People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Adrar

Faculty of Science and Technology

Department of Mathematics and Computer Science

A Report Presented

For Dynamic System Module

Option: Networks and intelligent systems

Akinator

Proposed by:
KAROUR Nasreddine
OULED JAAFRI Souleymane
TAOUZA Sarah

January 28, 2014

Contents

List of Figures

Introduction

In our days there are deficit of experts of the specific scopes. So we need the something that can substitute them and consult people in the specific scopes.

Expert systems are designed to solve complex problems, like an expert, and have a unique structure different from traditional programs.

Engineers who made these systems, wanted to make something like an Artificial intelligence, but it was impossible and has been impossible today, because the scientific-technical achievements does not allow this.

Such as the Artificial Intelligence Program Akinator, the Web Genius, the most interesting game, which It's able to long in to human mind to figure out which character are thinking of. In this report we have two chapters, the first one mentions Akinator definition, expert system definition, Akinator as an expert system and .

The second chapter contains decision trees algorithms such as ***ID3***, ***ID4***, ***ID5***, ***C4.5***, *and* ***CART***, *and contains also enhancing results algorithms which are* ***probabilistic methods***, ***SVM*** *and* ***binary trees***

Chapter 1

Akinator and Expert System

1.1 Akinator's Definition

is an internet game (interesting Artificial Intelligence Program), the game is accessed using this address *http://en.akinator.com/*, that can determine which character the player is thinking of by asking him or her a series of questions with Yes, No, Probably, Probably not and Don't know as possible answers, and by using each one, the program determines the best question to ask next, in order to narrow down the potential character that the user is thinking of.
Created by three French programmers in 2007, and it's become famous in the whole world with the launch of mobile apps by French mobile company **SCIMOB**, reaching highest ranks on app store.
Here its main page online

Figure 1.1: Akinator main interface

1.2 Expert System

1.2.1 Expert System Definition

An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice. To solve expert-level problems, expert systems will need efficient access to a substantial domain knowledge base , and a reasoning mechanism to apply the knowledge to the problems they are given. Usually they will also need to be able to explain, to the users who rely on them, how they have reached their decisions.

They will generally build upon the ideas of knowledge representation, production rules, search, and so on, that we have already covered.

Often we use an expert system shell which is an existing knowledge independent framework into which domain knowledge can be inserted to produce a working expert system.
We can thus avoid having to program each new system from scratch.

1.2.2 Design of an Expert System

Like a human expert, an expert system is expected to

- *be specialist:* know facts and procedural rules
- *use heuristics:* interpolate from known facts
- *justify its conclusions:* to establish credibility and confidence. The user can ask: how do you know a particular fact? why do you ask a particular question?
- *be able to learn:* be able to absorb new knowledge and apply it
- *estimate the reliability of its answer.*

The structure of a typical expert system is shown on the next page. There are many advantages to the separation and modularity. F or example, the knowledge base and inference engine being separate allows one to inspect the knowledge of the system, not just its responses to particular questions.

1.3 Akinator as an Expert System

Akinator uses an expert system named Limule, it is developed using C++ with a data base of about 100000 persons [1]. Limule is an open system, where new information may change the situation, and the earlier conclusion maybe not true.

The main feature of the systems is possibility to self-study and to remember the information, such as the principle of Akinator, may to remember the answers of users and uses it for next work.

For the storage and use of knowledge in Akinator is making system and knowledge representation, set of procedures required to enter of data gathered from memory and keep them in working

