networks:**

* For the kernel function , which should be used?
  + Use training instances or scatter uniformly throughout instance space (reflects instance distribution)
* What is the best way to train weights (assuming Gaussian )?
  + First, pick a variance (and maybe a mean) for each — for example, use EM.
  + After that, keep constant and train the linear output layer – a quick way to fit linear functions.

1. **Case-Based Reasoning:**

Case-based reasoning has been used to solve difficult scheduling problems by reusing relevant pieces of previously solved problems at a help desk, reasoning about legal matters by referring to previous instances, and storing and reusing past experience at a help desk.

One downside of instance-based techniques is that it can be expensive to classify new instances. This is because practically all computation occurs during categorization rather than when the training samples are encountered for the first time.

As a result, strategies for efficiently indexing training samples constitute a significant practical concern in lowering query time computation.

The second shortcoming of many instance-based techniques, particularly nearest neighbor algorithms, is that when retrieving similar training examples from memory, they often examine all attributes of the instances.

If only a few of the numerous available attributes are used to define the target idea, the instances that are actually “similar” may be a long way apart.

##### **How does it work?**

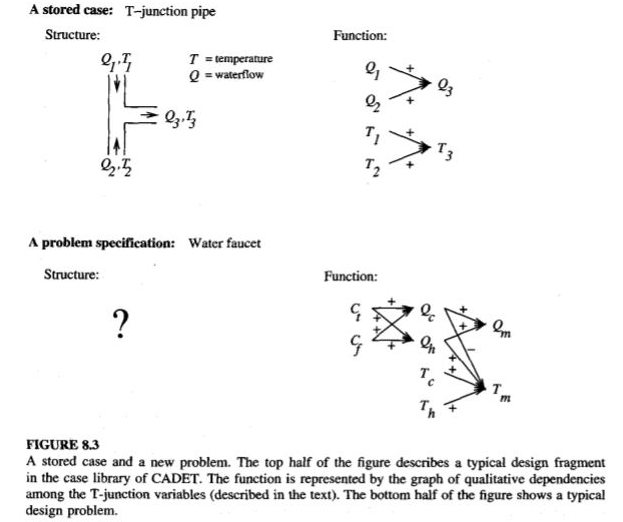
An Instance-based Reasoner (CBR) will first check if an identical training case exists before classifying a new case. If one is identified, the case’s solution is returned along with it.

If no equivalent example can be discovered, the CBR will look for training cases that have components that are comparable to the new cases. These training instances can be thought of as neighbors to the new case in terms of concept.

If the cases are represented as graphs, this entails looking for subgraphs that are similar to those in the new example. To provide a solution for the new example, the CBR attempts to merge the solutions of neighboring training cases.

If there are problems with the particular solutions, you may need to go back and look for other options. To propose a viable solution, the CBR may use prior information and problem-solving methodologies.

Let’s look at an example of case-based reasoning,

* Case-based reasoning is used in the CADET system (Sycara et al. 1992) to aid in the conceptual design of simple mechanical devices such as water faucets.
* It suggests conceptual ideas to satisfy the specifications of new design issues using a library of around 75 prior designs and design fragments. Each memory instance (for example, a water pipe) is described by specifying both its structure and qualitative function.
* By providing the intended function and demanding the associated structure, new design difficulties are provided. Figure 8.3 shows how to build up a situation like this.
* The top half of the diagram depicts a T-junction pipe, which is a common type of storage case. Its function is illustrated by the qualitative correlations between the amounts of water flow and the temperatures at its inputs and outputs.
* An arrow with a “+” label in the functional description to its right indicates that the variable at the arrowhead rises with the variable at its tail. For example, as the input water flow grows, the output water flows Q3 rises.
* 

**Chapter-2**

**Genetic Algorithms**

1. **Motivation:**

Genetic algorithms are preferred for solving optimization problems as they can yield a result in optimum time and is also relatively faster.

The need for the genetic algorithm is as follows:

**1. Gradient base model failures**

In the gradient base method, traditional calculus is used. It starts at a random point and moves in the direction of the gradient, till it reaches the top point. Though this method is effective for single peaked objective functions such as the cost function in linear regression, it cannot be very useful in real life where the problems are more complex such as landscapes, they are made of many peaks and valleys leading to the failure of these models. They will ultimately get stuck at the local optima.

**2. Solving difficult problems with ease.**

In computer science, we have many problems that even the most powerful computers take ages to compute. In this situation, a generic algorithm comes handy as it can give approximated solutions in optimal time.

**3. Time-efficient.**

Many problems such as the Travelling Salesperson Problem or TSP has many practical uses such as path finding and VLSI design. GA can give efficient results in optimum time. For example, how much of a trouble it would be if our GPS took hours to give us the path from our source to destination. But with GA involved we can get a good result in a small time.

