

VISUAL INTEGRATED MACHINE LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

CİHANGİR DEVRİM

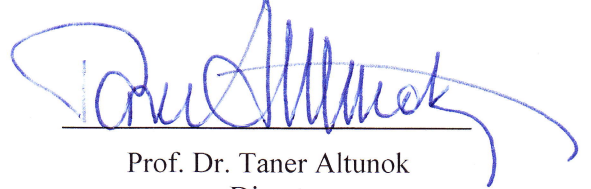
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

DECEMBER 2009

Title of the Thesis : **VISUAL INTEGRATED MACHINE LEARNING**

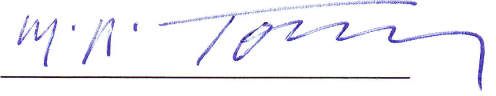
Submitted by **CİHANGİR DEVRİM**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University



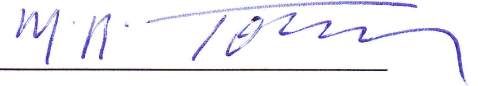
Prof. Dr. Taner Altunok
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.



Prof. Dr. Mehmet R. Tolun
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree Master of Science.

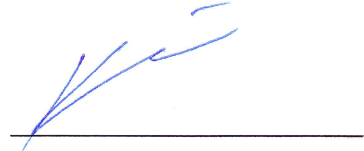


Prof. Dr. Mehmet R. Tolun
Supervisor

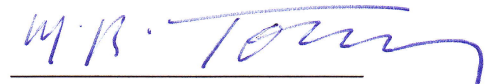
Examination Date : 02.12.2009

Examining Committee Members

Asst. Prof. Dr. Abdül Kadir Görür (Çankaya Univ.)



Prof. Dr. Mehmet R. Tolun (Çankaya Univ.)



Assoc. Prof. Dr. Ferda N. Alpaslan (METU)



STATEMENT OF NON PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Cihangir DEVRİM

Signature :



Date

: 02.12.2009

ABSTRACT

VISUAL INTEGRATED MACHINE LEARNING

DEVİRİM, Cihangir

M.Sc., Department of Computer Engineering

Supervisor : Prof. Dr. Mehmet R. Tolun

December 2009, 55 pages

In this thesis, an artificial intelligence system is developed for creating rules from user data. Before developing the subject of machine learning is researched and AQ software of EMERALD (Experimental Machine Example – based Reasoning and Learning Disciple) is examined in detail to better understand discovering rules from user data. The developed system (ILA Weather) based on Inductive Learning Algorithm (Tolun and Abu Soud, 1998) provides examples for the user to choose through custom design graphical user interface and discovers general rules from selections of the user by using ILA algorithm.

The examples belonging to the graphical user interface is prepared by using Weather Training Example (Quinlan, 1986) and by combining variety of picture sources. Java Swing technology provides wide set of GUI (Graphical User Interface) components for development of desktop applications that is used in the development of ILA Weather which is also built with NetBeans IDE (Integrated Development Environment), an open – source software development tool.

Keywords: Machine Learning, Inductive Learning Algorithm (ILA), AQ Emerald.

ÖZ

GÖRSELLİK ENTEGRE EDİLMİŞ MAKİNE ÖĞRENİMİ

DEVİRİM, Cihangir

Yükseklisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi : Prof. Dr. Mehmet R. Tolun

Aralık 2009, 55 sayfa

Bu tez çalışmamda kullanıcı verilerinden kurallar oluşturan bir yapay zeka sistemi geliştirilmiştir. Geliştirme öncesi makina öğrenimi konusu araştırılmış ve kullanıcı verilerinden kurallar keşfetmeyi daha iyi anlamak için EMERALD (Deneysel Makina Öğrenme Tabanlı Muhakeme ve Öğrenme Disiplini) 'ın AQ yazılımı detaylıca incelenmiştir. Endüktif Öğrenme Algoritması'na (Tolun and Abu Soud, 1998) dayalı olarak geliştirilmiş bu sistem (ILA Weather) özel grafik kullanıcı arayüzü ile kullanıcıya seçmesi için örnekler sunar ve ILA algoritmasını kullanarak kullanıcının seçimlerinden kurallar keşfeder.

Grafik kullanıcı arayüzüne ait olan örnekler, Hava Durumu Eğitim Örnekleri'ni (Quinlan, 1986) kullanarak ve çeşitli resim kaynaklarının bir araya getirilmesiyle hazırlanmıştır. ILA Weather'ın geliştirilmesinde bir masaüstü uygulaması geliştirmek için geniş grafiksel kullanıcı arayüzü bileşenleri sağlayan Java Swing teknolojisi kullanılmış ve aynı zamanda açık kaynak kodlu yazılım geliştirme aracı olan NetBeans IDE (Integrated Development Environment – Entegre Edilmiş Geliştirme Ortamı) ile yapılmıştır.

Anahtar Kelimeler: Makina Öğrenimi, Endüktif Öğrenme Algoritması, AQ Emerald.

ACKNOWLEDGMENTS

The author wishes to express his deepest gratitude to his supervisor Prof. Dr. Mehmet Reşit Tolun for his guidance, advice, criticism, encouragements and insight throughout the research.

TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM	iii
ABSTRACT.....	iv
ÖZ	v
TABLE OF CONTENTS.....	vii
LIST OF TABLES	ix
LIST OF FIGURES	x
CHAPTERS	
1. INTRODUCTION	1
1.1. Learning Activity	1
1.1.1. Designing a Learning System	3
1.1.1.1. Problem Description	3
1.1.1.2. Choosing Training Experience	3
1.1.1.3. Choosing a Target Function.....	4
1.1.1.4. Choosing Representation of the Target Function	6
1.1.1.5. Choosing Function Approximation Algorithm.....	7
1.1.1.6. Final Design.....	9
1.2. Types of Machine Learning.....	11
1.2.1. Supervised Learning	11
1.2.2. Unsupervised Learning	14
1.2.3. Reinforcement Learning	14
1.3. The Aim of the Thesis	15
2. EMERALD-AQ LEARNING SYSTEM.....	16
2.1. Emerald.....	16
2.2. Emerald-AQ.....	21
2.2.1. AQ Learning System	22
2.2.2. Pattern Discovery Methodology	24
2.2.3. Demonstration of Emerald-AQ.....	25

3.	ILA: INDUCTIVE LEARNING ALGORITHM & ILA WEATHER	34
3.1.	Inductive Learning Algorithm	34
3.1.1.	General Requirements.....	35
3.1.2.	Algorithm Steps	35
3.2.	ILA Weather	36
3.2.1.	General Requirements of ILA Weather	36
3.2.2.	Algorithm Steps for ILA Weather	37
3.2.3.	Creating Graphical User Interfaces.....	50
3.2.3.	Demonstration of ILA Weather	51
4.	SUMMARY AND CONCLUSION	54
4. 1.	Further Study	55
	REFERENCES	R1
	APPENDIX	
	WEATHER PICTURES FOR CORRESPONDING TRAINING EXAMPLE	A1

LIST OF TABLES

Table 2.1. All Discovered rules by AQ that robots can join the club.	30
Table 2.2. Discovered rules from each group by AQ.....	32
Table 3.1. Weather Training Examples.....	36
Table 3.2. Initially all rows set unmarked.	37
Table 3.3. Attributes with possible values.	37
Table 3.4. Separation of training examples into Sub-Table 1 and Sub-Table 2	38
Table 3.5. List of combinations according to combination count j	39
Table 3.6. Sub tables state after adding first rule.	40
Table 3.7. Sub table state after adding second rule.....	42
Table 3.8. Sub table state after adding third rule.	44
Table 3.9. Sub table state after adding fourth rule.	47
Table 3.10. Sub table state after adding fifth rule.	49
Table 3.11. Discovered rules by ILA – Weather Training Set.....	49
Table 3.12. Discovered rule list by ILA Weather.	53

LIST OF FIGURES

Figure 1.1. Final design of the checkers learning system.	9
Figure 1.2. Summary of choices in designing the checker learning program.....	10
Figure 1.3. Example of a training dataset where each circle corresponds to one data instance with input values in the corresponding axes and its sign indicates the class.....	12
Figure 1.4. A training dataset of used car and function fitted. For simplicity, milage is taken as only input attribute and a linear model is used.....	13
Figure 2.1. Welcome screen of EMERALD-ILLIAN.	17
Figure 2.2. Menu page of EMERALD-ILLIAN.	17
Figure 2.3. AQ Challenges you screen of EMERALD-ILLIAN.....	18
Figure 2.4. INDUCE Challenges you screen of EMERALD-ILLIAN.....	19
Figure 2.5. CLUSTER Challenges you screen of EMERALD-ILLIAN.	19
Figure 2.6. SPARC challenges you screen of EMERALD-ILLIAN.	20
Figure 2.7. ABACUS Discovers Stoke’s Law screen of EMERALD-ILLIAN.	20
Figure 2.8. The opening screen of the 1999 Java version of the EMERALD.	21
Figure 2.9. The Simple form of PD Mode in AQ21.	24
Figure 2.10. Starting screen of AQ-JPC.	25
Figure 2.11. Menu screen of AQ capabilities.	26
Figure 2.12. Introduction scenario about robots.	27
Figure 2.13. Simple rule of AQ about robots.....	27
Figure 2.14. New robots of AQ.	28
Figure 2.15. The new rule after adding new robots.	28
Figure 2.16. 16 robots – training examples of AQ.....	29
Figure 2.17. Given robots to AQ that some of the robots can join the club, some of the robots cannot join the club.....	29
Figure 2.18. Discovered rule by AQ.	30
Figure 2.19. Selection of groups that user gives examples of robots to AQ.....	31

Figure 2.20. Discovered rules from each group by AQ.....	32
Figure 3.1. Images used in ILA Weather.....	50
Figure 3.2. Welcome screen of ILA Weather.....	51
Figure 3.3. Information screen of weather conditions.....	51
Figure 3.4. An example of given weather conditions by user.....	52
Figure 3.6. A Discovered rule by ILA Weather.....	52

CHAPTER 1

INTRODUCTION

The ability to learn is one of the central features of intelligence, which makes it important for especially artificial intelligence (AI). The field of machine learning (ML) studies computational processes that underline learning in machines [1].

ML contains theories of many fields. These fields can be cognitive psychology, statistics, physiology, biology, some scientific theories (information theory, learning theory, etc.), computational complexities and AI is center of people interest today. Human encounters several problems in real life and waste time on the theories about problems then may be he/she can finds solutions of the problems or not. These efforts contribute evolution and growth of the ML, so we can summarize that to understand the ML we should study from different perspectives.

In the next section learning subject and designing a learning system is explained in detail.

1.1. Learning Activity

First of all, the meaning of "Learning" must be defined. Learning is to gain or understand something (knowledge, skill, behavior, etc.). Learning is everywhere in habitat and everybody (humans and animals) learns each other with some interactions (following, watching and applying). These interactions create experience reciprocally. ML may not reflect all of these interactions but absolutely many techniques of ML derive from these interactions via computational models. Helbert Simon defines the learning activity with the sentence: "Learning is any process by which a system improves performance from experience". To understand

the learning systems in ML, an example of well-posed learning problem and designing a learning system [2] will be summarized and explained. Tom M. Mitchell makes a basic definition of ML with relationship of experience, task and performance [2]:

“ *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 in T, as measured by P, improves with experience E. ”

According to the definition there is an assumption that a computer program learns to play checkers. Learning performance P of the program is measured by its ability to win at the class of task T including playing checkers games and the experience E obtained by playing chess games against itself. There are three points that must be determined to have a well-defined learning problem: the class of tasks, the measure of performance to be improved and the source of the experience. An example of the well – defined problem for checkers learning problem is shown below:

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself.

Some questions may come in mind as there is a checker learning problem but how a computer program learns playing checkers? What type of a system can learns? What is under of a learning system? Answers of these questions will be explained how a learning system can be designed in ML with the given example.

1.1.1. Designing a Learning System

Parts of designing a learning system are the following:

1. Problem Description
2. Choosing Training Experience
3. Choosing Target Function
4. Choosing Representation of the Target Function
5. Choosing Function Approximation Algorithm
6. Final Design

1.1.1.1. Problem Description

Before designing a learning system, of course there should be a problem. The problem is, as previously mentioned, how a computer learns playing checkers.

