UNIT I
Introduction to Machine Learning

1. Introduction

1.1 What Is Machine Learning?

Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be *predictive* to make predictions in the future, or *descriptive* to gain knowledge from data, or both.

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. He defined machine learning as "the field of study that gives computers the ability to learn without being explicitly programmed." However, there is no universally accepted definition for machine learning. Different authors define the term differently.

Definition of learning

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Examples

i) Handwriting recognition learning problem
- Task T: Recognising and classifying handwritten words within images
- Performance P: Percent of words correctly classified
- Training experience E: A dataset of handwritten words with given classifications

ii) A robot driving learning problem
- Task T: Driving on highways using vision sensors
- Performance measure P: Average distance traveled before an error
- training experience: A sequence of images and steering commands recorded while observing a human driver

iii) A chess learning problem
- Task T: Playing chess
- Performance measure P: Percent of games won against opponents
- Training experience E: Playing practice games against itself

Definition

A computer program which learns from experience is called a machine learning program or simply a learning program. Such a program is sometimes also referred to as a learner.

1.2 Components of Learning

Basic components of learning process

The learning process, whether by a human or a machine, can be divided into four components, namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the variouscomponents and the steps involved in the learning process.

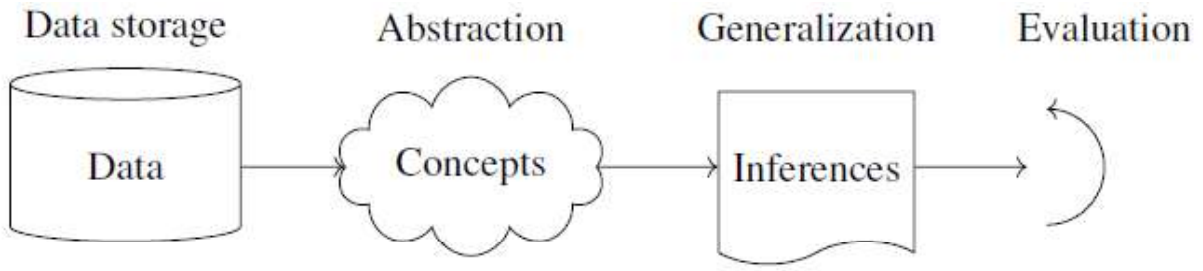


Figure 1.1: Components of learning process

1. Data storage

Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

• In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.

• Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

2. Abstraction

The second component of the learning process is known as abstraction.

Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models.

The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

3. Generalization

The third component of the learning process is known as generalisation.

The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

4. Evaluation

Evaluation is the last component of the learning process.

It is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then utilised to effect improvements in the whole learning process

Applications of machine learning

Application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having

high predictive accuracy.

The following is a list of some of the typical applications of machine learning.

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.

4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon and Go.

1.3 Learning Models

Machine learning is concerned with using the right features to build the right models that achieve the right tasks. The basic idea of Learning models has divided into three categories.
For a given problem, the collection of all possible outcomes represents the **sample space or instance space**.

- Using a Logical expression. (**Logical models**)
- Using the Geometry of the instance space. (**Geometric models)**
- Using Probability to classify the instance space. (**Probabilistic models**)
- Grouping and Grading

1.3.1 Logical models

Logical models use a logical expression to divide the instance space into segments and hence construct grouping models. A **logical expression** is an expression that returns a Boolean value, i.e., a True or False outcome. Once the data is grouped using a logical expression, the data is divided into homogeneous groupings for the problem we are trying to solve. For example, for a classification problem, all the instances in the group belong to one class.

There are mainly two kinds of logical models: **Tree models** and **Rule models**.

Rule models consist of a collection of implications or IF-THEN rules. For tree-based models, the 'if-part' defines a segment and the 'then-part' defines the behaviour of the model for this segment. Rule models follow the same reasoning.

Logical models and Concept learning

To understand logical models further, we need to understand the idea of **Concept Learning**. Concept Learning involves learning logical expressions or concepts from examples. The idea of Concept Learning fits in well with the idea of Machine learning, i.e., inferring a general function from specific training examples. Concept learning forms the basis of both tree-based and rule-based models. More formally, Concept Learning involves acquiring the definition of a general category from a given set of positive and negative training examples of the category. A Formal Definition for Concept Learning is "***The inferring of a Boolean-valued function from training examples of its input and output."*** In concept learning, we only learn a description for the positive class and label everything that doesn't satisfy that description as negative.