**Genotype representing**

Genotype representation is very important during the implementation of a genetic algorithm, as it directly impacts the performance of the genetic algorithm. A proper representation is where the phenotype and the genotype mappings have proper spaces between them.

The most common representation methods for GA are as follows:

**1. Binary representation**

Binary representation is one of the most common methods of representing GA.

The genotype in this method consists of bit strings. In the case of a Knapsack problem, where the solution space consists of Boolean decision variables, the binary representation is natural.

The genotype in this method consists of bit strings. In the case of a Knapsack problem, where the solution space consists of Boolean decision variables, the binary representation is natural.

For other problems, which may deal with numbers, we represent the numbers with their binary representation. The only drawback in this situation is that different bits have different meanings, which results in undesired consequences for the mutation and crossover operators.

**2. Integer representation**

In the case of discrete-valued genes, we cannot always limit the results to binary, so instead, we use integer representation. For example,  if we had to encode the three directions, we could have encoded them as: {1,2,3,4}, and represented using integer representation.

**3. Permutation representation**

Whenever the solutions are represented by an order of elements we can use permutation representation. We can take the same example of TSP, let us assume the person has to travel all the cities, visiting one city at a time, and then comes back to the source city. As we can see the order of the TSP is naturally becoming a permutation, therefore we can use the permutation representation.

**4. Real valued representation**

In the problems where we require to define the genes using continuous variables, we use real-valued representation.

1. **Genetic algorithm-Hypotheses**

Hypotheses in the genetic algorithm are defined by bit strings whose interpretation depends on the application. To find the appropriate hypotheses, we need to begin with a population of the initial hypotheses. Operations such as random mutation and crossover help the current population to give rise to the next generation. The best hypotheses in the genetic algorithm are defined on the basis of fitness, it is the one that optimizes predefined numerical measures for the problem at hand. It is known as the hypotheses fitness.

**What is the population?**

Subsets in the solution of the current generation are known as population. Some things we should keep in mind while discussing population in the genetic algorithm are as follows:

1. If the diversity of the population is not satisfied, it may lead to premature convergence.
2. If the population size is too small, then a satisfactory mating pool might not be available, while a larger population can slow down the genetic algorithm. Therefore, the population size should be set in an optimal way to avoid these situations.

There are two methods of population initialization:

**1. Random initialization**

The initial population is populated with completely random solutions.

**2. Heuristic initialization**

A known heuristic is used to populate the initial population

**Representation of hypotheses**

Hypotheses are represented using bit strings so that the manipulation by genetic operators is easier.

It is further explained through the following example:

Consider an outlook,that can have any of the following three values: sunny, overcast, or rain.

Then let the string 010 represents (Outlook = Overcast)

String 011 represents (Outlook = overcast or rain)

String 111 represents the general constraint.

* **Genetic operators**

**1. Crossover operator**

The crossover operator copies selected bits from the parent to produce two new offspring from the two-parent string.

An additional string known as the crossover mask helps predict the choice of parent responsible for the bit position i, this bit position i is copied from one parent and used as the bit position for the offspring.

The main types of crossover operators are as follows:

* **Single point crossover**

Single point crossover constructs the crossover mask in such a way that it starts with strings containing n contiguous 1s, which is followed by certain numbers of zeros to complete the string.

* **Two-point crossover**

Two-point crossover creates offsprings by substituting intermediate segments of the parent into the middle of the second parent string, implying two offsprings are created by two parents by switching the roles.

* **Uniform crossover**

It combines bit samples uniformly from the parent to create the offspring. The crossover mask in the uniform crossover is generated as a random bit string that is independent of others.

**2. Mutation**

Mutation uses only one single parent to produce offspring. The mutation operator chooses one single bit at a time to make small random changes to the bit string and is mostly performed after the crossover is already applied.

* **Fitness Function and selection**

The fitness function can be defined as a particular solution to a particular problem through corresponding input and produces output as to how good the solution is with respect to the given problem.

The fitness value calculation is done repeatedly in a genetic algorithm and that is why it must be fast enough. Slow problem-solving computation can have an effect on the genetic algorithm and can make it too slow to execute.

In some cases, the genetic fitness function and function objects are nearly equal or the same because the function objective is to either minimize or maximize some complex computation problems with multiple constraints. A designer algorithm must be used to have a different fitness function.

The following characteristics must be produced by the fitness function :

1. It must be fast to easily solve or compute complex problems.
2. It must measure in quantity how good or fit the function provided or how much good can it provide individuals from the particular solution.

Some cases may happen that a fitness function cannot calculate the given complex problem directly thereafter approximation of fitness function may be used to get with the correct solution, and that problem may occur because of inherit in complexities of the problem.