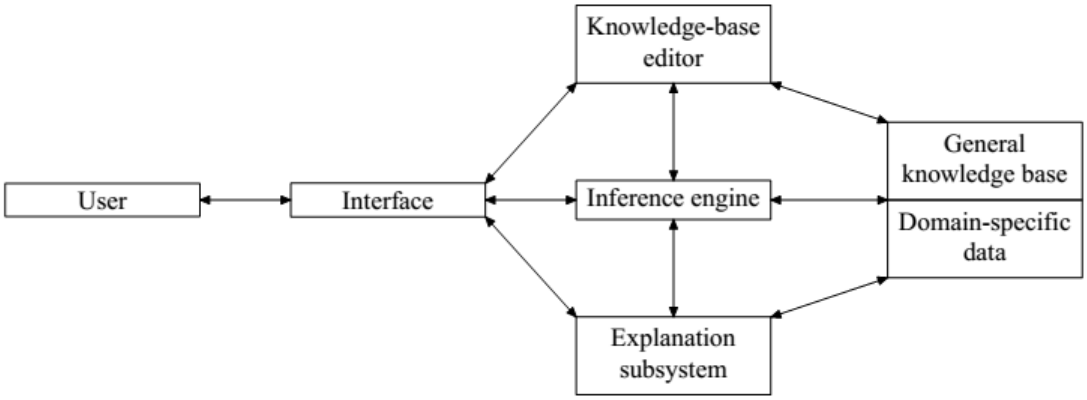

Figure 1.2: Typical Architecture (from Luger and Stubblefield)

order.

Procedures operating of the knowledge are divided into the following classes:

a.) Updating of knowledge.

b.) Classification of knowledge.

c.) Generalization of knowledge.

d.) Output of knowledge.

Akinator has a database of characters and questions with their answers.

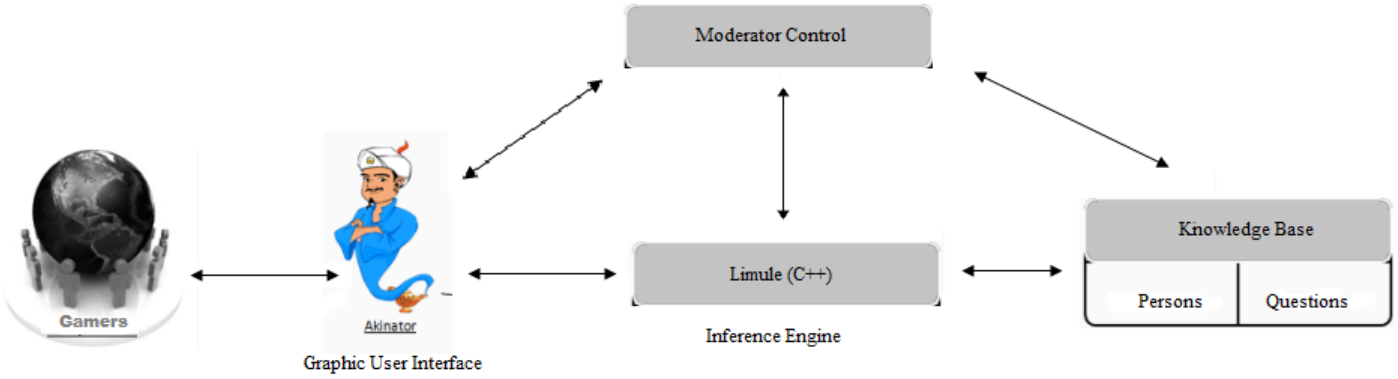

Figure 1.3: The main scheme of Akinator working

1.4 Resolution Principals

Akinator is a web-based game similar to Twenty Questions. The player has to think of a person or character (fictional or not) and answers yes/no questions. Akinator handles uncertainty, as "probably", "probably not" and "I don't know" are other possible answers. Using each answers, the program determines the best question to ask next and eventually gives a guess as to who the player is thinking of. It is a very interesting Artificial Intelligence Program that combines elements from decision trees and binary trees, as well as probabilistic methods and machine learning. If the first guess is not correct, Akinator continues to ask questions, and so on up to three guesses, the first one being generally after 15-20 questions. If the third guess is still not correct, the player is asked to add the character into the database. Although the exact algorithm Akinator uses is kept a secret, we can guess that a decision tree is built based on the character entries in the database, and several searches are performed in this tree.

Chapter 2

Decision Learning Tree Algorithms

2.1 Decision Trees Definition

A decision tree is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a decision. In other words, A decision tree can be seen as a divide-and-conquer strategy for object classification. Formally , one can define a decision tree to be either: [3]

1. a leaf node (or answer node) that contains a class name, or

2. a non-leaf node (or decision node) that contains an attribute test with a branch to another

decision tree for each possible value of the attribute.
Decision tree is commonly used for gaining information for the purpose of decision-making. Decision tree starts with a root node on which it is for users to take actions.
From this node, users split each node recursively according to decision tree learning algorithm.
The final result is a decision tree in which each branch represents a possible scenario of decision and its outcome [2].
Decision trees consist of three types of nodes which are,

- **Decision node:** Often represented by squares showing decisions that can be made. Lines emanating from a square show all distinct options available at a node.

