Artificial Intelligence

Course Title: CSE2051

Introduction

Machine Learning is the study of how to build computer systems that adapt and improve with experience. It is a subfield of Artificial Intelligence and intersects with cognitive science, information theory, and probability theory, among others.

Classical AI deals mainly with deductive reasoning, learning represents inductive reasoning. Deductive reasoning arrives at answers to queries relating to a particular situation starting from a set of general axioms, whereas inductive reasoning arrives at general axioms from a set of particular instances.

Classical AI often suffers from the knowledge acquisition problem in real life applications where obtaining and updating the knowledge base is costly and prone to errors. Machine learning serves to solve the knowledge acquisition bottleneck by obtaining the result from data by induction.

Formally, 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 in T, as measured by P, improves with experience E.

Thus a learning system is characterized by:

• Task T

• Experience E, and

• performance measure P

Examples:

Learning to play chess

T: Play chess

P: Percentage of games won in world tournament

E: Opportunity to play against self or other players

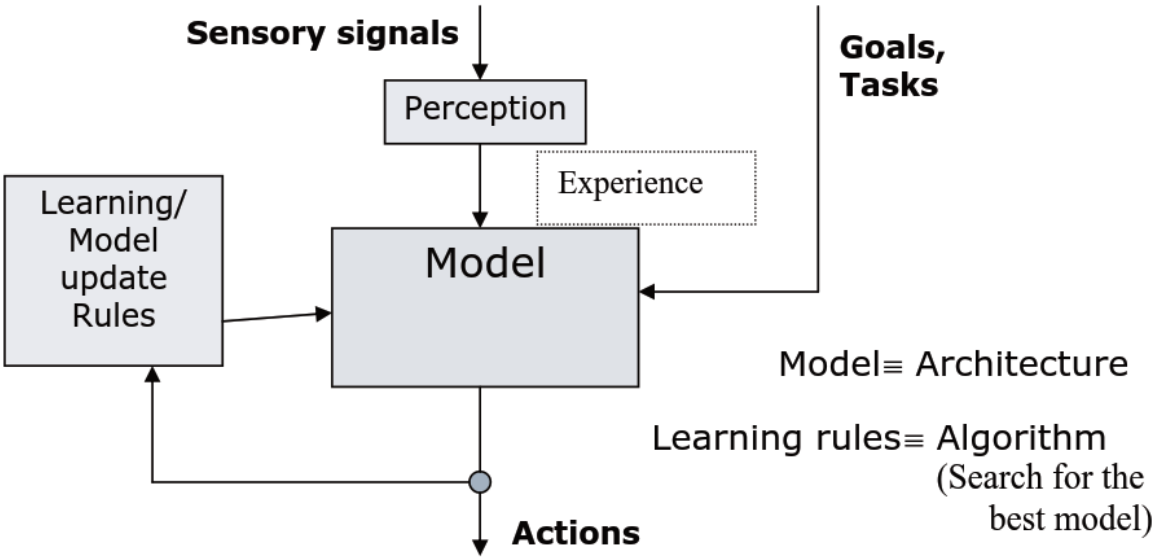
Learning to drive a van

T: Drive on a public highway using vision sensors

P: Average distance traveled before an error (according to human observer)

E: Sequence of images and steering actions recorded during human driving.

The block diagram of a generic learning system which can realize the above definition is shown below:



As can be seen from the above diagram the system consists of the following components:

• Goal: Defined with respect to the task to be performed by the system

• Model: A mathematical function which maps perception to actions

• Learning rules: Which update the model parameters with new experience such that the performance measures with respect to the goals is optimized

• Experience: A set of perception (and possibly the corresponding actions)

12.1.1 Taxonomy of Learning Systems

Several classification of learning systems are possible based on the above components as follows:

Goal/Task/Target Function:

Prediction: To predict the desired output for a given input based on previous input/output pairs. E.g., to predict the value of a stock given other inputs like market index, interest rates etc.

Categorization: To classify an object into one of several categories based on features of the object. E.g., a robotic vision system to categorize a machine part into one of the categories, spanner, hammer etc based on the parts' dimension and shape.