The following example explains this idea in more detail.

A Concept Learning Task – Enjoy Sport
Training Examples

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	YES
2	Sunny	Warm	High	Strong	Warm	Same	YES
3	Rainy	Cold	High	Strong	Warm	Change	NO
4	Sunny	Warm	High	Strong	Warm	Change	YES

ATTRIBUTES CONCEPT

A Concept Learning Task called "Enjoy Sport" as shown above is defined by a set of data from some example days. Each data is described by six attributes. The task is to learn to predict the value of Enjoy Sport for an arbitrary day based on the values of its attribute values. The problem can be represented by a **series of hypotheses**. Each hypothesis is described by a conjunction of constraints on the attributes. The training data represents a set of positive and negative examples of the target function. In the example above, each hypothesis is a vector of six constraints, specifying the values of the six attributes – Sky, AirTemp, Humidity, Wind, Water, and Forecast. The training phase involves learning the set of days (as a conjunction of attributes) for which Enjoy Sport = yes.

Thus, the problem can be formulated as:

- Given instances X which represent a set of all possible days, each described by the attributes:
  - Sky – (values: Sunny, Cloudy, Rainy),
  - AirTemp – (values: Warm, Cold),
  - Humidity – (values: Normal, High),
  - Wind – (values: Strong, Weak),
  - Water – (values: Warm, Cold),
  - Forecast – (values: Same, Change).

Try to identify a function that can predict the target variable Enjoy Sport as yes/no, i.e., 1 or 0.

1.3.2 Geometric models

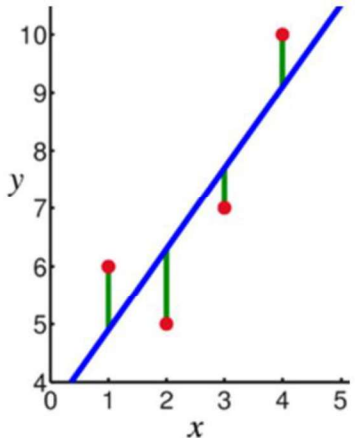
In the previous section, we have seen that with logical models, such as decision trees, a logical expression is used to partition the instance space. Two instances are similar when they end up in the same logical segment. In this section, we consider models that define similarity by considering the geometry of the instance space. In Geometric models, features could be described as points in two dimensions (*x*- and *y*-axis) or a three-dimensional space (*x*, *y,* and *z*). Even when features are not

intrinsically geometric, they could be modelled in a geometric manner (for example, temperature as a function of time can be modelled in two axes). In geometric models, there are two ways we could impose similarity.

- We could use geometric concepts like **lines or planes to segment (classify)** the instance space. These are called **Linear models**.
- Alternatively, we can use the geometric notion of distance to represent similarity. In this case, if two points are close together, they have similar values for features and thus can be classed as similar. We call such models as **Distance-based models**.

Linear models

Linear models are relatively simple. In this case, the function is represented as a linear combination of its inputs. Thus, if x_1 and x_2 are two scalars or vectors of the same dimension and a and b are arbitrary scalars, then $ax_1 + bx_2$ represents a linear combination of x_1 and x_2. In the simplest case where $f(x)$ represents a straight line, we have an equation of the form $f(x) = mx + c$ where c represents the intercept and m represents the slope.



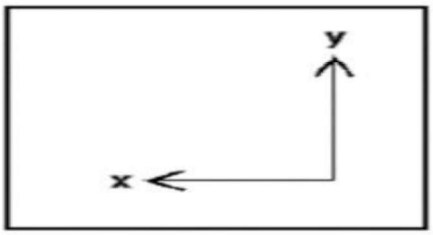
Linear models are **parametric**, which means that they have a fixed form with a small number of numeric parameters that need to be learned from data. For example, in $f(x) = mx + c$, m and c are the parameters that we are trying to learn from the data. This technique is different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.

Linear models are **stable**, i.e., small variations in the training data have only a limited impact on the learned model. In contrast, **tree models tend to vary more with the training data**, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well. As a result of having relatively few parameters, Linear models have **low variance and high bias**. This implies that **Linear models are less likely to overfit the training data** than some other models. However, they are more likely to underfit. For example, if we want to learn the boundaries between countries based on labelled data, then linear models are not likely to give a good approximation.