- **Chance node:** Often represented by circles showing chance outcomes. Chance outcomes are events that can occur but are outside the ability of the decision maker to control.

- **Terminal node:** Often represented by triangles or by lines having no further decision nodes or chance nodes. Terminal nodes depict the final outcomes of the decision making process.

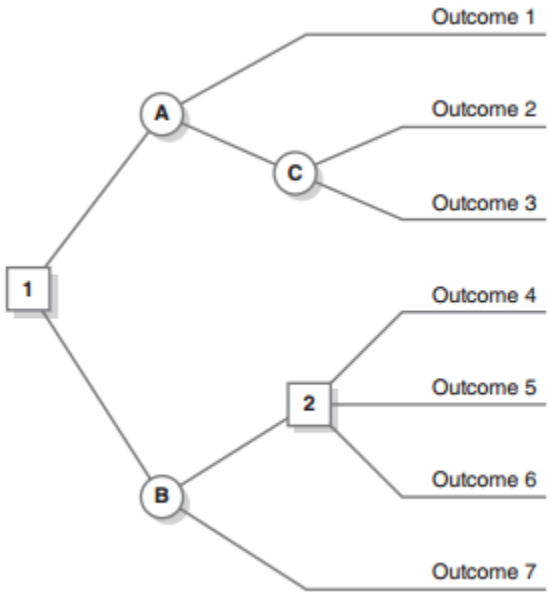

Figure 2.1: Decision trees are graphical models for describing sequential decision problems

2.2 Decision Learning Tree Algorithms

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
Decision tree learning is one of the most widely used and practical methods for inductive inference.

Decision tree learning algorithm has been successfully used in expert systems in capturing knowledge. The main task performed in these systems is using inductive methods to the given values of attributes of an unknown object to determine appropriate classification according to decision tree rules.

Decision trees classify instances by traverse from root node to leaf node. We start from root node of decision tree, testing the attribute specified by this node, then moving down the tree branch according to the attribute value in the given set. This process is the repeated at the sub-tree level. What is decision tree learning algorithm suited for: [2]

1. Instance is represented as attribute-value pairs. For example, attribute **Temperature** and its value **hot**, **mild**, **cool**. We are also concerning to extend attribute-value to continuous-valued data (numeric attribute value) in our project.

2. The target function has discrete output values. It can easily deal with instance which is assigned to a boolean decision, such as “true” and “false”, “p(positive)” and “n(negative)”. Although it is possible to extend target to real valued outputs, we will cover the issue in the later part of this report.

3. The training data may contain errors. This can be dealt with pruning techniques that we will not cover here.

2.2.1 Incremental Induction of Decision Trees 3

2.2.1.1 Definition

Incremental Induction of Decision Trees 3(ID3) is a simple decision tree learning algorithm developed by **Ross Quinlan** on 1983. The basic idea of ID3 algorithm is to construct the decision tree by employing a top-down, greedy search through the given sets to test each attribute at every tree node. In order to select the attribute that is most useful for classifying a given sets, we introduce a metric -information gain.

To find an optimal way to classify a learning set, what is needed to do is to minimize the questions asked (i.e. minimizing the depth of the tree). Thus, we need some function which can measure which questions provide the most balanced splitting.

The information gain metric is such a function.

2.2.1.2 The basic ID3 tree construction algorithm

1. *If all the instances are from exactly one class, then the decision tree is an answer node containing that class name.*
2. *Otherwise,*
   (a) *Define a_{best} to be an attribute with the lowest E-score.*
   (b) *For each value $v_{best,i}$ of a_{best}, grow a branch from a_{best} to a decision tree constructed recursively from all those instances with value $v_{best,i}$ of attribute a best [3].*

2.2.1.3 The Schemes Used To Obtain The Information Gain

There are two main operations during tree building:

(a) evaluation of splits for each attribute and selection of the best split and,

(b) creation of partitions using the best split.