Clustering: To organize a group of objects into homogeneous segments. E.g., a satellite image analysis system which groups land areas into forest, urban and water body, for better utilization of natural resources.

Planning: To generate an optimal sequence of actions to solve a particular problem. E.g., an Unmanned Air Vehicle which plans its path to obtain a set of pictures and avoid enemy anti-aircraft guns.

Models:

• Propositional and FOL rules

• Decision trees

• Linear separators

• Neural networks

• Graphical models

• Temporal models like hidden Markov models

Learning Rules:

Learning rules are often tied up with the model of learning used. Some common rules are gradient descent, least square error, expectation maximization and margin maximization.

Experiences:

Learning algorithms use experiences in the form of perceptions or perception action pairs to improve their performance. The nature of experiences available varies with applications. Some common situations are described below.

Supervised learning: In supervised learning a teacher or oracle is available which provides the desired action corresponding to a perception. A set of perception action pair provides what is called a training set. Examples include an automated vehicle where a set of vision inputs and the corresponding steering actions are available to the learner.

Unsupervised learning: In unsupervised learning no teacher is available. The learner only discovers persistent patterns in the data consisting of a collection of perceptions. This is also called exploratory learning. Finding out malicious network attacks from a sequence of anomalous data packets is an example of unsupervised learning.

Active learning: Here not only a teacher is available, the learner has the freedom to ask the teacher for suitable perception-action example pairs which will help the learner to improve its performance. Consider a news recommender system which tries to learn users preferences and categorize news articles as interesting or uninteresting to the user. The system may present a particular article (of which it is not sure) to the user and ask whether it is interesting or not.

Reinforcement learning: In reinforcement learning a teacher is available, but the teacher instead of directly providing the desired action corresponding to a perception, return reward and punishment to the learner for its action corresponding to a perception. Examples include a robot in an unknown terrain where it's get a punishment when it hits an obstacle and reward when it moves smoothly.

Inductive Learning

This involves the process of *learning by example* -- where a system tries to induce a general rule from a set of observed instances.

This involves classification -- assigning, to a particular input, the name of a class to which it belongs. Classification is important to many problem solving tasks.

A learning system has to be capable of evolving its own class descriptions:

- Initial class definitions may not be adequate.
- The world may not be well understood or rapidly changing.

The task of constructing class definitions is called *induction* or *concept learning.*

Inductive learning has been introduced to help in inducing general rules and predicting future activities. Inductive learning is learning from observation and earlier knowledge by generalization of rules and conclusions. Inductive learning allows for the identification of training data or earlier knowledge patterns and similarities which are then extracted as generalized rules. The identified and extracted generalized rules come to use in reasoning and problem solving.

Mathematical formulation of the inductive learning problem

• Extrapolate from a given set of examples so that we can make accurate predictions about future examples.

• *Supervised versus Unsupervised learning*

> Want to learn an unknown function f(x) = y, where x is an input example and y is the desired output. Supervised learning implies we are given a set of (x, y) pairs by a "teacher." Unsupervised learning means we are only given the x's. In either case, the goal is to estimate f.

Why do we call this inductive learning? We are given some data and we are trying to do induction to try to identify a function, which can explain the data. So, induction as oppose to deduction, unless we can see all the instances all the possible data points or we make some restrictive assumption about the language in which the hypothesis is expressed or some bias, this problem is not well defined so that is why it is called an inductive problem.

when we say we have to learn a function, it is a function of the features; so instances are described in terms of features. So, features are properties that describe each instance; and each instance can be described in a quantitative manner using features.

Categories of Inductive Learning Algorithms

In inductive learning different methods have been proposed to infer classification rules. These methods and techniques were divided into two main categories: Divide-and-Conquer (Decision Tree) and Separate-and-Conquer (Covering). Divide-and-conquer algorithms, such as ID3, C4.5 and CART are classification techniques that derive the general conclusions using decision tree. Separate-and-Conquer algorithms such as AQ family, CN2 (Clark and Niblett), and RULES (RULe Extraction System) family where rules are directly induced from a give set of training examples.

Decision Trees

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called "root" that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute's value. In the case of numeric attributes, the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Figure 1.1 describes a decision tree that reasons whether or not a potential customer will respond to a direct mailing.