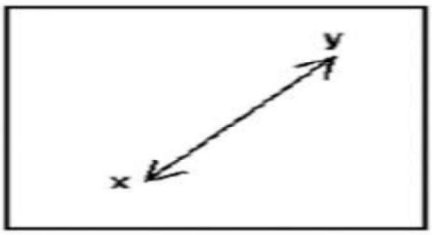
Distance-based models

Distance-based models are the second class of Geometric models. Like Linear models, distance-based models are based on the geometry of data. As the name implies, distance-based models work on the concept of distance. In the context of Machine learning, the concept of distance is not based on merely the physical distance between two points. Instead, we could think of the distance between two points considering the **mode of transport** between two points. Travelling between two cities by plane

covers less distance physically than by train because a plane is unrestricted. Similarly, in chess, the concept of distance depends on the piece used – for example, a Bishop can move diagonally. Thus, depending on the entity and the mode of travel, the concept of distance can be experienced differently. The distance metrics commonly used are **Euclidean**, **Minkowski**, **Manhattan**, and **Mahalanobis**.



Manhattan



Euclidean

Distance is applied through the concept of **neighbours and exemplars**. Neighbours are points in proximity with respect to the distance measure expressed through exemplars. Exemplars are either **centroids** that find a centre of mass according to a chosen distance metric or **medoids** that find the most centrally located data point. The most commonly used centroid is the arithmetic mean, which minimises squared Euclidean distance to all other points.

Notes:

- The **centroid** represents the geometric centre of a plane figure, i.e., the arithmetic mean position of all the points in the figure from the centroid point. This definition extends to any object in *n*-dimensional space: its centroid is the mean position of all the points.
- **Medoids** are similar in concept to means or centroids. Medoids are most commonly used on data when a mean or centroid cannot be defined. They are used in contexts where the centroid is not representative of the dataset, such as in image data.

Examples of distance-based models include the **nearest-neighbour** models, which use the training data as exemplars – for example, in classification. The **K-means clustering** algorithm also uses exemplars to create clusters of similar data points.

1.3.3 Probabilistic models

The third family of machine learning algorithms is the probabilistic models. We have seen before that the k-nearest neighbour algorithm uses the idea of distance (e.g., Euclidian distance) to classify entities, and logical models use a logical expression to partition the instance space. In this section, we see how the **probabilistic models use the idea of probability to classify new entities.**

Probabilistic models see features and target variables as random variables. The process of modelling represents and **manipulates the level of uncertainty** with respect to these variables. There are two types of probabilistic models: **Predictive and Generative**. Predictive probability models use the idea of a **conditional probability** distribution $P(Y|X)$ from which Y can be predicted from X. Generative models estimate the **joint distribution** $P(Y, X)$. Once we know the joint distribution for the generative models, we can derive any conditional or marginal distribution involving the same variables. Thus, the generative model is capable of creating new data points and their labels, knowing the joint probability distribution. The joint distribution looks for a relationship between two variables. Once this relationship is inferred, it is possible to infer new data points.
Naïve Bayes is an example of a probabilistic classifier.

We can do this using the **Bayes rule** defined as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

The Naïve Bayes algorithm is based on the idea of **Conditional Probability. Conditional probability is based on finding the** probability that something will happen, *given that something else* has already happened. The task of the algorithm then is to look at the evidence and to determine the likelihood of a specific class and assign a label accordingly to each entity.

Some broad categories of models:

Geometric models	Probabilistic models	Logical models
E.g. K-nearest neighbors, linear regression, support vector machine, logistic regression, …	Naïve Bayes, Gaussian process regression, conditional random field, …	Decision tree, random forest, …

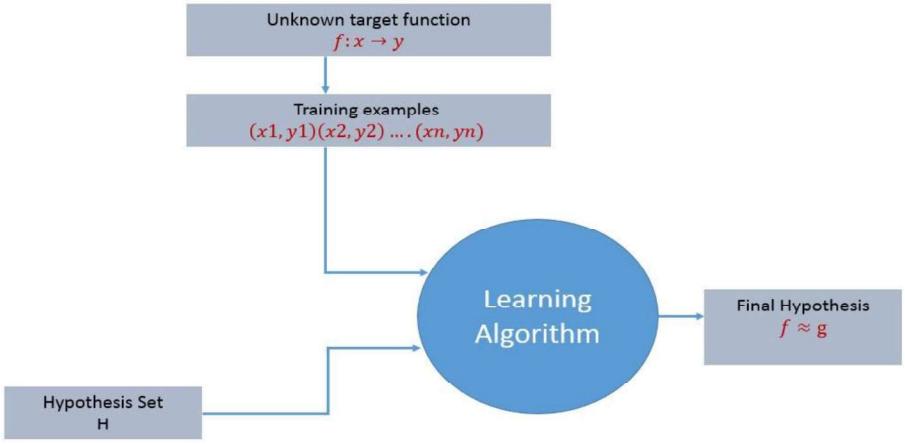
1.3.4 Grouping and Grading

Grading vs grouping is an orthogonal categorization to geometric-probabilistic-logical-compositional.