Having determined the overall best split, partitions can be created by a simple application of the splitting criterion to the data. The complexity lies in determining the best split for each attribute. A splitting index is used to evaluate the "goodness" of the alternative splits for an attribute. Several splitting schemes have been proposed in the past (Rastogi and Shim 2000). Two common schemes are considered: the *entropy* and the *gini* index. If a data set S contains examples from m classes, the $Entropy(S)$ and the $Gini(s)$ are defined as followings:

$$Entropy(S) = \sum_{j=1}^{m} P_j log(P_j) \tag{1}$$

$$Gini(S) = 1 - \sum_{j=1}^{m} P_j^2 \tag{2}$$

where P_j is the relative frequency of class j in S. Based on the entropy or the *gini* index, we can compute the information gain if attribute A is used to partition the data set S:

$$Gain(S; A) = Entropy(S) - \sum_{v \in A}(\frac{absS_v}{S} * Entropy(S_v)) \tag{3}$$

$$Gain(S; A) = Gini(S) \sum_{v \in A}(\frac{absS_v}{S} * Gini(S_v)) \tag{4}$$

Where v represents any possible values of attribute A;
S_v is the subset of S for which attribute A has value v;
$|S_v|$ is the number of elements in S_v ; $|S|$ is the number of elements in S.

2.2.2 ID4

Schlimmer and Fisher designed ID4 algorithm to address the incremental construction of decision trees in 1986. ID4 algorithm accepts a new training instance and then updates the decision tree, which avoids rebuilding decision tree for that a global data structure has been kept in the original tree. However, when the relative ordering of possible test attributes at a node changes during the training, it discards all sub-trees blow the node and costs much time to rebuild new sub-trees [7]. The basic ID4 algorithm tree-update procedure is given below.
inputs: *A decision tree, One instance*
output: *A decision tree*

1. *For each possible test attribute at the current node, update the count of positive or negative instances for the value of that attribute in the training instance.*
2. *If all the instances observed at the current node are positive (negative), then the decision tree at the current node is an answer node containing a "+"("-") to indicate a positive (negative) instance.*
3. *Otherwise,*
   (a) *If the current node is an answer node, then change it to a decision node containing an attribute test with the lowest E-score.*
   (b) *Otherwise, if the current decision node contains an attribute test that does not have the lowest E-score, then*
      i *Change the attribute test to one with the lowest E-score.*
      ii *Discard all existing sub-trees below the decision node.*
4. *Recursively update the decision tree below the current decision node along the branch of the value of the current test attribute that occurs in the instance description. Grow the branch if necessary [4].*

2.2.3 C4.5

C4.5 is one such system that learns decision-tree classifiers developed by **Quinlan** in 1993. Several authors have recently noted that C4.5's performance is weaker in domains with a preponderance

of continuous attributes than for learning tasks that have mainly discrete attributes.

C4.5 uses a divide-and-conquer approach to growing decision trees that was pioneered by Hunt and his co-workers(Hunt, Marin, and Stone,1966). Only a brief description of the method is given here.
The following algorithm generates a decision tree from a set of cases:

- *If satisfies a stopping criterion,the tree for is a leaf associated with the most frequent class in. One reason for stopping is that contains only cases of this class,but other criteria can also be formulated.*
- *Some test T with mutually exclusive out comes $T_1, T_2, \ldots, T_k$ is used to partition D into subsets $D_1, D_2, \ldots, D_k$, where D_i contains those cases that have outcome T_i.*
  The tree for D has test T as its root with one sub-tree for each outcome T_i that is constructed by applying the same procedure recursively to the cases in D_i.

Provided that there are no cases with identical attribute values that belong to different classes, any test that produces a non-trivial partition of will eventually lead to single class subsets as above. However, in the expectation that smaller trees are preferable(being easier to understand and often more accurate predictors),a family of possible tests is examined an done of them chosen to maximize the value of some splitting criterion. The default tests considered by C4.5 are:

- $A = ?$ for a discrete attribute, with one outcome for each value of.
- $A \leq t$ for a continuous attribute, with two outcomes, true and false.To find the threshold that maximizes the splitting criterion, the cases in D are sorted on their values of attribute A to give ordered distinct values $v_1, v_2, \ldots, v_N$. Every pair of adjacent values suggests a potential threshold $t = (v_i + v_{i+1})/2$ and a corresponding partition of D. The threshold that yields the best value of the splitting criterion is then selected.