Internal nodes are represented as circles, whereas leaves are denoted as triangles. Note that this decision tree incorporates both nominal and numeric attributes. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral characteristics of the entire potential customers population regarding direct mailing. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values.

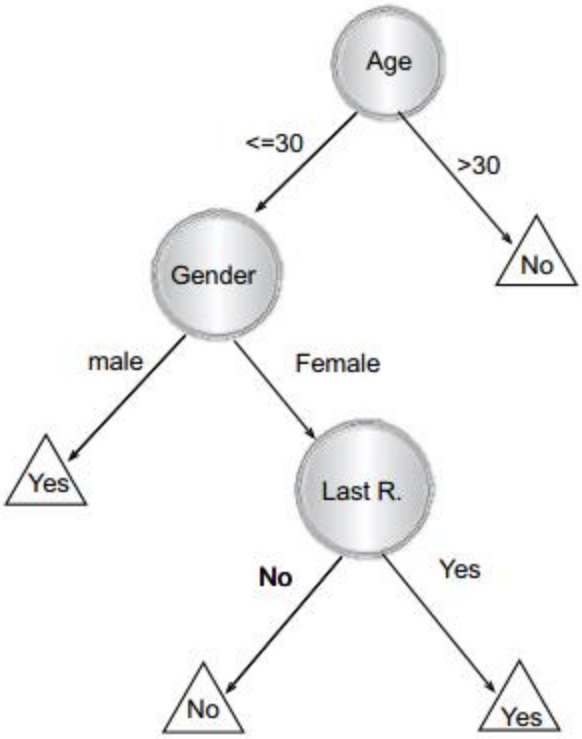


Figure 1.1. Decision Tree Presenting Response to Direct Mailing.

In case of numeric attributes, decision trees can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of the axes. Naturally, decision-makers prefer less complex decision trees, since they may be considered more comprehensible. Furthermore, the tree complexity has a crucial effect on its accuracy. The tree complexity is explicitly controlled by the stopping criteria used and the pruning method employed. Usually the tree complexity is measured by one of the following metrics: the total number of nodes, total number of leaves, tree depth and number of attributes used. Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value. For example, one of the paths in Figure 1.1 can be transformed into the rule:

"If customer age is is less than or equal to or equal to 30, and the gender of the customer is "Male" – then the customer will respond to the mail". The resulting rule set can then be simplified to improve its comprehensibility to a human user, and possibly its accuracy.

Algorithmic Framework for Decision Trees

Decision tree inducers are algorithms that automatically construct a decision tree from a given dataset. Typically the goal is to find the optimal decision tree by minimizing the generalization error. However, other target functions can be also defined, for instance, minimizing the number of nodes or minimizing the average depth.

Induction of an optimal decision tree from a given data is considered to be a hard task. It has been shown that finding a minimal decision tree consistent with the training set is NP–hard. Moreover, it has been shown that constructing a minimal binary tree with respect to the expected number of tests required for classifying an unseen instance is NP–complete. Even finding the minimal equivalent decision tree for a given decision tree or building the optimal decision tree from decision tables is known to be NP–hard.

The above results indicate that using optimal decision tree algorithms is feasible only in small problems. Consequently, heuristics methods are required for solving the problem. Roughly speaking, these methods can be divided into two groups: top–down and bottom–up with clear preference in the literature to the first group.

There are various top–down decision trees inducers such as ID3, C4.5, CART. Some consist of two conceptual phases: growing and pruning (C4.5 and CART). Other inducers perform only the growing phase.