**USES OF GENETIC FITNESS FUNCTIONS:**

1. We test with complex problems and come up with the best case solution.
2. Each solution is given a particular approximation to check whether it is near the original value or not.

**REQUIREMENTS OF A FITNESS FUNCTION:**

1. It must be clearly defined so that it can be understood how efficiently the fitness is being calculated.
2. It must be efficiently produced executed or implemented. If the function becomes the congestion of the algorithm then the efficiency of the overall algorithm may be lowered.
3. The particular fitness function must produce an approximated solution.

#### TO OVERCOME A PARTICULAR FITNESS FUNCTION FOR A GIVEN PROBLEM

 Every particular problem has its own fitness function. There is no particular rule to use a fitness function. However, a few methods are being adopted by scientists that deal with data.

For optimization of a particular complex problem, functions that are basic must be used such as the total set of calculations of parameters that are directly associated with the complexity of the problem and its domain can be used to solve in a fitness function.

A fitness function must be such a kind of function that can measure how fast or better it can produce solutions for a given set of problems. For example, suppose a fitness function is almost zero unless a function produces it is correct or not is not better, since it does not provide the data how approximate is the solution to the correct answer.

Similarly, a fitness function that is increasing in nature will give solutions better but it would not identify or reallocate the solution which is best is not approximately close enough to the actual answer.

Thus, the fitness function is also known as the evaluation function which evaluates how close or approximate it is or how nearly it is to the original solution.

1. **Hypothesis Space Search**

It is clear that genetic algorithms employ a randomized beam search method to seek maximally fit hypotheses. In the hypothesis space search method, we can see that the gradient descent search in backpropagation moves smoothly from one hypothesis to another. On the other hand, the genetic algorithm search can move much more abruptly. It replaces the parent hypotheses with an offspring that can be very different from the parent. Due to this reason, genetic algorithm search has lower chances of it falling into the same kind of local minima that plaques the gradient descent methods.

There is one practical difficulty that is often encountered in genetic algorithms, it is crowding. Crowding can be defined as the phenomenon in which some individuals that are more fit in comparison to others, reproduce quickly, therefore the copies of this individual take over a larger fraction of the population. Most of the strategies used in the genetic algorithms are inspired by biological evolution. One such other strategy used is fitness sharing, in which the measured fitness of an individual is decreased by the presence of another individual of a similar kind. The third method is to restrict all the individuals to combine to form offspring. To better understand we can say that by allowing individuals of the same kind to recombine, clusters of similar individuals are formed, forming multiple subspecies in the population.

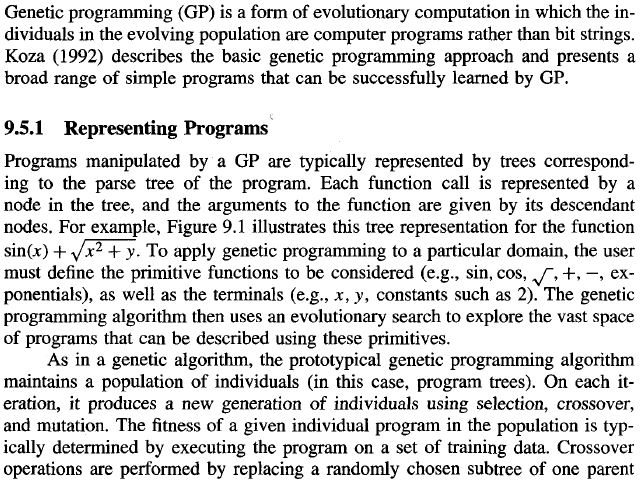
Another method would be to spatially distribute individuals and allow only nearby individuals to combine.

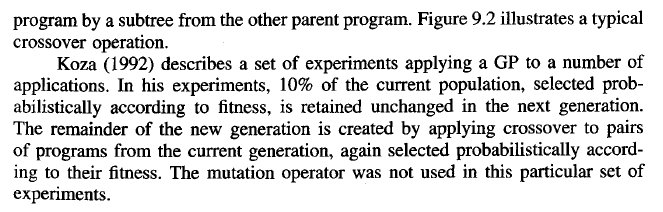
**Population evolution and schema theorem.**

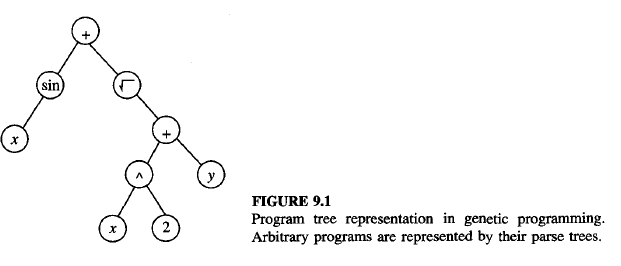
The schema theorem of Holland is used to mathematically characterize the evolution over time of the population with respect to time. It is based on the concept of schema. So, what is schema? Schema is any string composed of 0s, and 1s, and \*s, where \* represents null, so a schema 0\*10, is the same as 0010 and 0110. The schema theorem characterizes the evolution within a genetic algorithm on the basis of the number of instances representing each schema. Let us assume the m(s, t) to denote the number of instances of schema denoted by ‘s’, in the population at the time ‘t’, the expected value in the schema theorem is described as m(s, t+1), in terms of m(s, t), and the other parameters of the population, schema, and GA.