The default splitting criterion used by C4.5 is gain ratio, an information-based measure that takes into account different numbers(and different probabilities)of test outcomes. Let C denote the number of classes and $p(D, j)$ the proportion of cases in D that belong to the j^{th} class. The residual uncertainty about the class to which a case in D belongs can be expressed as

$$Info(D) = \sum_{j=1}^{C} p(D, j) \; x \; log_2(p(D, j))$$

and the corresponding information gained by a test with outcomes as

$$Gain(D, T) = Info(D) - \sum_{j=1}^{k} \frac{|D_i|}{D} \; x \; Info(D_i)$$

The information gained by a test is strongly affected by the number of outcomes and is maximal when there is one case in each subset D_i. On the other hand, the potential information obtained by partitioning a set of cases is based on knowing the subset D_i into which a case falls; this split information

$$Split(D, T) = - \sum_{j=1}^{k} \frac{|D_i|}{D} \; x \; log_2(\frac{|D_i|}{D})$$

tends to increase with the number of outcomes of a test. The gain ratio criterion assesses the desirability of a test as the ratio of its information gain to its split information. The gain ratio of every possible test is determined and, among those with at least average gain, the split with maximum gain ratio is selected.
In some situations, every possible test splits D into subsets that have the same class distribution. All tests then have zero gain, and C4.5 uses this as an additional stopping criterion.
The recursive partitioning strategy above results in trees that are consistent with the training data, if this is possible. In practical applications data are of ten noisy attribute values are incorrectly recorded and cases are misclassified. Noise leads to overly complex trees that attempt to account for these anomalies. Most systems prune the initial tree, identifying sub-trees that contribute little to predictive accuracy and replacing each by a leaf [5].

2.3 Algorithms Enhancement

2.3.1 Probabilistic Methods

Akinator program could possibly uses this kind of method to reduce the number of questions required to decide which character fits the responses the most. The root of the decision tree can be seen as the first question Akinator asks. In reality, the first question is not always the same, but it is always a question that would divide the search space very efficiently, such as “Is this person real?” or “Is this person a man?”. The interior nodes of the decision tree are the following questions the player answers which allow to advance to a next node of the tree. The leaves of the decision tree can be seen as the characters that fit the answers on the path to that leaf. The entire character database should be used to create a tree that is as most binary balanced as possible, allowing to divide the search space by two for every question. Although having a binary tree reduces the cost of the search, it would still be a very high cost to ask every question that gives an element corresponding to a character.
The fact of that in some problems, decisions are made under uncertainty, this fact is the case of Akinator program. As the player answers questions, the program has to eliminate the characters that do not fit the answer given so far. If the data is represented by a decision tree, giving an answer should mean advancing to a next node. The cost of gathering all the information about the character the player is thinking of is high and should require asking too many questions. Therefore the search cost has to be reduced in some way. In this paper, the authors propose a probabilistic method to reduce the search cost, based on past experience. As games are played, Akinator collects a lot of data that could be used. The idea is to skip some decisions for certain nodes, that is to say not ask the question of that particular node if based on experience the probability of taking this path is high enough.
A gain function is introduced. It can be seen as how much the search cost can be reduced. It is reduced the number of decisions that are taken automatically via statistical data. By doing this, there are risks that it was a wrong decision therefore the cost can be increased by the backtracking cost if the statistical decision was wrong. For the gain to be maximum, the decision tree nodes have to be optimally partitioned, that is to say the statistical nodes have to be chosen carefully. Also the backtracking cost has to be reduced. The conclusion is that using probabilities, it is possible to reduce the search cost in a binary decision tree. The method works best when the tree is perfectly binary balanced, which is hard to achieve for Akinator [6].

2.3.2 Binary Trees

A binary tree is made of nodes, where each node contains a left pointer, a right pointer, and a data element. The root pointer points to the topmost node in the tree.