1.1.1.2. Choosing Training Experience

Choice of training experience impacts on success or failure of leaning system. There are three important parts of choosing type of the training experience.

The first part is that the system learns via *direct training* or *indirect training*. The system learns from checkers board state and correct move of each checkers, called direct training. For another, the system learns from sequences of move and final outcome of several checkers games, called indirect training. In indirect training, the system has a problem called credit assignment, each move of the sequence has a risk when an early move of the opposite is optimal if a poor move is played.

The second part, the system's learner controls sequence of moves, different board states and training examples. If any confusing when the controls are gone, the learner may trusts a teacher and ask it to find the correct move. Otherwise, the learner may learn itself with complete control of board states and no teacher.

The third part, it is important that how well the training experience E represents training examples for measuring the final system performance P . Distribution of the training examples should be similar to future training examples (test examples). If the training experience E consists of only games that the learner plays against itself, it overcomes only playing ability of it. On the other hand, if the training experience E consists of only games that the learner plays against human checker champion (well in playing checkers), it may never encounter most crucial of board states.

Most current theory of ML stands on crucial assumption which the distribution of the training experience is identical the distribution of training test examples. The assumption can be violated in practice.

From the problem, the learner studies to learn the game – playing checkers via playing against itself that we previously mentioned in playing checkers problem as E .

1.1.1.3. Choosing a Target Function

After choosing training experience, there are three main parts to complete the learning system design. These parts are the following:

1. The exact knowledge type to be learned
2. The representation of knowledge
3. The learning mechanism

An assumption when choosing target function that there is a program that generates legal moves on board states and another program – function needs to learn best move from these legal moves. This job can be called as learning task.

The program or function, to choose among the legal moves is the most obvious choice for type of information to be learned. The function must also choose the best move from any board states. The function name is *ChooseMove* which is notated by

ChooseMove: $B \rightarrow M$, B refers input from set of legal board states; M refers output of set of legal moves that is produced by the function.

In ML one of aims is to raise improving performance P at task T. Improving performance is the problem of ML and solution is using function called *target function* such as *ChooseMove*: $B \rightarrow M$. Therefore choosing target function is key design choice of learning system.

The target function *ChooseMove* is shown suitable choice for now but if there is indirect training experience used in learning system, the function will not determine easier best legal move of the board state. An alternative target function is assigning any given board state with numerical scores – values. The new target function is *V* that maps any legal board states B with some real number R, it is denoted by $V: B \rightarrow R$. High scored board states is better board state from others.

If the learning system can learn successfully from the target function, it can easily choose best move from any current board states. The system generates successor board state after every legal move then *V* chooses the best successor state from the board, so best successor state is the best legal move. Any target function can assign the better board state with higher score but it is important to define one target function *V* with a target value $V(b)$ that is arbitrary board state of legal board states B and also produces optimal play on the board states. Rules of the target value $V(b)$ are the following:

1. If b is final board state that is won, then $V(b) = 100$
2. If b is final board state that is lost, then $V(b) = -100$
3. If b is final board state that is drawn, then $V(b) = 0$
4. If b is a not a final board state, then $V(b) = V(b')$, where b' is the best board state that can be achieved starting from b and playing optimally until the end of the game

If above rule definition is not efficiently computable by the checker playing – learning program, it is called *non-operational definition*. The aim of learning task is to find an *operational definition* of V . The checkers playing programs with an operational definition evaluates the board state and can select the legal moves within realistic time bounds. To gain operational description - definition of an ideal target function, the learning task is reduced by using some approximation techniques called *function approximation* in ML. It is another saying that in order to learn perfectly with a target function, some approximation techniques called function approximation is used in the learning systems.

1.1.1.4. Choosing Representation of the Target Function

When representing target function V , a large table (board states and scores) is used with distinct entry specifying the values for each distinct board state. On the other hand, a collection of rules or polynomial function or an artificial neural network can be also used for representing target function. New target function after function approximation is represented by \hat{v} . The program actually learns playing checkers via the target function \hat{v} .

In checkers playing example a linear combination is used for representing the board features. The features are the following:

- x_1 : the number of black pieces on the board
- x_2 : the number of red pieces on the board
- x_3 : the number of black kings on the board
- x_4 : the number of red kings on the board
- x_5 : the number of black pieces threatened by red
- x_6 : the number of red pieces threatened by black

The checkers playing ideal target function is represented by a linear function $\hat{v}(b)$ of the form:

$$\hat{v}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 \quad (1.1)$$

w_0, w_1, \dots, w_6 are numerical coefficients - weights, w_1, w_2, \dots, w_6 shows importance of various board states and w_0 is initial constant board value.

Summarization of design called partial design of a checkers learning program up to this time is the following:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself.
- Target function: $V: B \rightarrow R$
- Target function representation:

$$\hat{v}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

1.1.1.5. Choosing Function Approximation Algorithm

The target function \hat{v} requires a set of training examples. The training example is represented in form of $\hat{v}_{train}(b)$ which is training value of specified board state b and ordered pair of the form $\{b, \hat{v}_{train}(b)\}$.

When direct training experience and rules of target value which previously mentioned in choosing target function part are considered that if b is final board state and number of remaining red pieces equals to zero (0), then black has won the game with the target value $\hat{v}_{train}(b) = +100$. The linear equation is like:

$$\hat{v}_{train}(b) = w_0 + w_1x_1 + w_20 + w_3x_3 + w_40 + w_5x_5 + w_60 = +100$$

Otherwise, when indirect training experience is considered, then there is needed to be estimating training value of $\hat{v}_{train}(b)$ and adjusting the weights w_i .

➤ *Estimating Training Value*

There is an approach for estimating training value of $\hat{v}_{train}(b)$. The approach uses successor board which previously mentioned in choosing target function part. The

training value of $\hat{v}_{train}(b)$ is assigned with $\hat{v}_{train}(Successor(b))$ for any intermediate board state (b) before the checkers game's end. $Successor(b)$ indicates next board state that follows board state (b). The rule for estimating training values is the following:

$$\hat{v}_{train}(b) \leftarrow \hat{v}_{train}(Successor(b)) \quad (1.2)$$

➤ *Adjusting The Weights*

Aim of adjusting the weights is to find best set of weights of a linear function that minimize *squared error* E between training values \hat{v}_{train} and the values predicated by hypothesis estimated \hat{v} . The squared error E refers to *measured training error* that is used to derive a weight learning rule for linear units in *gradient-descent search* [1]. The squared error E can be determined as difference between actual and estimated values of training examples.

$$E \equiv \sum_{\{b, \hat{v}_{train}(b)\} \in \text{training examples}} (\hat{v}_{train}(b) - \hat{v}(b))^2 \quad (1.3)$$

LMS (Least Mean Squares) algorithm is used to decrease E and defined in the following:

For each training example $\{b, \hat{v}_{train}(b)\}$.

- Use the current weights to calculate $\hat{v}(b)$
- For each weight w_i update is as

$$w_i \leftarrow w_i + \eta (\hat{v}_{train}(b) - \hat{v}(b)) X_i \quad (1.4)$$

In algorithm, η is small constant (e.g. 0.1) that affects the size of the weight update. If the difference between training value and estimated value $\hat{v}_{train}(b) - \hat{v}(b)$ is zero, then there is no change about weight update. If the difference is greater than zero, then each weight increases from w_1 to w_i . This raises the estimated value of $\hat{v}(b)$ and reduces the training error.

1.1.1.6. Final Design

The final design of the learning system – checkers learning system has four distinct modules that are components of many learning systems. The modules [2] are shown in Figure 1.1.

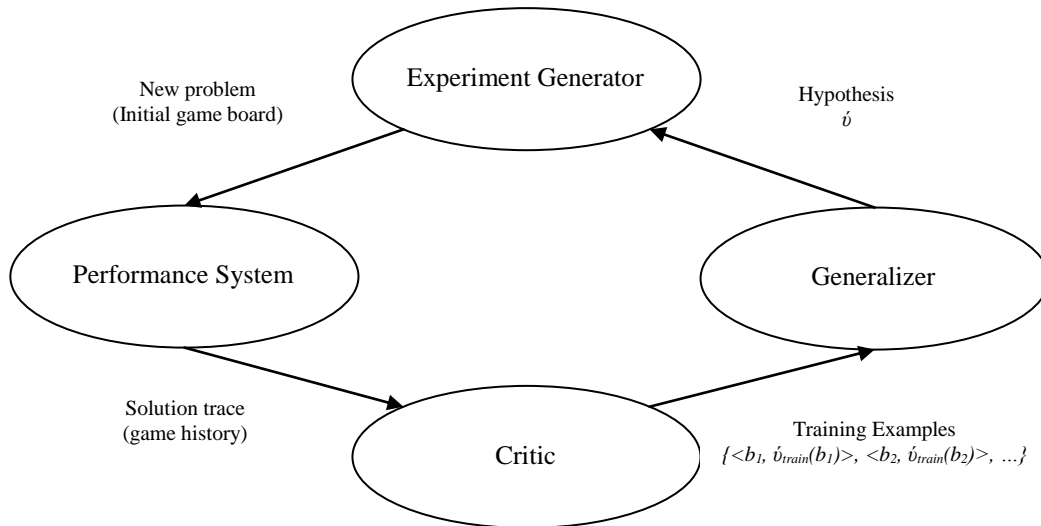


Figure 1.1. Final design of the checkers learning system.

➤ *Performance System*

The module solves the given performance task T – playing checkers by using learned target functions. It takes initial game board as input and produces game history as output. In checkers learning system, it selects the next move at each board state determined by learned target function \hat{v} .

➤ *Critic*

It takes the game history produced by performance system as input and produces set of training example of the target function as output. Training examples are produced by using estimating rule for training values in the choosing function approximation algorithm part.

➤ *Generalizer*

It takes the training examples produced by critic as input and produces hypothesis \hat{v} which is estimation of target function as output. The hypothesis \hat{v} is produced by

adjusting the weights w_0 to w_6 in the choosing function approximation algorithm part.

➤ *Experiment Generator*

It takes generated current hypothesis \hat{v} produced by generalizer as input and produces new problem for the performance system as output.

Many ML systems may be generated from these four modules of the design choice for the checkers learning problem. Sequence of the design choice [2] is summarized in Figure 1.2.

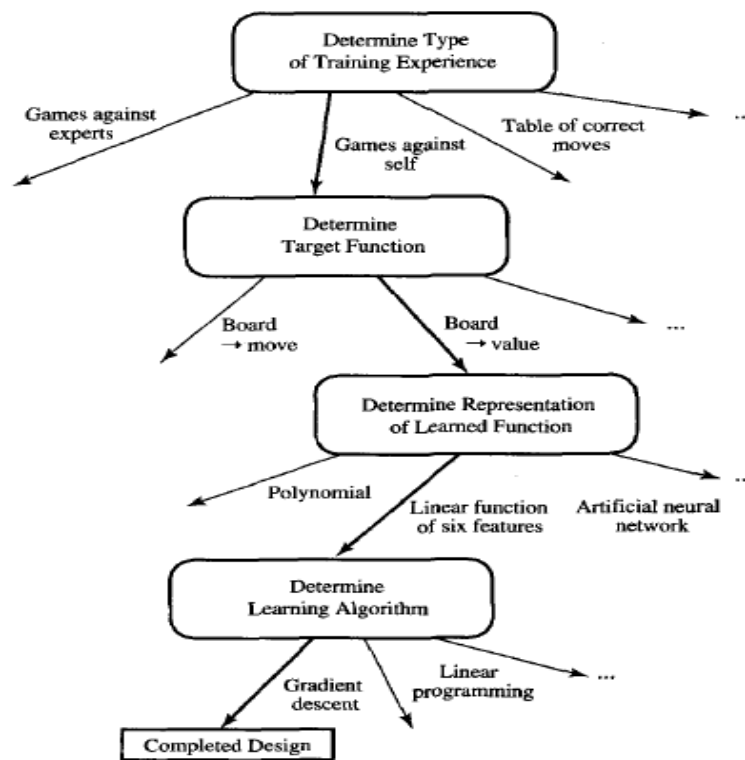


Figure 1.2. Summary of choices in designing the checker learning program.