Figure 1.2 presents a typical algorithmic framework for top–down inducing of a decision tree using growing and pruning. Note that these algorithms are greedy by nature and construct the decision tree in a top–down, recursive manner (also known as "divide and conquer"). In each iteration, the algorithm considers the partition of the training set using the outcome of a discrete function of the input attributes. The selection of the most appropriate function is made according to some splitting measures. After the selection of an appropriate split, each node further subdivides the training set into smaller subsets, until no split gains sufficient splitting measure or a stopping criteria is satisfied.

```
TreeGrowing (S,A,y)
Where:
S - Training Set
A - Input Feature Set
y - Target Feature
Create a new tree T with a single root node.
IF One of the Stopping Criteria is fulfilled THEN
    Mark the root node in T as a leaf with the most
    common value of y in S as a label.
ELSE
    Find a discrete function f(A) of the input
        attributes values such that splitting S
        according to f(A)'s outcomes (v1,...,vn) gains
        the best splitting metric.
    IF best splitting metric > treshold THEN
        Label t with f(A)
        FOR each outcome vi of f(A):
            Set Subtreei= TreeGrowing (σf(A)=vi S,A,y).
            Connect the root node of tT to Subtreei with
                    an edge that is labelled as vi
        END FOR
    ELSE
        Mark the root node in T as a leaf with the most
            common value of y in S as a label.
    END IF
END IF
RETURN T
TreeFruning (S,T,y)
Where:
S - Training Set
y - Target Feature
T - The tree to be pruned
DO
    Select a node t in T such that pruning it
        maximally improve some evaluation criteria
    IF t≠Ø THEN T=pruned(T,t)
UNTIL t=Ø
RETURN T
```

Figure 1.2. Top-Down Algorithmic Framework for Decision Trees Induction.

ID3 Algorithm

ID3 builds a decision tree from a fixed set of examples. The resulting tree is used to classify future samples. The example has several attributes and belongs to a class (like yes or no). The leaf nodes of the decision tree contain the class name whereas a non-leaf node is a decision node. The decision node is an attribute test with each branch (to another decision tree) being a possible value of the attribute. ID3 uses information gain to help it decide which attribute goes into a decision node. The advantage of learning a decision tree is that a program, rather than a knowledge engineer, elicits knowledge from an expert. ID3 is based off the Concept Learning System (CLS) algorithm. The basic CLS algorithm over a set of training instances C:

Step 1: If all instances in C are positive, then create YES node and halt.

If all instances in C are negative, create a NO node and halt.

Otherwise select a feature, F with values v1, ..., vn and create a decision node.

Step 2: Partition the training instances in C into subsets C1, C2, ..., Cn according to the values of V.

Step 3: Apply the algorithm recursively to each of the sets Ci.

Note, the trainer (the expert) decides which feature to select.

ID3 improves on CLS by adding a feature selection heuristic. ID3 searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively operates on the n (where n = number of possible values of an attribute) partitioned subsets to get their "best" attribute. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices.

ID3 is a nonincremental algorithm, meaning it derives its classes from a fixed set of training instances. An incremental algorithm revises the current concept definition, if necessary, with a new sample. The classes created by ID3 are inductive, that is, given a small set of training instances, the specific classes created by ID3 are expected to work for all future instances. The distribution of the unknowns must be the same as the test cases. Induction classes cannot be proven to work in every case since they may classify an infinite number of instances. Note that ID3 (or any inductive algorithm) may misclassify data.

Data Description

The sample data used by ID3 has certain requirements, which are:

- Attribute-value description - the same attributes must describe each example and have a fixed number of values.
- Predefined classes - an example's attributes must already be defined, that is, they are not learned by ID3.
- Discrete classes - classes must be sharply delineated. Continuous classes broken up into vague categories such as a metal being "hard, quite hard, flexible, soft, quite soft" are suspect.
- Sufficient examples - since inductive generalization is used (i.e. not provable) there must be enough test cases to distinguish valid patterns from chance occurrences.

Attribute Selection

How does ID3 decide which attribute is the best? A statistical property, called information gain, is used. Gain measures how well a given attribute separates training examples into targeted classes. The one with the highest information (information being the most useful for classification) is selected. In order to define gain, we first borrow an idea from information theory called entropy. Entropy measures the amount of information in an attribute.

Given a collection S of c outcomes

Entropy(S) = $\sum -$ p(I) log2 p(I)

where p(I) is the proportion of $\sum$ belonging to class I. S is over c. Log2 is log base 2.

Note that S is not an attribute but the entire sample set.

Example 1

If S is a collection of 14 examples with 9 YES and 5 NO examples then

Entropy(S) = - (9/14) Log2 (9/14) - (5/14) Log2 (5/14) = 0.940

Notice entropy is 0 if all members of S belong to the same class (the data is perfectly classified). The range of entropy is 0 ("perfectly classified") to 1 ("totally random").