The left and right pointers recursively point to smaller sub-trees on either side. A null pointer represents a binary tree with no elements – the empty tree. The formal recursive definition is: a binary tree is either empty (represented by a null pointer), or is made of a single node, where the left and right pointers (recursive definition ahead) each point to a binary tree.

A binary search tree (BST) or ordered binary tree is a type of binary tree where the nodes are arranged in order: for each node, all elements in its left sub-tree are less-or-equal to the node (<=), and all the elements in its right sub-tree are greater than the node (>).

The tree shown above is a binary search tree -the root node is a 5, and its left sub-tree nodes (1,3,4) are <= 5, and its right sub-tree nodes (6,9) are > 5. Recursively, each of the sub-trees must also obey the binary search tree constraint: in the (1,3,4) sub-tree, the 3 is the root, the 1 <= 3 and 4 > 3. Watch out for the exact wording in the problems a binary search tree is different from a binary tree.

The nodes at the bottom edge of the tree have empty sub-trees and are called leaf nodes (1,4,6) while the others are internal nodes (3,5,9).

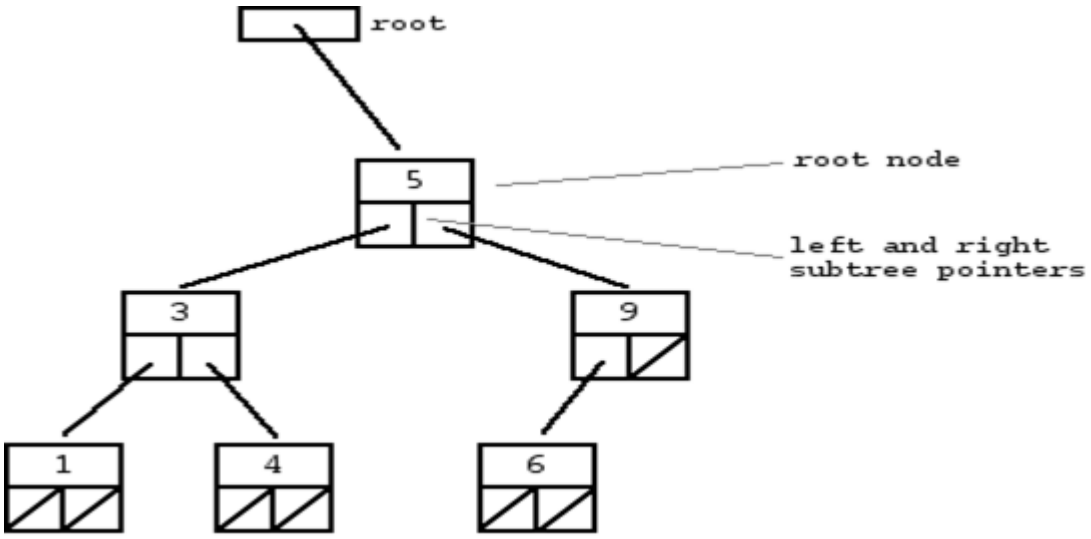

Figure 2.2: General structure of a binary tree

2.3.3 Machine Learning using (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples [9].

Support vector machines (SVMs) are a set of related supervised learning methods that analyze