1.2. Types of Machine Learning

There are many learning types in ML. Bu some types are more popular and known by people.

Some of main types of ML are:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

One kind of learning called *empirical learning* or *inductive learning* that takes externally supplied examples to produce general rules. Many algorithms in this form of learning compares the training examples to find a similarity between them, so this learning is also sometimes called similarity – based learning (SBL) but not all inductive learning algorithms are similarity – based. The empirical learning is subdivided into two types: One is supervised learning, the other is unsupervised learning.

1.2.1. Supervised Learning

Supervised learning is a ML technique for deducing a function from training data. The training data consist of pairs of input objects (typically vectors), and desired outputs. On the other hand, in supervised learning the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor [3].

Definition [4] of supervised learning that there is a given training set of N example for input and output pairs in the form of $(x_1, y_1), (x_2, y_2) \dots (x_N, y_N)$ where each y_j was generated by an unknown function $f(x) = y$ and the learning task T discovers a function h that approximates the true function f . In the definition, x and y can be any value, they need not be numbers. The function h is a hypothesis, j stands for index of N examples and x always refers to input pairs; y always refers to output pairs of training set. In the learning a search is done through the space of possible hypothesis until one performs well. To measure the accuracy of a hypothesis test set of examples which are distinct from the training set is used.

There are two learning problem in supervised learning. These are *classification problem* and *regression problem*. If there are only two possible values of y (e.g. positive or negative) the learning problem is called *classification*.

An example [3] of classification problem that the bank calculates the risk given the amount of credit and information about customer is shown in Figure 1.3.

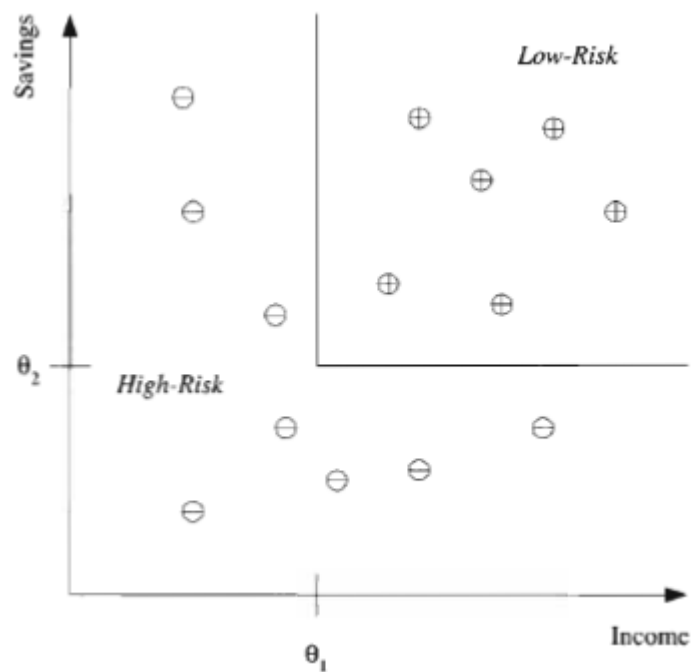


Figure 1.3. Example of a training dataset where each circle corresponds to one data instance with input values in the corresponding axes and its sign indicates the class.

Only two customer attributes, income and savings are taken as input and the two classes are low-risk ('+') and high-risk ('-'). An example discriminant that separates the two types of examples is also shown in Figure 1.3.

The bank has record of past loan of customer and there is a learner that determines risk value of customer from past record called *prediction*. There are only two possible value of risk to give credit: low-risk and high risk customer. The information about customer, savings and income are input pairs of the problem. The

possible values of the risk, low-risk and high risk are output pairs of the problem. The classification may be learned with a predicted rule like this:

$$\text{IF income} > \theta_1 \text{ AND savings} > \theta_2 \text{ THEN low - risk ELSE high - risk} \quad (1.5)$$

If there are possible values of y that has continuous number value for appropriate $f(x)$ the learning problem is called *regression*. The output y is number that provides continues value.

An example [3] of regression problem that the system tries to predict price of used car is shown in Figure 1.4.

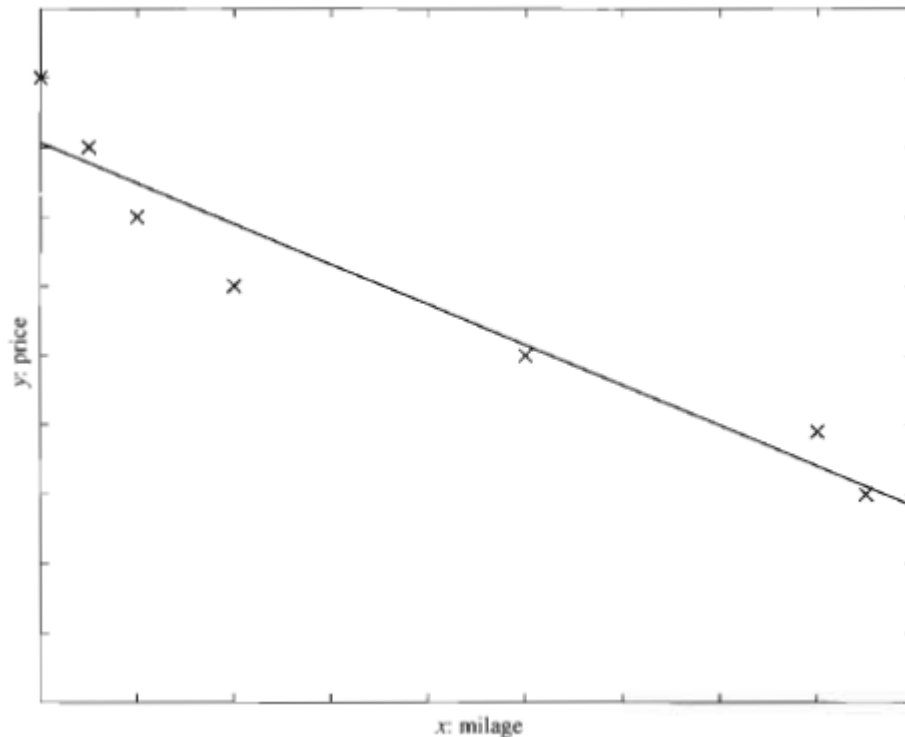


Figure 1.4. A training dataset of used car and function fitted. For simplicity, milage is taken as only input attribute and a linear model is used.

Input pairs are attributes of car that are brand, year, capacity, milage etc. In this example the attribute milage is used for input pair. The output pair is the price of the used car. Using again prediction for past transaction and collecting training data, the learner (machine learning program) may learn the regression with fitted function of

the form $y = w_x + w_0$, x donates the car attribute, y donates the price of the car. w and w_0 are suitable values that the parameters optimized for best fit the training data in linear model.

1.2.2. Unsupervised Learning

The other empirical learning is unsupervised learning. There is no such supervisor in unsupervised learning and only input data exists in training set. On the other hand, unsupervised learning is a class of problems in which one seeks to determine how the data are organized. It is distinguished from supervised learning and reinforcement learning in that the learner is given only unlabeled examples. The aim of this type of learning is to find regularities in the input [3].

Two classic examples of unsupervised learning are clustering and dimensionality reduction [5]. Clustering aims to find clusters of inputs. For example, an interesting application of clustering is in *image compression*. In this case, the input instances are image pixels represented as RGB values. A clustering program groups pixels with similar colors in the same group, and such groups correspond to the colors occurring frequently in the image [3]. The dimensionality reduction aims to decrease dimensionality of the inputs.

1.2.3. Reinforcement Learning

In reinforcement learning [3] the output of the system is a sequence of some *actions*. In such a case, a single action is not important; what is important is the *policy* that is the sequence of correct actions to reach the goal. There is no such thing as the best action in any intermediate state; an action is good if it is part of a good policy. In such a case, the machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. Such learning methods are called *reinforcement learning* algorithms. A good example is *game playing* where a single move by itself is not that important; it is the sequence of right moves that is good. A move is good if it is part of a good game playing policy.

1.3. The Aim of the Thesis

The aim of the thesis is to create an AI system for discovering rules from user data. The AI system will use ILA to discover rules from user data. Weather domain is chosen for creating rules from user data. Custom graphical user interfaces prepared for chosen domain and is implemented with linking prepared graphical user interfaces using ILA algorithm.

CHAPTER 2

EMERALD-AQ LEARNING SYSTEM

EMERALD (Experimental Machine Example – based Reasoning and Learning Discipline) integrates several ML and discovery programs that have been developed on the basis of research conducted by Ryszard S. Michalski over the span of over twenty years, first at the University of Illinois at Urbana-Champaign, and then at the Machine Learning and Inference (MLI) Laboratory at George Mason University (GMU) [6]. In this section information about EMERALD system is given and EMERALD-AQ is focused on in detail.

2.1. Emerald

EMERALD [6] is intended to support teaching and research in the area of ML and demonstrate ML capabilities. The capabilities of EMERALD system include the ability to learn general concepts or decision rules from example, to create meaningful classification of observations, to predict sequence of objects and discover mathematical laws. In the demonstration, the examples consist of very simple objects – pictures of robots, geometric figures, cards, etc. The EMERALD includes also capabilities of natural language processing, voice communication and highly user-oriented graphical interface.

First version of EMERALD, called EMERALD – ILLIAN (SUN - An Integrated Large-Scale Learning and Discovery System for Education and Research in Machine Learning) is initial – short version specifically for demonstrating ML capabilities. This version was developed by MLI Laboratory under the direction of Ryszard.S. Michalski for national exhibition “Robot and Beyond: The Age of Intelligent Machines”, organized by a consortium of eight United State (U.S.)

Museum of Science (Boston, Charlotte, Fort Worth, Los Angeles, Seattle, Chicago, Philadelphia and Columbus) during the years 1987 – 1989. Two version of the system have been implemented at this time, one for DEC VaxStation (EMERALD/M) and the other for Sun workstation (EMERALD/S). Both version are extension of EMERALD-ILLIAN and was developed under direction of R.S. Michalski and in collaboration with K. DeJong, K. Kaufman, A. Schulz, P. Stefanski and J. Zhang [6]. Welcome screen of EMERALD-ILLIAN is shown in Figure 2.1.



Figure 2.1. Welcome screen of EMERALD-ILLIAN.

The EMERALD system integrates five modules – robots that each is displayed a capability for some form of learning and discovery [7]. EMERALD-ILLIAN version will be used to introduce the robots of EMERALD system with an example of screen shot. The EMERALD system has a menu page that represents robots and informs user about their capabilities. The menu page is shown in Figure 2.2.



Figure 2.2. Menu page of EMERALD-ILLIAN.

The user first encounters standard set of choice squares that located at the bottom of the menu page screen. The set of standard choice differs when a robot is selected.

Each robot consists of at least three common sub-programs [6]:

- ❖ Robot Challenges You: It introduces abilities of robots to user.
- ❖ You Challenge Robot: It allows user to experiment with robot.
- ❖ Find out How Robot Works: It explains briefly the theory behinds its operation.

The integrated five modules-robots [6] with an example screen shot of EMERALD system are the following:

- AQ: The module AQ learns decision rules from examples of correct or incorrect decision made by an expert. An example of “AQ Challenges You” screen is shown in Figure 2.3.



Figure 2.3. AQ Challenges you screen of EMERALD-ILLIAN

- *INDUCE*: The module INDUCE learns description of groups of objects and determines important distinction between groups. An example of “INDUCE Challenges You” is shown in Figure 2.4.