- Grouping models break the instance space up into groups or segments and in each segment apply a very simple method (such as majority class).
    - E.g. decision tree, KNN.
- Grading models form one global model over the instance space.
    - E.g. Linear classifiers – Neural networks

1.4 Designing a Learning System

For any learning system, we must be knowing the three elements — **T (Task)**, **P (Performance Measure)**, and **E (Training Experience)**. At a high level, the process of learning system looks as below.



The learning process starts with task T, performance measure P and training experience E and objective are to find an unknown target function. The target function is an exact knowledge to be learned from the training experience and its unknown. For example, in a case of credit approval, the learning system will have customer application records as experience and task would be to classify whether the given customer application is eligible for a loan. So in this case, the training examples can be represented as

(x1,y1)(x2,y2)..(xn,yn) where X represents customer application details and y represents the status of credit approval.

With these details, what is that exact knowledge to be learned from the training experience?

So the target function to be learned in the credit approval learning system is a mapping function f:X →y. This function represents the exact knowledge defining the relationship between input variable X and output variable y.

Design of a learning system

Just now we looked into the learning process and also understood the goal of the learning. When we want to design a learning system that follows the learning process, we need to consider a few design choices. The design choices will be to decide the following key components:

1. **Type of training experience**
2. **Choosing the Target Function**
3. **Choosing a representation for the Target Function**
4. **Choosing an approximation algorithm for the Target Function**
5. **The final Design**

We will look into the game - checkers learning problem and apply the above design choices. For a checkers learning problem, the three elements will be,

1. Task T: To play checkers
2. Performance measure P: Total percent of the game won in the tournament.
3. Training experience E: A set of games played against itself

1.4.1 Type of training experience

During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

1. **Direct or Indirect training experience —** In the case of direct training experience, an individual board states and correct move for each board state are given.
   In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games. How to assign credit or blame to individual moves is the credit assignment problem.

2. **Teacher or Not —** Supervised — The training experience will be labeled, which means, all the board states will be labeled with the correct move. So the learning takes place in the presence of a supervisor or a teacher.
   Unsupervised — The training experience will be unlabeled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervision or teacher involvement.

Semi-supervised — Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing.

3. **Is the training experience good —** Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.

The checker player learns by playing against oneself. Its experience is indirect. It may not encounter moves that are common in human expert play. Once the proper training experience is available, the next design step will be choosing the Target Function.

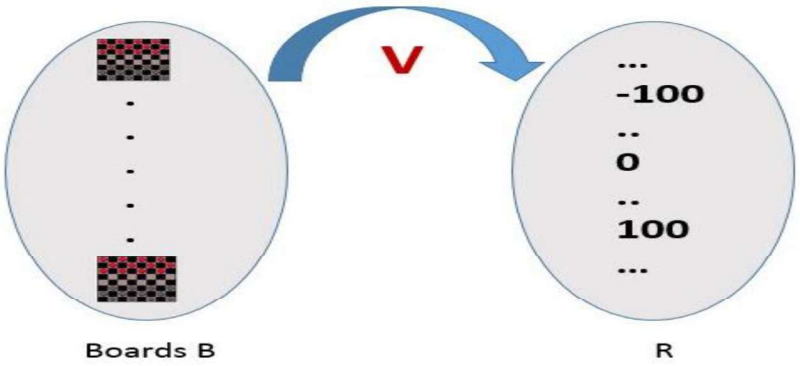
1.4.2 Choosing the Target Function

When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different possibilities. You think and apply the learning that you have gained from the experience. Here the learning is, for a specific board, you move a checker such that your board state tends towards the winning situation. Now the same learning has to be defined in terms of the target function.

Here there are 2 considerations — direct and indirect experience.