Gain(S, A) is information gain of example set S on attribute A is defined as

Gain(S, A) = Entropy(S) - S (($|S_v|$ / |S|) * Entropy(S_v))

Where:

S is each value v of all possible values of attribute A

S_v = subset of S for which attribute A has value v

$|S_v|$ = number of elements in S_v

|S| = number of elements in S

Example 2

Suppose S is a set of 14 examples in which one of the attributes is wind speed. The values of Wind can be *Weak* or *Strong*. The classification of these 14 examples are 9 YES and 5 NO. For attribute Wind, suppose there are 8 occurrences of Wind = Weak and 6 occurrences of Wind = Strong. For Wind = Weak, 6 of the examples are YES and 2 are NO. For Wind = Strong, 3 are YES and 3 are NO. Therefore

$$Gain(S,Wind)=Entropy(S)-(8/14)*Entropy(S_{weak})-(6/14)*Entropy(S_{strong})$$

$$= 0.940 - (8/14)*0.811 - (6/14)*1.00$$

$$= 0.048$$

Entropy (S_{weak}) = - (6/8)*log2(6/8) - (2/8)*log2(2/8) = 0.811

Entropy (S_{strong}) = - (3/6)*log2(3/6) - (3/6)*log2(3/6) = 1.00

For each attribute, the gain is calculated and the highest gain is used in the decision node.

Example of ID3

Suppose we want ID3 to decide whether the weather is amenable to playing baseball. Over the course of 2 weeks, data is collected to help ID3 build a decision tree (see Table 1).

The target classification is "should we play baseball?" which can be yes or no.

The weather attributes are outlook, temperature, humidity, and wind speed. They can have the following values:

outlook = { sunny, overcast, rain }

temperature = {hot, mild, cool }

humidity = { high, normal }

wind = {weak, strong }

Examples of set S are:

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

We need to find which attribute will be the root node in our decision tree. The gain is calculated for all four attributes:

Gain(S, Outlook) = 0.246

Gain(S, Temperature) = 0.029

Gain(S, Humidity) = 0.151

Gain(S, Wind) = 0.048 (calculated in example 2)

Outlook attribute has the highest gain, therefore it is used as the decision attribute in the root node.

Since Outlook has three possible values, the root node has three branches (sunny, overcast, rain). The next question is "what attribute should be tested at the Sunny branch node?" Since we=92ve used Outlook at the root, we only decide on the remaining three attributes: Humidity, Temperature, or Wind.

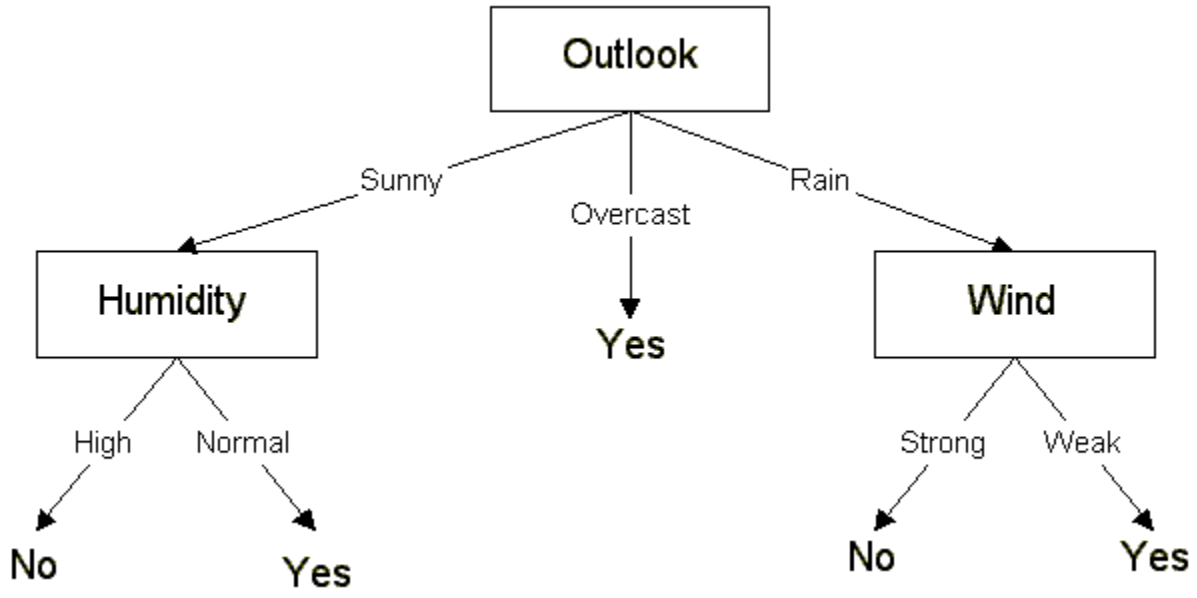
Ssunny = {D1, D2, D8, D9, D11} = 5 examples from table 1 with outlook = sunny