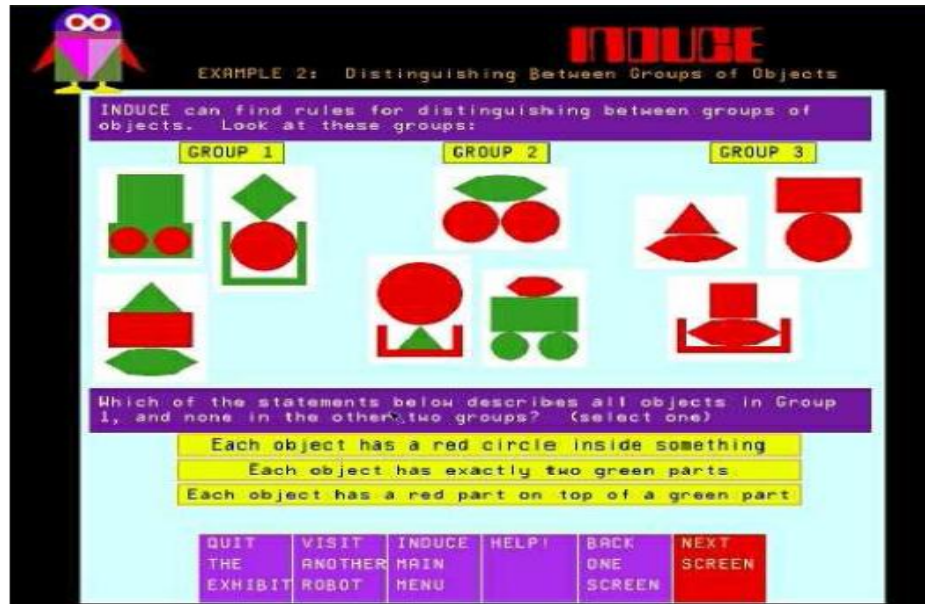


Figure 2.4. INDUCE Challenges you screen of EMERALD-ILLIAN

- *CLUSTOR*: The module CLUSTOR creates meaningful categories and classifications from objects. An example of “CLUSTOR Challenges You” is shown in Figure 2.5.

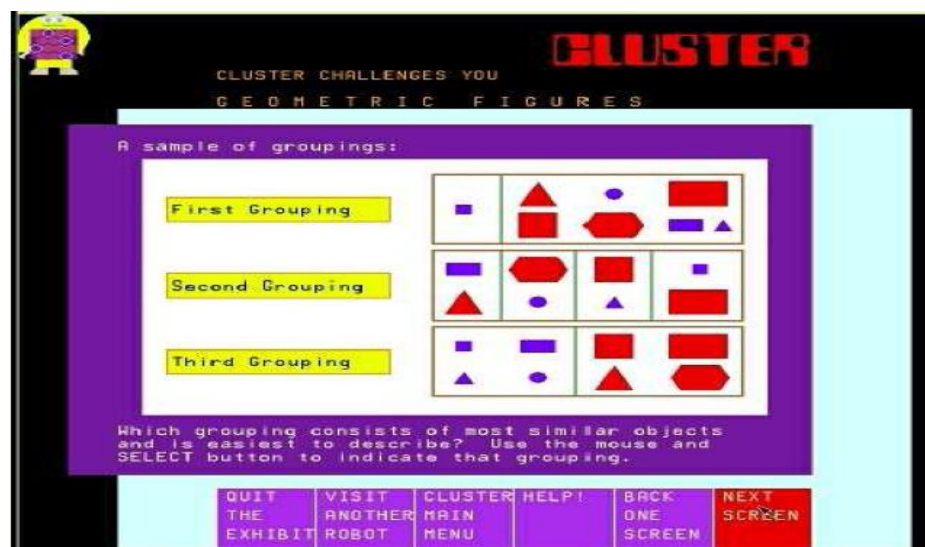


Figure 2.5. CLUSTER Challenges you screen of EMERALD-ILLIAN.

- *SPARC*: The module SPARC predicts possible feature objects in a sequence by discovering rule characterizing the sequence of objects. An example of “SPARC Challenges You” is shown in Figure 2.6.



Figure 2.6. SPARC challenges you screen of EMERALD-ILLIAN.

- *ABACUS*: The module ABACUS conducts experiments, collects data, discovers mathematical and logical description of data, and then uses these descriptions for predicting behavior of an event. An example of “ABACUS DISCOVERS STOKES’S LAW” is shown in Figure 2.7.



Figure 2.7. ABACUS Discovers Stoke’s Law screen of EMERALD-ILLIAN.

In 1999, Java version of EMERALD system developed in the MLI Laboratory [8]. The new version has more user friendly graphical user interface and some extra learning capabilities. The robots AQ, INDUCE, CLUSTOR, SPARC and ABACUS has more colorful. The new Java version opening screen of the EMERALD is shown in Figure 2.8.



Figure 2.8. The opening screen of the 1999 Java version of the EMERALD.

2.2. Emerald-AQ

EMERALD-AQ (A System for Demonstrating Natural Induction for Education and Research in Machine Learning) is a Java-based re-implementation of AQ robot of EMERALD-ILLIAN version. An AI system and knowledge discovery capabilities developed to demonstrate ML. The implementation of the system was completed by Guido Cervone. After re-implementation EMERALD-AQ has new graphical user interface, newly-drawn robots, added animations and software based voice synthesizer. EMERALD-AQ is also previous version of iAQ which demonstrates natural induction to people and last version of AQ learning of the EMERALD system. The other modules-robots (INDUCE, CLUSTOR, SPARC, ABACUS) of new EMERALD system are in the process of re-implementation under Windows OS at MLI Laboratory of GMU.

EMERALD-AQ illustrates natural induction that is a process of creating inductive hypothesis from data in the forms of natural to people. The natural is such as natural language – like expressions or graphical representations. The user creates positive

and negative examples concepts, then EMERALD-AQ tries to discover concepts of the user and describe them in natural language, also verbally in a written form.

There can be said that EMERALD-AQ uses a sub-program - AQ21. It [9] aims to perform natural induction that is a process of generating inductive hypothesis in human-oriented forms which are easy and understand. The human-oriented forms provided by using highly expressive language and Attributional Calculus (AC) whose statements resemble natural language description. AQ21 has property of Pattern Discovery (PD) which produces attributional rules that capture strong regularities in data. The natural induction which seeks patterns represented as rule in AC and it is more expressive than rules typically used in ML. Another important AQ21 features are that it can discover different types of regularities in data, such as conjunctive patterns, general rules with exceptions, gives a choice to the user to include rule-set for parallel or sequential execution and can generate an optimized collection of alternative hypothesis from same data.

There two important parts of EMERALD-AQ. First one is AQ learning system that what inductive hypothesis is used when learning, the other is how the property of PD is provided.

2.2.1. AQ Learning System

Ryszard S. Michalski was the creator of Algorithm Quasi-Optimal or better known as AQ or A^q algorithm. It was originally created in 1969. Mihai stated that AQ algorithm was designed to solve general covering problems of high complexity. In other words, the algorithm was designed to generate generalization or induction from very complex problems [10].

AQ learning system - algorithm generates general hypotheses H_1, \dots, H_k about classes C_1, \dots, C_k , respectively, on the basis of a set of training examples, e_1, \dots, e_n , drawn from these classes. The AQ learning methodology generates hypotheses in the form of attributional rule-sets that optimize a given multi-criterion measure of hypothesis

utility. An attributional rule-set is a set of attributional rules describing the same class [9].

The basic form of an attributional rule [9] is shown like;

$$\text{CONSEQUENT} \leq \text{PREMISE} \quad (2.1)$$

where both CONSEQUENT and PREMISE are conjunctions of attributional conditions in the form:

$$[L \text{ rel } R: A] \quad (2.2)$$

where L is an attribute, an internal conjunction or disjunction of attributes, a *compound attribute*, or a *counting attribute*; rel is one of =, :, >, <, ≤, ≥, or ≠, and R is an attribute value, an internal disjunction of attribute values, an attribute, or an internal conjunction of values of attributes that are constituents of a compound attribute, and A is an optional annotation that lists statistical information about the condition (e.g. p_c and n_c condition coverage, defined as the numbers of *positive* and *negative* examples, respectively, that satisfy the condition).

An example [9] of a simple attributional rule:

$$\begin{aligned} & [\text{activity}=\text{running_experiments}] \leq \\ & [\text{day} = \text{weekend}] \ \& \ [\text{clock_speed} \geq 2\text{GHz}] \ \& \\ & [\text{location} = \text{lab1} \vee \text{lab3}] \ \& \ [\text{weather: quiet} \ \& \ \text{warm}] \end{aligned}$$

which can be paraphrased: the activity is “running experiments” if it is weekend (a higher-level value of the *structured* attribute “day”), the computer clock_speed is at least 2 GHz, the experiment takes place in lab1 or lab3, and the weather is quiet & warm. The attribute “weather” is an example of a *compound attribute*, a new type of attribute introduced in AQ learning that takes a conjunction of values.

The attributional rule in AQ learning uses a richer representation language than in typical rule learning programs, in which conditions are usually limited to a simple form:

$$[\langle \text{attribute} \rangle \ \langle \text{relation} \rangle \ \langle \text{attribute_value} \rangle] \quad (2.3)$$

There is another saying [10] that summarizes definition of AQ algorithm: AQ algorithm used ‘separate and conquer’ approach, where the data would be separated and general rules would be created from the separation. According to Mihai, the central concept of the algorithm is the ‘STAR’ defined as a set of general descriptions of a particular event (a ‘seed’) that satisfies given constraints. In finding the rules, the AQ algorithm used the ‘beam search’ method to explore the data. AQ algorithm is described as follows:

1. The data set would be divided into two parts, according to the conclusion. These parts are known as positive data set and negative data set.
2. One data would be selected randomly from the positive data set. Then this data would be extended against the negative data set by using the STAR method as described above.
3. All the positive data that satisfy the STAR would be removed and one of the remaining positive data would be selected. The STAR method would be applied again.
4. The process stopped when there are no more data in the positive data set.

2.2.2. Pattern Discovery Methodology

The AQ21 program searches for strong patterns that maximize an assumed pattern quality measure. The method takes as input a set of positive example (Pos), a set of negative examples (Neg), and multi-criterion pattern quality measure, defined by the user using Lexicographic Evaluation Functional (LEF) [9]. It follows general algorithm [9] that is shown in Figure 2.9.

```

Hypothesis = null
While Pos is not empty
  Select a seed s from Pos
  Generate approximate star G(s, Neg)
  Select from G the best k rules according
  to LEF, and include them in Hypothesis
  Remove from Pos all positive examples
  covered by the selected rules
Optimize final rules according to LEF
Select the final hypothesis

```

Figure 2.9. The Simple form of PD Mode in AQ21.

The algorithm starts by focusing attention on one positive example of a concept, called the *seed*, and then generates a *star*, defined as set of alternative patterns (generalizations) of the seed that maximize LEF [9]. In PD, the default LEF is to maximize the *pattern quality*, then its coverage, and then to minimize the pattern length (number of conditions) [9]. The pattern quality [9] is defined as:

$$Q(w) = \text{cov}^w * \text{config}^{1-w} \quad (2.4)$$

where $\text{cov} = p/P$ is the relative coverage of the pattern, P and N are number of positive and negative examples in Pos and Neg. The config, *confidence gain* is $((p / (p + n)) - (P / (P + N))) * (P + N) / N$, and w is a parameter controlling the relative importance of relative coverage and confidence gain. The method for generating *star* has been described in various past publications.

2.2.3. Demonstration of Emerald-AQ

The Java source code of EMERALD-AQ is obtained with the aim of research about generating rules from user data. An application with name AQ-JPC was developed by MLI Laboratory of GMU represents EMERALD-AQ capabilities with animated demonstration. The demonstration of EMERALD-AQ is presented with example of screen shots. Starting screen of EMERALD-AQ is shown in Figure 2.10.

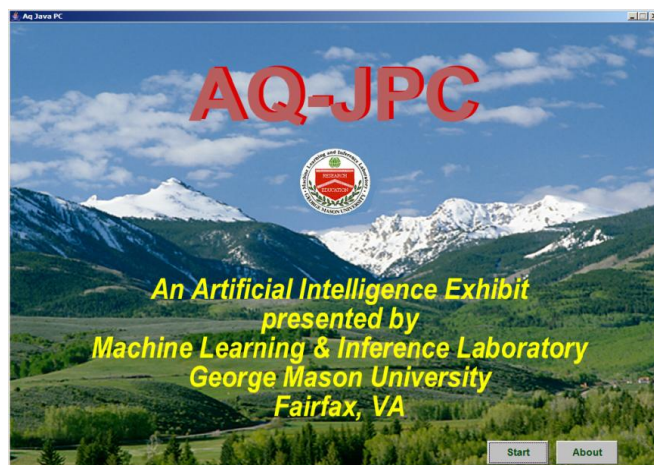


Figure 2.10. Starting screen of AQ-JPC.