- **During the direct experience**, the checkers learning system, it needs only to learn how to choose the best move among some large search space. We need to find a target function that will help us choose the best move among alternatives. Let us call this function ChooseMove and use the notation **ChooseMove : B →M** to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
- **When there is an indirect experience**, it becomes difficult to learn such function. How about assigning a real score to the board state.

So the function be **V : B →R** indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states.



If the system can successfully learn such a target function V, then it can easily use it to select the best move from any board position.

Let us therefore define the target value V(b) for an arbitrary board state b in B, as follows:
1. if b is a final board state that is won, then V(b) = 100
2. if b is a final board state that is lost, then V(b) = -100
3. if b is a final board state that is drawn, then V(b) = 0
4. if b is a not a final state in the game, then V (b) = V (b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

The (4) is a recursive definition and to determine the value of V(b) for a particular board state, it performs the search ahead for the optimal line of play, all the way to the end of the game. So this definition is not efficiently computable by our checkers playing program, we say that it is a nonoperational definition.

The goal of learning, in this case, is to discover an operational description of V ; that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.
It may be very difficult in general to learn such an operational form of V perfectly. We expect learning algorithms to acquire only some approximation to the target function ^V.

1.4.3 Choosing a representation for the Target Function

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the function ^V that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial
neural network. In general, this choice of representation involves a crucial tradeoff. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.

On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation:
for any given board state, the function ^V will be calculated as a linear combination of the following board features:

- x1(b) — number of black pieces on board b
- x2(b) — number of red pieces on b
- x3(b) — number of black kings on b
- x4(b) — number of red kings on b
- x5(b) — number of red pieces threatened by black (i.e., which can be taken on black's next turn)
- x6(b) — number of black pieces threatened by red

^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b)

Where w0 through w6 are numerical coefficients or weights to be obtained by a learning algorithm.
Weights w1 to w6 will determine the relative importance of different board features.

Specification of the Machine Learning Problem at this time — Till now we worked on choosing the type of training experience, choosing the target function and its representation. The checkers learning task can be summarized as below.

- **Task T : Play Checkers**
- **Performance Measure : % of games won in world tournament**
- **Training Experience E : opportunity to play against itself**
- **Target Function : V : Board → R**
- **Target Function Representation : ^V = w0 + w1 · x1(b) + w2 · x2(b) + w3 · x3(b) + w4 · x4(b) +w5 · x5(b) + w6 · x6(b)**

The first three items above correspond to the specification of the learning task,whereas the final two items constitute design choices for the implementation of the learning program.

1.4.4 Choosing an approximation algorithm for the Target Function

Generating training data —

To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b. Each training example is an ordered pair <b,V_train(b)>

For example, a training example may be <(x1 = 3, x2 = 0, x3 = 1, x4 = 0, x5 = 0, x6 = 0), +100">. This is an example where black has won the game since x2 = 0 or red has no remaining pieces. However, such clean values of V_train (b) can be obtained only for board value b that are clear win, loss or draw.

In above case, assigning a training value V_train(b) for the specific boards b that are clean win, loss or draw is direct as they are direct training experience. But in the case of indirect training experience, assigning a training value V_train(b) for the intermediate boards is difficult. In such case, the training values are updated using temporal difference learning. **Temporal difference (TD) learning is a concept central to reinforcement learning, in which learning happens through the iterative correction of your estimated returns towards a more accurate target return.**

Let Successor(b) denotes the next board state following b for which it is again the program's turn to move. ^V is the learner's current approximation to V. Using these information, assign the training value of V_train(b) for any intermediate board state b as below :

V_train(b) ← ^V(Successor(b))

Adjusting the weights

Now its time to define the learning algorithm for choosing the weights and best fit the set of training examples. One common approach is to define the best hypothesis as that which minimizes the squared error E between the training values and the values predicted by the hypothesis ^V.

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in training\ examples} (V_{train}(b) - \hat{V}(b))^2$$

The learning algorithm should incrementally refine weights as more training examples become available and it needs to be robust to errors in training data Least Mean Square (LMS) training rule is the one training algorithm that will adjust weights a small amount in the direction that reduces the error.