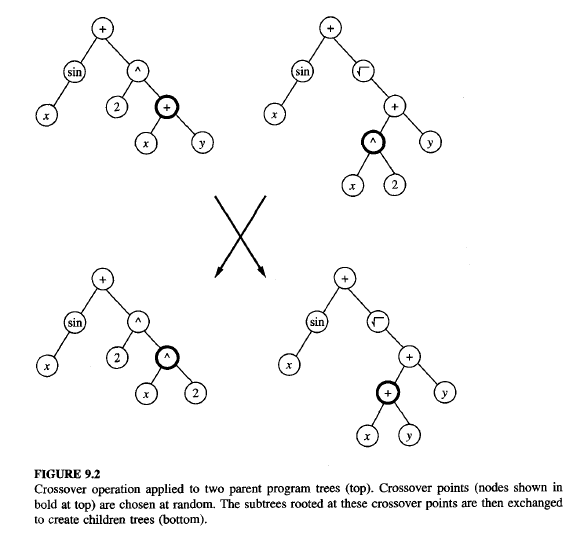
In a genetic algorithm, the evolution of the population depends on the selection step, the recombination step, and the mutation step. The schema theorem is one of the most widely used theorems in the characterization of population evolution within a genetic algorithm. If it fails to consider the positive effects of crossover and mutation, it is in a way incomplete. There are many other recent theoretical analyses that have been proposed, many of these analogies are based on models such as Markov chain models and the statistical mechanical model.

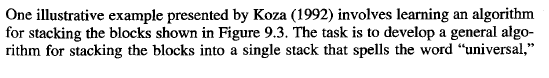
1. **Genetic Programming**

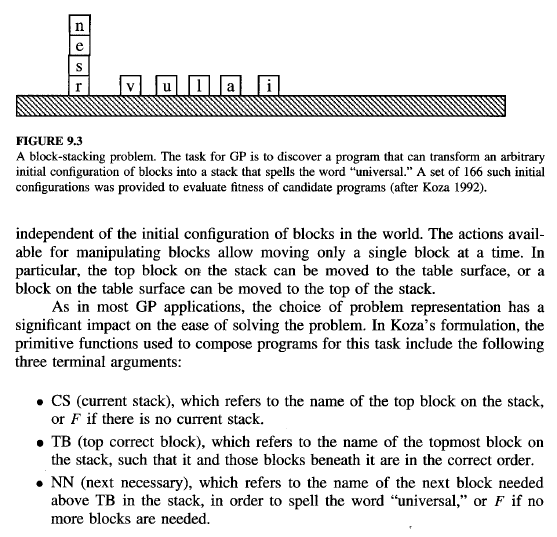
****

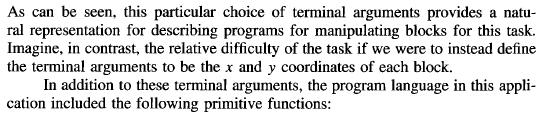
****

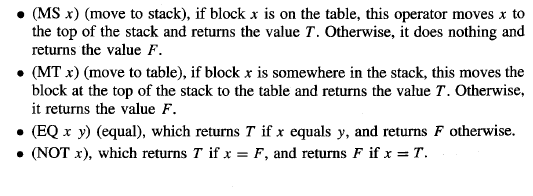
****

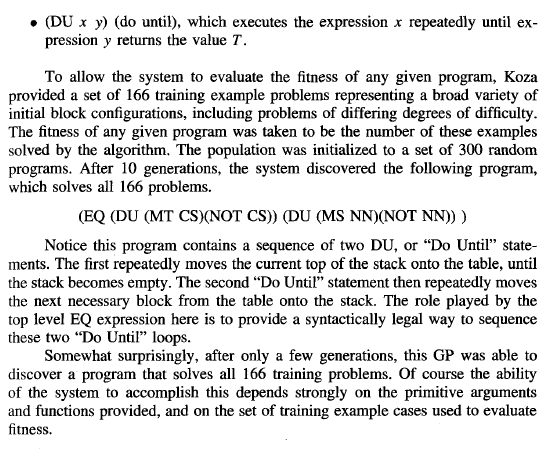
****

****





****

****