There are two buttons in initial screen of AQ-JPC. “Start” button shows the capabilities of AQ, the other “About” button gives information about AQ-JPC and EMERALD system.

After pressing “Start” button in Figure 2.10, the user encounters a menu screen that contains five choices about AQ robot. These are “AQ challenges you with a simple problem”, “Challenge AQ with a simple problem”, “Challenge AQ with a complex problem”, “Find out how AQ works”, and “Run AQ with your own data”. The “Run AQ with your own data” choice is not researched, so this part will not be mentioned. This menu screen of AQ is shown in Figure 2.11.



Figure 2.11. Menu screen of AQ capabilities.

Training examples of AQ are robots. AQ separate the robots into two types of “*Friendly*” and “*Unfriendly*” robots and challenge the user to determine a simple rule about robots. A robot has many properties of jacket color, head, body shape, holding balloon-flag-sword, color of antennas etc. The introduction of scenario and robots screen is shown in Figure 2.12.



Figure 2.12. Introduction scenario about robots.

After next button AQ gives a simple rule that differ two types of robots. This rule is that “A robot is friendly if it is smiling; A robot is unfriendly if it is not smiling”. Simple rule of AQ about robots is shown in Figure 2.13.



Figure 2.13. Simple rule of AQ about robots.

After next button, AQ adds new robots into the training examples to raise degree of the challenge. There are some robots that are smiling but unfriendly. New robots of AQ are shown in Figure 2.14.



Figure 2.14. New robots of AQ.

After next button, AQ gives new the rule (complex rule) that differ two types of robots after adding new robots. This rule is “A robot is friendly if it is smiling and not holding sword, A robot is unfriendly if it is holding a sword or not smiling”. The new rule is shown in Figure 2.15.



Figure 2.15. The new rule after adding new robots.

After next button, AQ gives training example of 16 robots that each one has different property previously mentioned in Figure 2.12. AQ ask the user to invite the robots that can join your club or not. 16 robots (training examples) of AQ are shown

in Figure 2.16 and this screen also belongs to “Challenge AQ with simple problem” choice of menu screen in Figure 2.11.



Figure 2.16. 16 robots – training examples of AQ.

After next button, the user gives some robots that can join your club and some robots cannot join your club. An example of given robots to AQ is shown in Figure 2.17.

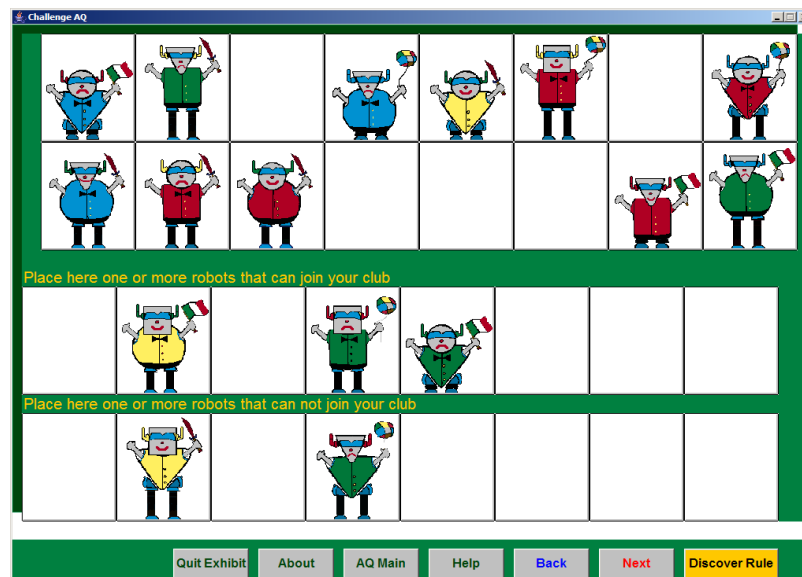


Figure 2.17. Given robots to AQ that some of the robots can join the club, some of the robots cannot join the club.

After giving some robots to AQ, there is “Discover Rule” button that AQ finds rule using AQ algorithm, “A robot can join the club its head is round or square and its height is short or medium” and continues to find rules when each “Discover Alternative Rule” button is clicked. First discovered rule is shown in Figure 2.18.

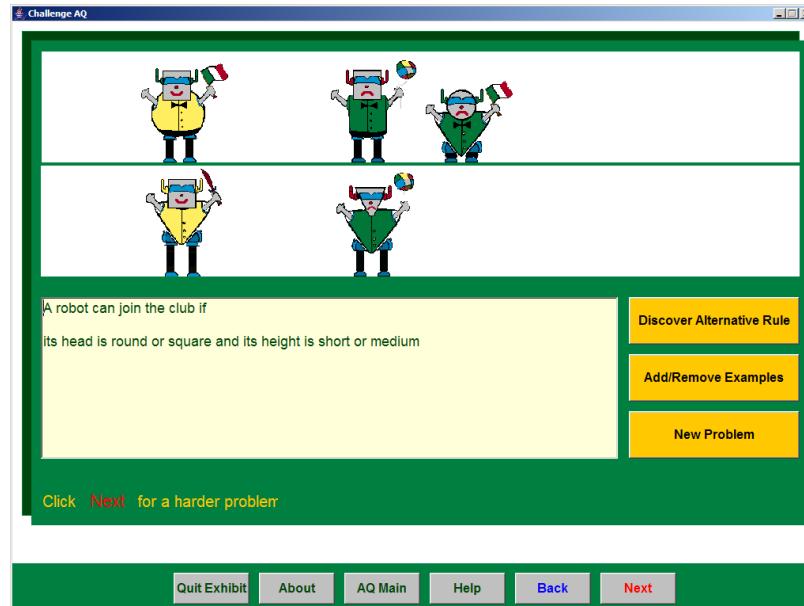


Figure 2.18. Discovered rule by AQ.

The button “Add/Remove Example” turns back the screen in Figure 2.17. The discovered rules by AQ that robots can join the club are shown in Table 2.1.

Table 2.1. All Discovered rules by AQ that robots can join the club.

NO	RULE
RULE 1	<i>A robot can join the club its head is round or square and its height is short or medium.</i>
RULE 2	<i>A robot can join the club if it is wearing a tie.</i>
RULE 3	<i>A robot can join the club if its head is round or square and its antennas are red or green.</i>
RULE 4	<i>A robot can join the club if its head is round or square and it is holding a balloon or a flag.</i>
RULE 5	<i>A robot can join the club if its body is round or square, or if its antennas are green.</i>
RULE 6	<i>A robot can join the club if its head is round or square and its jacket is green, or if its antennas are green.</i>

After next button, the user gives group of robots to AQ by selecting how many groups is used, and then AQ discovers rules for each group. Figure 2.19 shows selection of groups screen that user gives example of robots in those groups and this screen also belongs to “Challenge AQ with a complex problem” choice of menu screen in Figure 2.11.



Figure 2.19. Selection of groups that user gives examples of robots to AQ.

After selecting number of groups 3, the user gives example of robots which are the same robots in Figure 2.17 and AQ discovers rule from each group at the same time. This group selection provides to increase number of property of robots in discovered rule because any property of a robot can be observed as rule for each group. Discovered rules from each group of robots are shown in Figure 2.20.

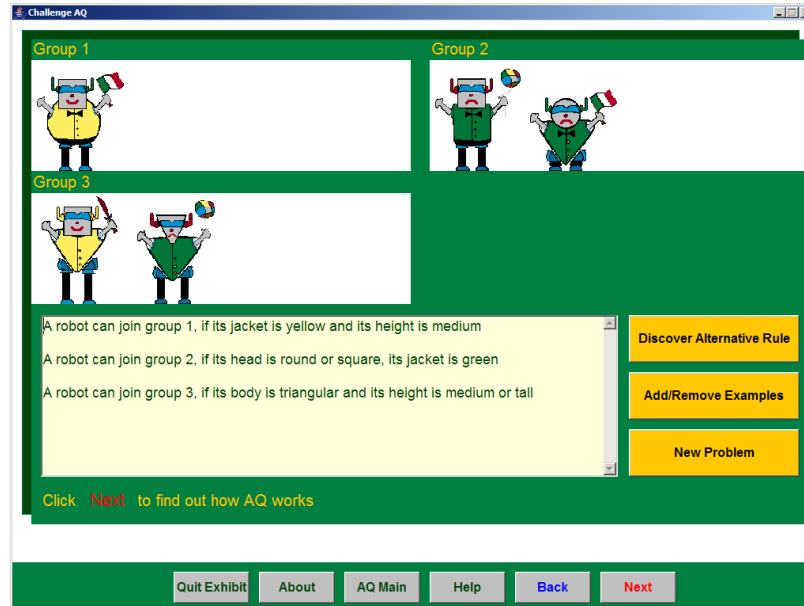


Figure 2.20. Discovered rules from each group by AQ.

The discovered rules from each groups by AQ is shown in Table 2.2.

Table 2.2. Discovered rules from each group by AQ.

NO	RULE
RULE 1	<p>A robot can join group 1, if its jacket is yellow and its height is medium</p> <p>A robot can join group 2, if its head is round or square, its jacket is green</p> <p>A robot can join group 3, if its body is triangular and its height is medium or tall</p>
RULE 2	<p>A robot can join group 1, if its antennas are green and its height is medium</p> <p>A robot can join group 2, if its jacket is green, it is wearing a tie</p> <p>A robot can join group 3, if it is not wearing a tie</p>
RULE 3	<p>A robot can join group 1, if its antennas are green and its height is medium</p> <p>A robot can join group 2, if its head is round or square, its body is square or triangular and its height is short or medium</p> <p>A robot can join group 3, if its body is triangular, its antennas are red or yellow</p>

Table 2.2. Discovered rules from each group by AQ (continued).

NO	RULE
RULE 4	A robot can join group 1, if it is holding a flag and its height is medium A robot can join group 2, if its head is round or square, it is not smiling A robot can join group 3, if its body is triangular and it is holding a sword or a balloon
RULE 5	A robot can join group 1, if it is holding a flag and its height is medium A robot can join group 2, if its body is square or triangular, it is wearing a tie A robot can join group 3, if its head is square or triangular and its body is triangular

CHAPTER 3

ILA: INDUCTIVE LEARNING ALGORITHM & ILA WEATHER

ILA (Inductive Learning Algorithm) invented by Mehmet R. Tolun and Saleh M. Abu-Soud produces IF-THEN rules directly from a set of training examples in general-to-specific way (i.e. starting off with most general rule possible and producing specific rules whenever it is deemed necessary) [11]. ILA also eliminates all unnecessary and irrelevant conditions from extracted and its rules are more simple and general than other algorithms. The generality [11] of rules increases the classification capability of ILA. If a rule becomes more general, IF-part of the rule becomes fewer [11]. In this section information and algorithm steps about ILA is given, and then the developed AI system, ILA Weather is explained.

3.1. Inductive Learning Algorithm

ILA is a new inductive algorithm for generating a set of classification rules from a collection of training examples [11]. The algorithm works in an iterative fashion, each iteration searches a rule that covers a large number of training examples of a single class [11]. When ILA finds a rule, removes those examples it covers from the training set (examples) by marking them and appends a rule at the end of its rule set [11]. In the other words, the algorithm works on a rule-per-class basis. For each class, rules are induced to separate examples in that class from examples in all remaining classes therefore this separation process produces an ordered list of rules [11]. The important parts of ILA are general requirements and steps of the algorithm [11].

3.1.1. General Requirements

1. The examples are to be listed in a table where each row corresponds to an example each column contains attribute values.
2. A set of m training examples, each example composed of k attributes and a class attribute with n possible decisions.
3. A rule set, R with an initial value of ϕ .
4. All rows in the table are initially unmarked.