Gain(Ssunny, Humidity) = 0.970

Gain(Ssunny, Temperature) = 0.570

Gain(Ssunny, Wind) = 0.019

Humidity has the highest gain; therefore, it is used as the decision node. This process goes on until all data is classified perfectly or we run out of attributes.



The final decision = tree

The decision tree can also be expressed in rule format:

IF outlook = sunny AND humidity = high THEN playball = no

IF outlook = rain AND humidity = high THEN playball = no

IF outlook = rain AND wind = strong THEN playball = yes

IF outlook = overcast THEN playball = yes

IF outlook = rain AND wind = weak THEN playball = yes

ID3 has been incorporated in a number of commercial rule-induction packages. Some specific applications include medical diagnosis, credit risk assessment of loan applications, equipment malfunctions by their cause, classification of soybean diseases, and web search classification.

ID3 Algorithm—Gain Ratio Version

The information gain version of ID3, listed in Section 2, builds the decision tree with some bias—attributes with greater number of values are preferred by the algorithm. In order to avoid this bias, another version of ID3 has been developed. This version uses another criterion for attribute selection, called gain ratio.

AQ Algorithm

Another rule induction algorithm, developed by R. S. Michalski and his collaborators in the early seventies, is an algorithm called AQ. Many versions of the algorithm have been developed, under different names. Some definitions involved in AQ algorithm:

Let A be the set of all attributes, A = $\{A_1, A_2 \ldots\ldots, A_k\}$. A *seed* is a member of the concept, i.e., a positive case. A *selector* is an expression that associates a variable (attribute or decision) to a value of the variable, e.g., a negation of value, a disjunction of values, etc. A *complex* is a conjunction of selectors. A *partial star* $G(e|e_1)$ is a set of all complexes describing the *seed* $e = (x_1, x_2 \ldots\ldots, x_k)$ and not describing a negative case $e_1 = (y_1, y_2, \ldots\ldots y_k)$. Thus, the complexes of $G(e|e_1)$ are conjunctions of selectors of the form $(A_i, \neg y_i)$, for all i such that $x_i \neq y_i$. A *star* $G(e|F)$ is constructed from all partial stars $G(e|e_i)$, for all $e_i \epsilon F$, and by conjuncting these partial stars by each other, using absorption law to eliminate redundancy. For a given concept C , a *cover* is a disjunction of complexes describing all positive cases from C and not describing any negative cases from F = U − C. The main idea of the AQ algorithm is to generate a cover for each concept by computing stars and selecting from them single complexes to the cover. For the example from Table 1.1, in which attributes are: *Temperature, Headache, Weakness, Nausea*, and the decision is *Flu*. and concept C = {1, 2, 4, 5} described by (Flu, yes), set F of negative cases is equal to 3, 6, 7. A seed is any member of C, say that it is case 1. Then the partial star G(1|3) is equal to

Case	Attributes				Decision
	Temperature	Headache	Weakness	Nausea	Flu
1	very_high	yes	yes	no	yes
2	high	yes	no	yes	yes
3	normal	no	no	no	no
4	normal	yes	yes	yes	yes
5	high	no	yes	no	yes
6	high	no	no	no	no
7	normal	no	yes	no	no