The LMS algorithm is defined as follows:

For each training example b

1. Compute $error(b) = V_{train}(b) - \hat{V}(b)$
2. for each board feature x_i, update weight w_i

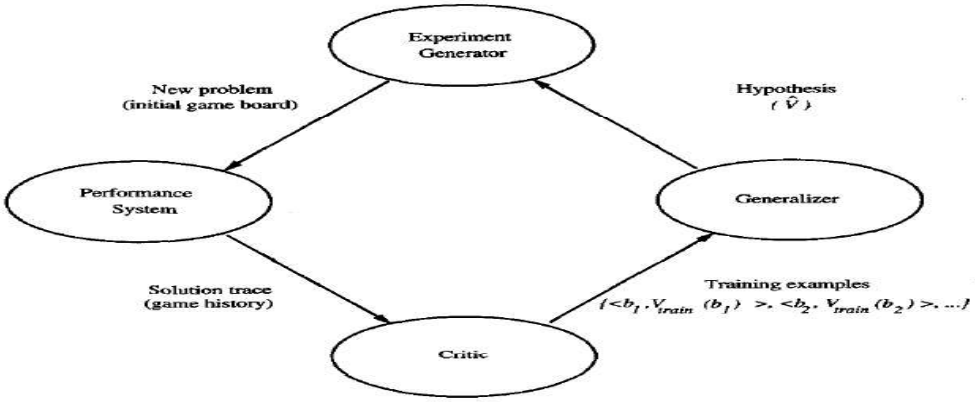
$$w_i \leftarrow w_i + \mu \, . error(b) \, . x_i$$

Here μ is a small constant (e.g., 0.1) that moderates the size of the weight update

1.4.5 Final Design for Checkers Learning system

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.

1. The performance System — Takes a new board as input and outputs a trace of the game it played against itself.
2. The Critic — Takes the trace of a game as an input and outputs a set of training examples of the target function.
3. The Generalizer — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
4. The Experiment Generator — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers learning program.

1.5 Types of Learning

In general, machine learning algorithms can be classified into three types.

- Supervised learning
- Unsupervised learning
- Reinforcement learning

1.5.1 Supervised learning

A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalises to respond correctly to all possible inputs. This is also called learning from exemplars. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value. A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples. In the optimal case, the function will correctly determine the class labels for unseen instances. Both classification and regression

problems are supervised learning problems. A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.

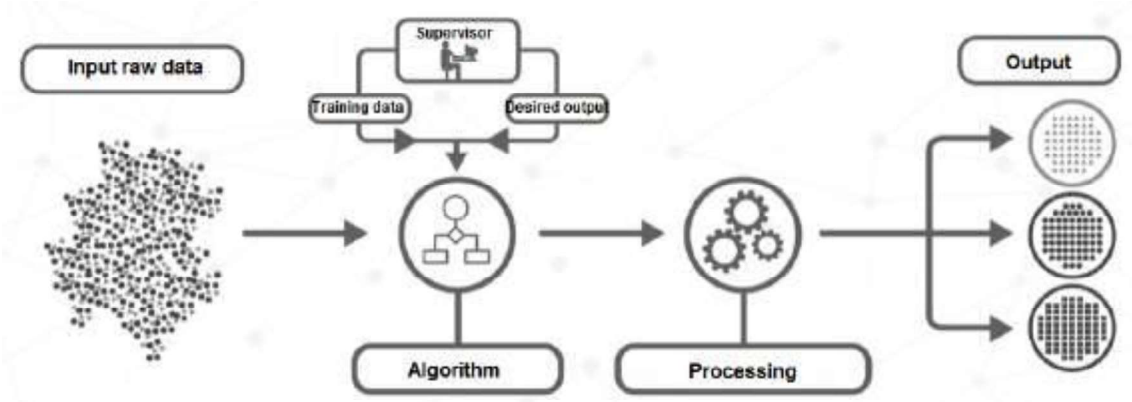


Figure 1.4: Supervised learning

Remarks

A "supervised learning" is so called because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers (that is, the correct outputs), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labeled as "healthy" or "sick".

gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	healthy
F	34	sick
M	21	healthy

1.5.2 Unsupervised learning

Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together. The statistical approach to unsupervised learning is
known as density estimation.

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, a classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns

or grouping in data.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.

gender	age
M	48
M	67
F	53
M	49
F	34
M	21

Based on this data, can we infer anything regarding the patients entering the clinic?

1.5.3 Reinforcement learning

This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a critic because of this monitor that scores the answer, but does not suggest improvements.

Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards. A learner (the program) is not told what actions to take as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards.

Example

Consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs. Reinforcement learning is different from supervised learning. Supervised learning is learning from examples provided by a knowledgeable expert.

1.6 PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Perspectives in Machine Learning

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.

For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights ***wo*** through ***w6.*** The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.