3.1.2. Algorithm Steps

- ❖ **Step 1:** Partition the table which contains m examples into n sub-tables. One table for each possible value of the class attributes.
(*** Steps 2 through 8 are repeated for each sub-table ***)
- ❖ **Step 2:** Initialize attributes combination count j as $j = 1$.
- ❖ **Step 3:** For the sub-table under consideration, divide the attribute list into distinct combinations, each combination with j distinct attributes.
- ❖ **Step 4:** For each combination of attributes, count the number of occurrence of attribute values that appear under the same combination of attributes in unmarked rows of sub-table under consideration but at the same time that should not appear under the same combination of attributes of other sub-tables. Call the first combination with maximum number of occurrence as max-combination.
- ❖ **Step 5:** If max-combination $=\phi$, increase j by 1 and go to Step 3.
- ❖ **Step 6:** Mark all rows of the sub-table under consideration, in which the values of max-combination appear, as classified.
- ❖ **Step 7:** Add a rule to R whose left hand side comprise attribute names of max-combination with their values separated by *AND* operator(s) and its right hand side contains the decision attribute value associated with the sub-table.

- ❖ **Step 8:** If all rows are marked as classified, then move on to process another sub-table and go to Step 2. Otherwise (i.e. if there are still unmarked rows) go to Step 4. If no sub-table available, exit with set of rules obtained so far.

3.2. ILA Weather

The developed application with name *ILA Weather* takes its name from Weather Training Examples (Quinlan, 1986) and uses ILA algorithm to discover rules from user data. Weather training examples are shown in Table 3.1.

Table 3.1. Weather Training Examples.

where *P* = Positive and *N* = Negative.

Example	Outlook	Temperature	Humidity	Windy	Like
1	Sunny	Hot	High	Not Exist	N
2	Sunny	Hot	High	Exist	N
3	Overcast	Hot	High	Not Exist	P
4	Rain	Mild	High	Not Exist	P
5	Rain	Cool	Normal	Not Exist	P
6	Rain	Cool	Normal	Exist	N
7	Overcast	Cool	Normal	Exist	P
8	Sunny	Mild	High	Not Exist	N
9	Sunny	Cool	Normal	Not Exist	P
10	Rain	Mild	Normal	Not Exist	P
11	Sunny	Mild	Normal	Exist	P
12	Overcast	Mild	High	Exist	P
13	Overcast	Hot	Normal	Not Exist	P
14	Rain	Mild	High	Exist	N

The training examples (or training set, data set) shown above describes the weather conditions for some unspecified games (i.e. playing tennis, playing ball, etc.) but here a little meaningful change is made in class (or decision attribute) such as “*You like the weather if outlook is sunny*” or “*You don’t like the weather if outlook is rain*”. If the decision attribute “Like” is N: negative means “*You do not like the weather*”, otherwise “*You like the weather*”.

3.2.1. General Requirements of ILA Weather

In remembering the general requirements of ILA that previously mentioned, there are fourteen training examples so $m = 14$, each example is composed of attributes

“Outlook”, “Temperature”, “Humidity” and “Windy” so $k = 4$, the class attribute “Like” with two possible of decision of “Negative” and “Positive” so $n = 2$. There is no rule yet so R with an initial value of ϕ . First third general requirements of ILA are accomplished and to accomplish all general requirement of ILA, all rows of training examples is set as unmarked and is shown in Table 3.2.

Table 3.2. Initially all rows set unmarked.

Example	Outlook	Temperature	Humidity	Windy	Like	Marked
1	Sunny	Hot	High	Not Exist	N	No
2	Sunny	Hot	High	Exist	N	No
3	Overcast	Hot	High	Not Exist	P	No
4	Rain	Mild	High	Not Exist	P	No
5	Rain	Cool	Normal	Not Exist	P	No
6	Rain	Cool	Normal	Exist	N	No
7	Overcast	Cool	Normal	Exist	P	No
8	Sunny	Mild	High	Not Exist	N	No
9	Sunny	Cool	Normal	Not Exist	P	No
10	Rain	Mild	Normal	Not Exist	P	No
11	Sunny	Mild	Normal	Exist	P	No
12	Overcast	Mild	High	Exist	P	No
13	Overcast	Hot	Normal	Not Exist	P	No
14	Rain	Mild	High	Exist	N	No

The attributes of training example with possible values is shown in Table 3.3.

Table 3.3. Attributes with possible values.

Attribute	Number of Possible Values	Possible Values
Outlook	3	{Overcast, Sunny, Rain}
Temperature	3	{Hot, Mild, Cool}
Humidity	2	{High, Normal}
Windy	2	{Exist, Not Exist}
Like	2	{Negative, Positive}

3.2.2. Algorithm Steps for ILA Weather

The possible values of the decision attribute $n = 2$ so there are two sub-tables that first one contains “Positive” decisions; the other contains “Negative” decisions. The

training examples into two sub-tables (Sub-Table 1 and Sub-Table 2) are shown in Table 3.4.

Table 3.4. Separation of training examples into Sub-Table 1 and Sub-Table 2

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	No
2	4	Rain	Mild	High	Not Exist	P	No
3	5	Rain	Cool	Normal	Not Exist	P	No
4	7	Overcast	Cool	Normal	Exist	P	No
5	9	Sunny	Cool	Normal	Not Exist	P	No
6	10	Rain	Mild	Normal	Not Exist	P	No
7	11	Sunny	Mild	Normal	Exist	P	No
8	12	Overcast	Mild	High	Exist	P	No
9	13	Overcast	Hot	Normal	Not Exist	P	No
Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	No
2	2	Sunny	Hot	High	Exist	N	No
3	6	Rain	Cool	Normal	Exist	N	No
4	8	Sunny	Mild	High	Not Exist	N	No
5	14	Rain	Mild	High	Exist	N	No

In remembering Step 2 and 3, first Sub-Table 1 is under consideration and initial value of combination count $j = 1$. The important point is to divide the attribute list into distinct combinations and each combination with j distinct attributes so we can understand that if combination count $j = 1$, that means there is one-attribute combinations, if the combination count $j = 2$, that means there is two-attribute combinations so far. List of the combinations according to combination count j is shown in Table 3.5.

Table 3.5. List of combinations according to combination count j

j	Combinations	Distinct Combination
1	(Outlook, Outlook), (Temperature, Temperature), (Humidity, Humidity),(Windy, Windy)	One – Attribute Combination
2	(Outlook, Temperature), (Outlook, Humidity), (Outlook, Windy), (Temperature, Humidity), (Temperature, Windy), (Humidity, Windy)	Two – Attribute Combination
3	(Outlook, Temperature, Humidity), (Outlook, Temperature, Windy), (Temperature, Humidity, Windy)	Three – Attribute Combination
4	(Outlook, Temperature, Humidity, Windy)	Four – Attribute Combination

In remembering Step 4, 5, 6 and 7 with the initial combination count $j = 1$; Sub-Table 1 is under consideration and the one – attribute combinations list of Sub-Table 1 are the following:

(Outlook, Outlook) Combinations

- **Overcast** : Not in Sub-Table 2

For row = 1, number of occurrence = 1,

For row = 4, number of occurrence = 2,

For row = 8, number of occurrence = 3,

For row = 9, number of occurrence = 4

- Sub-Table 2 contains {Rain}, {Sunny}, so they are not considered

There is max-combination = {Overcast} with number of occurrence 4.

(Temperature, Temperature) Combinations

- Sub-Table 2 contains {Hot}, {Mild}, {Cool}, so they are not considered

There is no max-combination of (Temperature, Temperature).

(Humidity, Humidity) Combinations

- Sub-Table contains {High}, {Normal}, so they are not considered

There is no max-combination of (Humidity, Humidity).

(Windy, Windy) Combinations

- Sub-Table 2 contains {Exist}, {Not Exist}, so they are not considered

There is no max-combination of (Windy, Windy).

End of the one – attributes combinations when $j = 1$, {Overcast} is max-combination and we marked all rows of Sub-Table 1 which contains {Overcast}. Then we add new rule and set max-combination = ϕ , the sub tables after adding first rule is shown in Table 3.6.

RULE 1: You like the weather if Outlook is {Overcast}.

Table 3.6. Sub tables state after adding first rule.

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	Yes
2	4	Rain	Mild	High	Not Exist	P	No
3	5	Rain	Cool	Normal	Not Exist	P	No
4	7	Overcast	Cool	Normal	Exist	P	Yes
5	9	Sunny	Cool	Normal	Not Exist	P	No
6	10	Rain	Mild	Normal	Not Exist	P	No
7	11	Sunny	Mild	Normal	Exist	P	No
8	12	Overcast	Mild	High	Exist	P	Yes
9	13	Overcast	Hot	Normal	Not Exist	P	Yes
Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	No
2	2	Sunny	Hot	High	Exist	N	No
3	6	Rain	Cool	Normal	Exist	N	No
4	8	Sunny	Mild	High	Not Exist	N	No
5	14	Rain	Mild	High	Exist	N	No

All max-combination of one- attribute combinations is ϕ , and then increase j to 2, and now $j = 2$, we are applying ILA for unmarked rows of Sub-Table 1. Two – attribute combination list of Sub-Table 1 are the following:

(Outlook, Temperature) Combinations

- **Sunny, Cool:** Not in Sub-Table 2

For row =5, number of occurrence = 1

- Sub-Table 2 contains {Rain, Mild}, {Rain, Cool}, {Sunny, Mild}, so they are not considered

There is max-combination = {Sunny, Cool} with number of occurrence 1.

(Outlook, Humidity) Combinations

- **Sunny, Normal:** Not in Sub-Table 2
For row = 5, number of occurrence = 1
For row = 7, number of occurrence = 2
- Sub-Table 2 contains {Rain, High}, {Rain, Normal}, so they are not considered

There is max-combination = {Sunny, Normal} with number of occurrence 2.

(Outlook, Windy) Combinations

- **Rain, Not Exist:** Not in Sub-Table 2
For row = 2, number of occurrence 1
For row = 3, number of occurrence 2
For row = 6, number of occurrence 3
- Sub-Table 2 contains {Sunny, Not Exist}, {Sunny, Exist}, so they are not considered

There is max-combination = {Rain, Not Exist} with number of occurrence 3.

(Temperature, Humidity) Combinations

- **Mild, Normal:** Not in Sub-Table 2
For row = 6, number of occurrence 1
For row = 7, number of occurrence 2
- Sub-Table 2 contains {Mild, High}, {Cool, Normal}, so they are not considered.

There is max-combination = {Mild, Normal} with number of occurrence 2.

(Temperature, Windy) Combinations

- **Cool, Not Exist:** Not in Sub-Table 2
For row = 3, number of occurrence 1

For row = 5, number of occurrence 2

- Sub-Table 2 contains {Mild, Not Exist}, {Mild, Exist}, so they are not considered

There is max-combination = {Cool, Not Exist} with number of occurrence 2.

(Humidity, Windy) Combinations

- **Normal, Not Exist:** Not in Sub-Table 2

For row = 3, number of occurrence 1

For row = 5, number of occurrence 2

For row = 6, number of occurrence 3

- Sub-Table 2 contains { High, Not Exist }, { Normal, Exist }, so they are not considered

There is max-combination = {Normal, Not Exist} with number of occurrence 3.

End of the tow – attributes combinations when $j = 2$, **(Outlook, Windy) Combinations** and **(Humidity, Windy) Combinations** has max combination with number of occurrence 3 but ILA select first max number of occurrence, so {Rain, Not Exist} is max-combination. We marked all rows of Sub-Table 1 which contains {Rain, Not Exist}. Then we add new rule and set max-combination = ϕ , the sub tables after adding second rule is shown in Table 3.7.

RULE 2: You like the weather if Outlook is {Rain} AND Windy is {Not Exist}.

Table 3.7. Sub table state after adding second rule.

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	Yes
2	4	Rain	Mild	High	Not Exist	P	Yes
3	5	Rain	Cool	Normal	Not Exist	P	Yes
4	7	Overcast	Cool	Normal	Exist	P	Yes
5	9	Sunny	Cool	Normal	Not Exist	P	No
6	10	Rain	Mild	Normal	Not Exist	P	Yes
7	11	Sunny	Mild	Normal	Exist	P	No
8	12	Overcast	Mild	High	Exist	P	Yes
9	13	Overcast	Hot	Normal	Not Exist	P	Yes

Table 3.7. Sub table state after adding second rule (continued).

Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	No
2	2	Sunny	Hot	High	Exist	N	No
3	6	Rain	Cool	Normal	Exist	N	No
4	8	Sunny	Mild	High	Not Exist	N	No
5	14	Rain	Mild	High	Exist	N	No

After adding second rule, combination count j is still 2, $j = 2$. We are applying ILA for unmarked rows of Sub-Table 1. Two – attribute combination list of Sub-Table 1 are the following:

(Outlook, Temperature) Combinations

- **Sunny, Cool:** Not in Sub-Table 2

For row =5, number of occurrence = 1

- Sub-Table 2 contains {Sunny, Mild}, so it is not considered

There is max-combination = {Sunny, Cool} with number of occurrence 1.

(Outlook, Humidity) Combinations

- **Sunny, Normal:** Not in Sub-Table 2

For row = 5, number of occurrence = 1

For row = 7, number of occurrence = 2

There is max-combination = {Sunny, Normal} with number of occurrence 2.

(Outlook, Windy) Combinations

- Sub-Table 2 contains {Sunny, Not Exist}, { Sunny, Exist}, so they are not considered

There is no max-combination of (Outlook, Windy).

(Temperature, Humidity) Combinations

- **Mild, Normal:** Not in Sub-Table 2

For row = 7, number of occurrence 1

- Sub-Table 2 contains {Cool, Normal}, so it is not considered

There is max-combination = {Mild, Normal} with number of occurrence 1.

(Temperature, Windy) Combinations

- **Cool, Not Exist:** Not in Sub-Table 2
For row = 5, number of occurrence 1
- Sub-Table 2 contains {Mild, Exist}, so it is not considered

There is max-combination = {Cool, Not Exist} with number of occurrence 1.

(Humidity, Windy) Combinations

- **Normal, Not Exist:** Not in Sub-Table 2
For row = 5, number of occurrence 1
- Sub-Table 2 contains {Normal, Exist}, so it is not considered

There is max-combination = {Normal, Not Exist} with number of occurrence 1.

End of the two – attributes combinations when $j = 2$, **(Outlook, Humidity) Combinations** has max combination with max number of occurrence 2, so {Sunny, Normal} is max combination. We marked all rows of Sub-Table 1 which contains {Sunny, Normal}. Then we add new rule and set max-combination = ϕ , the sub tables after adding third rule is shown in Table 3.8.

RULE 3: You like the weather if Outlook is {Sunny} AND Humidity is {Normal}.

Table 3.8. Sub table state after adding third rule.

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	Yes
2	4	Rain	Mild	High	Not Exist	P	Yes
3	5	Rain	Cool	Normal	Not Exist	P	Yes
4	7	Overcast	Cool	Normal	Exist	P	Yes
5	9	Sunny	Cool	Normal	Not Exist	P	Yes
6	10	Rain	Mild	Normal	Not Exist	P	Yes
7	11	Sunny	Mild	Normal	Exist	P	Yes
8	12	Overcast	Mild	High	Exist	P	Yes
9	13	Overcast	Hot	Normal	Not Exist	P	Yes

Table 3.8. Sub table state after adding third rule (continued).

Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	No
2	2	Sunny	Hot	High	Exist	N	No
3	6	Rain	Cool	Normal	Exist	N	No
4	8	Sunny	Mild	High	Not Exist	N	No
5	14	Rain	Mild	High	Exist	N	No

All rows of Sub-Table 1 are marked, so Sub-Table 2 is under consideration for applying ILA. The combination count $j = 1$ again and one – attribute combination list of Sub-Table 1 are the following:

(Outlook, Outlook) Combinations

- Sub-Table 1 contains {Sunny},{Rain}, so they are not considered

There is no max-combination of (Outlook, Outlook).

(Temperature, Temperature) Combinations

- Sub-Table 1 contains {Hot},{Mild},{Cool} so they are not considered

There is no max-combination of (Temperature, Temperature).

(Humidity, Humidity) Combinations

- Sub-Table 1 contains {High},{Normal} so they are not considered

There is no max-combination of (Humidity, Humidity).

(Windy, Windy) Combinations

- Sub-Table 1 contains {Exist}, {Not Exist}, so they are not considered

There is no max-combination of (Windy, Windy).

There is no max-combination in one – attribute combination list of Sub-Table 2, increase j to 2, $j = 2$ and tow – attributes combination list of Sub-Table 2 are the following:

(Outlook Temperature) Combinations

- **Sunny, Hot:** Not in Sub-Table 1

For row = 1, number of occurrence 1

For row = 2, number of occurrence 2

- Sub-Table 1 contains {Rain, Cool}, {Sunny, Mild}, {Rain, Mild}, so they are not considered

There is max-combination = {Sunny, Hot} with number of occurrence 2.

(Outlook, Humidity) Combinations

- **Sunny, High:** Not in Sub-Table 1

For row = 1, number of occurrence 1

For row = 2, number of occurrence 2

For row = 3, number of occurrence 3

- Sub-Table 1 contains {Rain, Normal},{Rain, High}, so they are not considered

There is max-combination = {Sunny, High} with number of occurrence 3.

(Outlook, Windy) Combinations

- **Rain, Exist:** Not in Sub-Table 1

For row = 3, number of occurrence 1

For row = 5, number of occurrence 2

- Sub-Table 1 contains {Sunny, Not Exist}, {Sunny, Exist} so they are not considered

There is max-combination = {Rain, Exist} with number of occurrence 2.

(Temperature, Humidity) Combinations

- Sub-Table 1 contains {Hot, High}, {Cool, Normal}, {Mild, High} so they are not considered

There is no max-combination of (Temperature, Humidity).

(Temperature, Windy) Combinations

- **Hot, Exist:** Not in Sub-Table 1

For row = 2, number of occurrence 1

- Sub-Table 1 contains {Hot, Not Exist}, {Cool, Exist},{Mild, Not Exist}, {Mild, Exist}, so they are not considered

There is max-combination = {Hot, Exist} with number of occurrence 1.

(Humidity, Windy) Combinations

- Sub-Table 1 contains {High, Not Exist}, {High, Exist}, {Normal, Exist} so they are not considered

There is no max-combination of (Humidity, Windy).

End of the two – attributes combinations when $j = 2$, **(Outlook, Humidity) Combinations** has max combination with max number of occurrence 3, so {Sunny, High} is max combination. We marked all rows of Sub-Table 2 which contains {Sunny, High}. Then we add new rule and set max-combination = ϕ , the sub tables after adding forth rule is shown in Table 3.9.

RULE 4: You do not like weather if Outlook is {Sunny} AND Humidity is {High}.

Table 3.9. Sub table state after adding forth rule.

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	Yes
2	4	Rain	Mild	High	Not Exist	P	Yes
3	5	Rain	Cool	Normal	Not Exist	P	Yes
4	7	Overcast	Cool	Normal	Exist	P	Yes
5	9	Sunny	Cool	Normal	Not Exist	P	Yes
6	10	Rain	Mild	Normal	Not Exist	P	Yes
7	11	Sunny	Mild	Normal	Exist	P	Yes
8	12	Overcast	Mild	High	Exist	P	Yes
9	13	Overcast	Hot	Normal	Not Exist	P	Yes
Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	Yes
2	2	Sunny	Hot	High	Exist	N	Yes
3	6	Rain	Cool	Normal	Exist	N	No
4	8	Sunny	Mild	High	Not Exist	N	Yes
5	14	Rain	Mild	High	Exist	N	No

After adding forth rule, combination count j is still 2, $j = 2$. We are applying ILA for unmarked rows of Sub-Table 2. Two – attribute combination list of Sub-Table 2 are the following:

(Outlook, Temperature) Combinations

- Sub-Table 1 contains {Rain, Cool}, {Rain, Mild}, so they are not considered

There is no max-combination of (Outlook, Temperature).

(Outlook, Humidity) Combinations

- Sub-Table 1 contains {Rain, Normal}, {Rain, High}, so they are not considered

There is no max-combination of (Outlook, Humidity).

(Outlook, Windy) Combinations

- **Rain, Exist:** Not in Sub-Table 1
For row = 3, number of occurrence 1
For row = 5, number of occurrence 2

There is max-combination = {Rain, Exist} with number of occurrence 2.

(Temperature, Humidity) Combinations

- Sub-Table 1 contains {Cool, Normal}, {Mild, High} so they are not considered

There is no max-combination of (Temperature, Humidity).

(Temperature, Windy) Combinations

- Sub-Table 1 contains {Cool, Exist}, {Mild, Exist}, so they are not considered

There is no max-combination of (Temperature, Windy).

(Humidity, Windy) Combinations

- Sub-Table 1 contains {High, Exist}, {Normal, Exist} so they are not considered

There is no max-combination of (Humidity, Windy).

End of the two – attributes combinations when $j = 2$, **(Outlook, Windy) Combinations** has max combination with max number of occurrence 2, so {Rain, Exist} is max combination. We marked all rows of Sub-Table 2 which contains

{Rain, Exist}. Add rule and set max-combination = ϕ , the sub tables after adding fifth rule is shown in Table 3.10.

RULE 5: You do not like weather if Outlook is {Rain} AND Windy is {Exist}.

Table 3.10. Sub table state after adding fifth rule.

Sub-Table 1							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	3	Overcast	Hot	High	Not Exist	P	Yes
2	4	Rain	Mild	High	Not Exist	P	Yes
3	5	Rain	Cool	Normal	Not Exist	P	Yes
4	7	Overcast	Cool	Normal	Exist	P	Yes
5	9	Sunny	Cool	Normal	Not Exist	P	Yes
6	10	Rain	Mild	Normal	Not Exist	P	Yes
7	11	Sunny	Mild	Normal	Exist	P	Yes
8	12	Overcast	Mild	High	Exist	P	Yes
9	13	Overcast	Hot	Normal	Not Exist	P	Yes
Sub-Table 2							
Row	Ex No	Outlook	Temperature	Humidity	Windy	Like	Marked
1	1	Sunny	Hot	High	Not Exist	N	Yes
2	2	Sunny	Hot	High	Exist	N	Yes
3	6	Rain	Cool	Normal	Exist	N	Yes
4	8	Sunny	Mild	High	Not Exist	N	Yes
5	14	Rain	Mild	High	Exist	N	Yes

After adding fifth rule there is no unmarked rows, so ILA terminates. List of discovered rules by ILA is shown in Table 3.11.

Table 3.11. Discovered rules by ILA – Weather Training Set.

DISCOVERED RULES
RULE 1: You like weather if Outlook is {Overcast}.
RULE 2: You like weather if Outlook is {Rain} AND Windy is {Not Exist}.
RULE 3: You like weather if Outlook is {Sunny} AND Humidity is {Normal}.
RULE 4: You don't like weather if Outlook is {Sunny} AND Humidity is {High}.
RULE 5: You don't like weather if Outlook is {Rain} AND Windy is {Exist}.

3.2.3. Creating Graphical User Interfaces

ILA Weather uses weather conditions pictures that are related to training examples. First, many set of pictures – images, photos, icons, etc. that are appropriate and for weather conditions were researched and then obtained via the Internet. To fully represent the training example, some of these graphical sources were merged with each other with using the photo editing tool (i.e. Adobe Photoshop), used images to represent the training example of ILA Weather are shown in Figure 3.1.

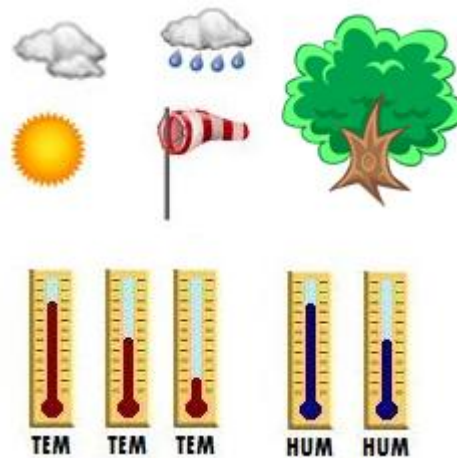


Figure 3.1. Images used in ILA Weather.