Table 1.1 An Example of DataSet

$$\{(Temperature, \neg normal), (Headache, \neg no), (Weakness, \neg no)\}.$$

Obviously, partial star $G(1|3)$ describes negative cases 6 and 7. The partial star $G(1|6)$ equals

$$\{(Temperature, \neg high), (Headache, \neg no), (Weakness, \neg no)\}$$

The conjunct of $G(1|3)$ and $G(1|6)$ is equal to

$$\begin{gathered}\{(Temperature, very\_high),\\ (Temperature, \neg normal)\ \&\ (Headache, \neg no),\\ (Temperature, \neg normal)\ \&\ (Weakness, \neg no),\\ (Temperature, \neg high)\ \&\ (Headache, \neg no),\\ (Headache, \neg no),\\ (Headache, \neg no)\ \&\ (Weakness, \neg no),\\ (Temperature, \neg high)\ \&\ (Weakness, \neg no),\\ (Headache, \neg no)\ \&\ Weakness, \neg no),\\ (Weakness, \neg no)\},\end{gathered}$$

after using the absorption law, this set is reduced to the following set $G(1|\{3,6\})$:

$$\{(Temperature, very\_high), (Headache \neg no), (Weakness, \neg no)\}.$$

The preceding set describes negative case 7. The partial star $G(1|7)$ is equal to

$$\{(Temperature, \neg normal), Headache, \neg no)\}.$$

The conjunct of G(1|{3, 6}) and G(1|7) is

$$\{(Temperature, very\_high),$$
$$(Temperature, very\_high)\ \&\ (Headache, \neg no),$$
$$(Temperature, \neg normal)\ \&\ Headache, \neg no),$$
$$(Headache, \neg no),$$
$$(Temperature, \neg normal)\ \&\ (Weakness, \neg no),$$
$$(Headache, \neg no)\ \&\ (Weakness, \neg no)\}.$$

The above set, after using the absorption law, is already a star $G(1|F)$

$$\{(Temperature, very\_high),$$
$$(Headache, \neg no),$$
$$(Temperature, \neg normal)\ \&\ (Weakness, \neg no)\}.$$

The first complex describes only one positive case 1, while the second complex describes three positive cases: 1, 2, and 4. The third complex describes two positive cases: 1 and 5. Therefore, the complex

$$(Headache, \neg no)$$

should be selected to be a member of the star of C. The corresponding rule is

$$(Headache, \neg no)\ \rightarrow\ (Flu, yes).$$

If rules without negation are preferred, the preceding rule may be replaced by the following rule

$$(Headache, yes) \rightarrow (Flu, yes).$$

The next seed is case 5, and the partial star $G(5|3)$ is the following set

$$\{(Temperature, \neg normal), (Weakness, \neg no)\}.$$

The partial star $G(5|3)$ covers cases 6 and 7. Therefore, we compute $G(5|6)$, equal to

$$\{(Weakness, \neg no)\}$$

A conjunct of $G(5|3)$ and $G(5|6)$ is the following set

$$\{(Temperature, \neg normal)\ \&\ (Weakness, \neg no), (Weakness, \neg no)\}$$

After simplification, the set $G(5|\{3,6\})$ equals

$$\{Weakness, \neg no)\}.$$

The above set covers case 7. The set $G(5|7)$ is equal to

$$\{(Temperature, \neg normal)\}$$

Finally, the partial star $G(5|\{3,6,7\})$ is equal to

$$\{(Temperature, \neg normal)\ \&\ (Weakness, \neg no)\},$$

so the second rule describing concept {1, 2, 4, 5} is

$$(Temperature, \neg normal)\ \&\ (Weakness, \neg no)\ \rightarrow\ (Flu, yes).$$

It is not difficult to see that the following rules describe the second concept from Table 1.1:

$$Temperature, \neg high)\ \&\ (Headache, \neg yes) \rightarrow (Flu, no),$$

$$(Headache, \neg yes)\ \&\ (Weakness, \neg yes) \rightarrow (Flu, no).$$

Note that the AQ algorithm demands computing conjuncts of partial stars. In the worst case, time complexity of this computation is $O(n^m)$, where n is the number of attributes and m is the number of cases.