Images [12] sun, cloud and cloud is rain were used to represent “Outlook” attribute, image degree is used to represent “Temperature” and “Humidity attribute”, the image of wind flag[12] is used to represent “Windy” attribute and the tree is located in the middle of them to provide or characterize events happened in the earth. Created pictures to represent each training example are given in Appendix A.

After pictures that identify the training example are ready, ILA was implemented with Java Programming Language and by using Java Swing which provides Swing API (Application Programming Interface) to develop commercial-quality desktop applications. NetBeans IDE is chosen for development environment. It is easy to develop desktop application with NetBeans IDE that provides Java GUI Builder to create desktop application easily. It also has drag – and – drop feature that provides

easy design your screens with using GUI components (i.e. JLabel, JButton, JTextField, etc.).

3.2.3. Demonstration of ILA Weather

ILA Weather is a simple desktop application which takes example sets from user and displays discovered rules to the user. Welcome - initial screen of ILA Weather is shown in Figure 3.2.



Figure 3.2. Welcome screen of ILA Weather.

After start button ILA Weather informs user about weather conditions used in graphical user interfaces. The information screen is shown in Figure 3.3.

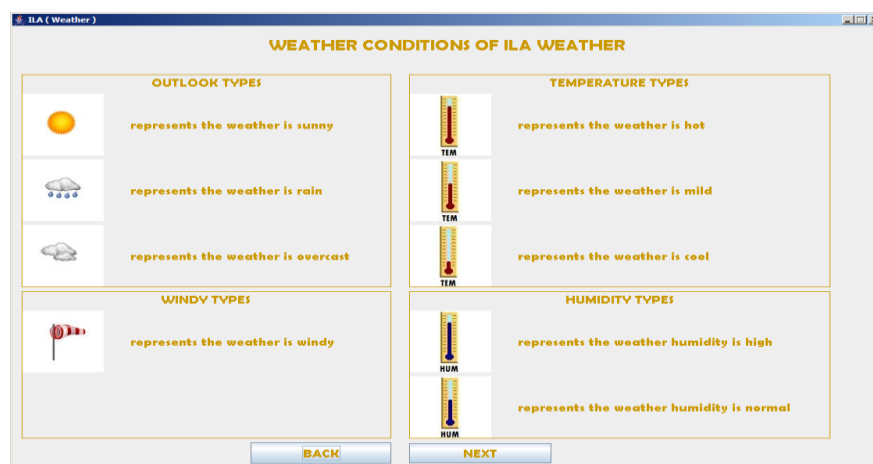


Figure 3.3. Information screen of weather conditions.

After next button, the user gives weather conditions that the user likes or not. An example of given weather conditions by user is shown in Figure 3.4.



Figure 3.4. An example of given weather conditions by user.

After discover rules button, ILA Weather process ILA on given examples of user, and then displays discovered rules on the next screen. A sample discovered rule by ILA Weather is shown in Figure 3.5.

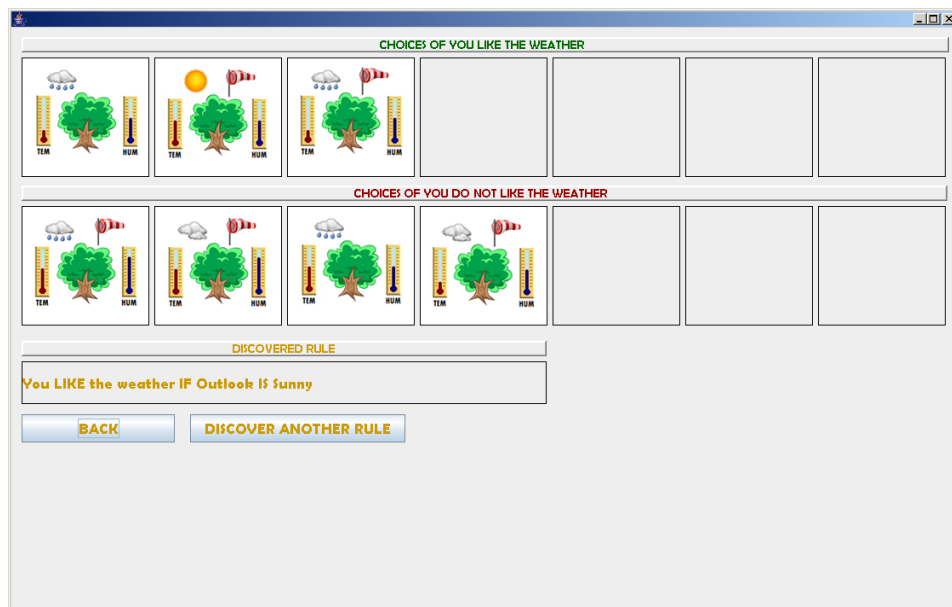


Figure 3.6. A Discovered rule by ILA Weather.

List of discovered rules is according to given examples in Figure 3.4., are shown in Table 3.12.

Table 3.12. Discovered rule list by ILA Weather.

NO	RULE
RULE 1	You LIKE the weather IF Outlook IS Sunny.
RULE 2	You LIKE the weather IF Outlook IS Rain AND Temperature IS Cool.
RULE 3	You do not LIKE the weather IF Outlook IS Overcast.
RULE 4	You do not LIKE the weather IF Humidity IS High.
RULE 5	You do not LIKE the weather IF Outlook IS RAIN AND Temperature IS Mild.

CHAPTER 4

SUMMARY AND CONCLUSION

ILA is a supervised and simple inductive learning algorithm for classifying symbolic data. In particular, it deals discrete and symbolic rules. In this thesis, a visual demonstration of the inductive learning algorithm, ILA, has been realized.

After analyzing the EMERALD System that demonstrates capabilities of ML and an example of the EMERALD System, EMERAL-AQ that creates natural rules from examples – user data via inductive hypothesis, a similar system using ILA algorithm has been designed and implemented in Java.

“Weather Conditions” is chosen as the domain which seemed appropriate for creating rules from the user data having ILA to process user entries on the selected ILA Weather domain. The system discovers rules from user provided data and demonstrates these rules through its custom graphical user interface.

ILA Weather can also be applied to different domains. The domain should provide at least two decision classes that ILA algorithm separates training examples into decision classes and discovers production rules from attributes of training examples. Visual aids of the domain should be carefully determined and introduced to the user. This study does not aim to measure the accuracy of the algorithm under consideration. The contribution of the thesis is to generate a visual demonstration based on examples provided to the system.

The domain to apply ILA algorithm may be determined based on more complex training examples which enforce ILA algorithm to discover rules with more combination of attributes but simple training examples may show clearly how ILA

algorithm works on decision classes. Therefore, ILA Weather has demonstrated the operation of the ILA algorithm clearly with the selected weather condition training examples.

4.1. Further Study

The developed system can be applied easily to other domains, however, in order to do this, domain – dependent graphics should be designed and utilized. Designed graphics can be put on the developed system with a small change in the code. But there are only two restrictions of the system for being used in other domains. First, is the name of the training example attributes (Outlook, Temperature, etc.) , because variable definitions are created according to these attributes. The second is the number of training examples which changes design induction and selection screen of the system. After setting graphics of training examples and some code change on the system, ILA Weather can discover production rules for other domains.

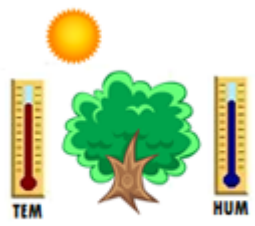
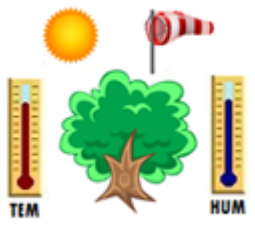
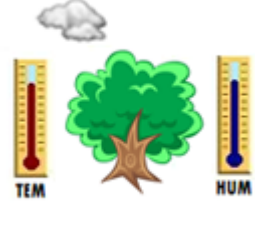
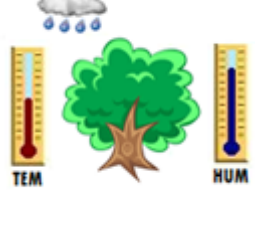
The system user interface consists of only simple graphics - images and user interaction. To have a more powerful demonstration the features of some sounds, animated images, natural language support, etc. can be integrated to the system.

REFERENCES

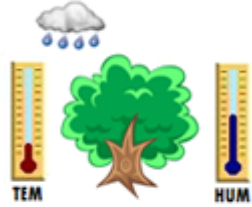
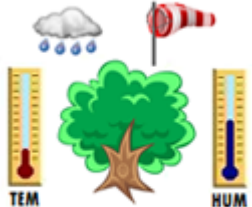
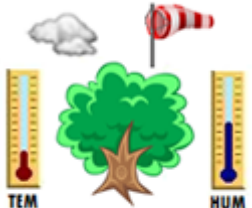
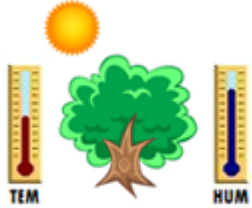
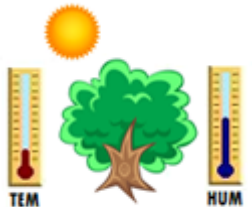
- [1] **Langley P.** (1996), *Elements of Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco.
- [2] **Mitchell T.M.** (1997), *Machine Learning*, McGraw-Hill.
- [3] **Apaydin E.** (2004), *Introduction to Machine Learning*, MIT Press, Cambridge, London.
- [4] **Russell S., Norving P.** (2010), *Artificial Intelligence A Modern Approach Third Edition*, Prentice Hall of Pearson Education Inc.
- [5] **Ghahramani Z.** (2004), *Unsupervised Learning*, Gatsby Computational Neuroscience Unit University College London, UK.
- [6] **Kaufman K. A., Michalski, R. S.** (1997), *EMERALD 2: An Integrated System of Machine Learning and Discovery Programs for Education and Research, Programmer's Guide for the Sun Workstation (Updated Edition)*, Reports of the Machine Learning and Inference Laboratory, MLI 97-9, George Mason University, Fairfax, VA.
- [7] <http://www.mli.gmu.edu/projects/emeraldSun.html>
- [8] <http://www.mli.gmu.edu/msoftware.html>
- [9] **Wojtusiak, J., Michalski, R. S., Kaufman, K., Pietrzykowski, J.** (2006), *The AQ21 Natural Induction Program for Pattern Discovery: Initial Version and its Novel Features*, Proceedings of The 18th IEEE International Conference on Tools with Artificial Intelligence, USA, Washington D.C.
- [10] **Azlinah Hj. M., Shukri Bin Jahabar M.H.** (2006), *Implementation and Comparison of Inductive Learning Algorithms on Timetabling*, Faculty of Information Technology and Quantitative Sciences, Universiti Teknologi MARA, Selangor.
- [11] **Tolun M.R., Abu-Soud S.M.** (1998), *Expert System with Applications, Volume 14, Number 3, ILA: an inductive learning algorithm for rule extraction*, Elsevier.
- [12] <http://iconbest.com/2009/01/07/waether-iconset/>

APPENDIX A

WEATHER PICTURES FOR CORRESPONDING TRAINING EXAMPLE

Ex. No	Outlook	Temperature	Humidity	Windy	Created Picture
1	Sunny	Hot	High	Not Exist	
2	Sunny	Hot	High	Exist	
3	Overcast	Hot	High	Not Exist	
4	Rain	Mild	High	Not Exist	

**WEATHER PICTURES FOR CORRESPONDING TRAINING EXAMPLE
(continued)**

Ex. No	Outlook	Temperature	Humidity	Windy	Created Picture
5	Rain	Cool	Normal	Not Exist	
6	Rain	Cool	Normal	Exist	
7	Overcast	Cool	Normal	Exist	
8	Sunny	Mild	High	Not Exist	
9	Sunny	Cool	Normal	Not Exist	

**WEATHER PICTURES FOR CORRESPONDING TRAINING EXAMPLE
(continued)**

Ex. No	Outlook	Temperature	Humidity	Windy	Created Picture
10	Rain	Mild	Normal	Not Exist	
11	Sunny	Mild	Normal	Exist	
12	Overcast	Mild	High	Exist	
13	Overcast	Hot	Normal	Not Exist	
14	Rain	Mild	High	Exist	