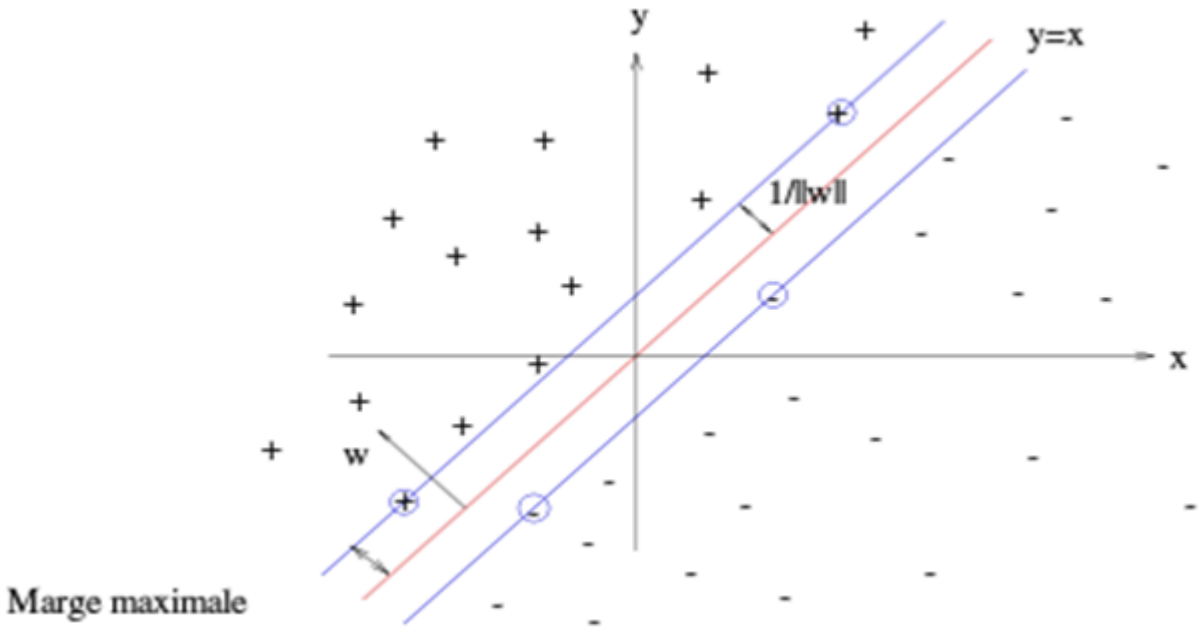

Figure 2.3: General structure of a binary tree

data and recognize patterns, used for classification (machine learning)|classification and regression analysis. In other words, we can say a support vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space|high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. Whereas the original problem may be stated in a finite dimensional space, it often happens that in that space the sets to be discriminated are not linearly separable. For this reason it was proposed that the original finite dimensional space be mapped into a much higher dimensional space presumably making the separation easier in that space. SVM schemes use a mapping into a larger space so that cross products may be computed easily in terms of the variables in the original space making the computational load reasonable. The cross products in the larger space are defined in terms of a kernel function $K(x, y)$ which can be selected to suit the problem. The hyperplanes in the large space are defined as the set of points whose cross product with a vector in that space is constant. The vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors which occur in the data base. With this choice of a hyperplane the points x in the feature space which are mapped into the hyperplane are defined by the relation:

$$\sum_i \alpha_i K(x_i, x) = constant$$

Note that if $K(x, y)$ becomes small as y grows further from x, each element in the sum measures the degree of closeness of the test point to the corresponding data base point x_i. In this way the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of points x mapped into any hyperplane can be quite convoluted as a result allowing much more complex discrimination between sets which are far from convex in the original space [10].

Conclusion

Akinator, the Web Genius, for some is just an online played game, a game similarly to a human being playing “Guess Who?”, however for the others, it is more then that, it is an astonishing Artificial Intelligence Program, that even with all the advances in the computing and the programming still considered as a prodigy that makes programmers and computer scientist look deeper and deeper to solve that prodigy.

So some of them mention that the Decision trees is the main key, but some other disagree and say the Binary trees is the main key, and other motion the that combines elements from decision trees and binary trees, as well as probabilistic methods and machine learning. Well tell now we can’t Prove nothing, therefore, the original creation of The algorithm that used in Akinator is still going to be a secret.

Bibliography

[1] *http://intelligenceartificiellesi.wordpress.com/2012/12/03/exemple-dun-systeme-expert-akinator/* visited on 12/09/2013.

[2] MITCHELL, Tom M. *Machine Learning*, 414 pp. 1997.

[3] UTGOFF, Paul E. Incremental induction of decision trees. *Machine learning*, 1989, vol. 4, no 2, p. 161-186

[4] SCHLIMMER, Jeffrey C. et FISHER, Douglas. *A case study of incremental concept induction.* In : AAAI. 1986. p. 496-501.

[5] QUINLAN, J.. Ross . *Improved use of continuous attributes in C4.5.* arXiv preprint cs/9603103, 1996.

[6] RONTOGIANNIS, Athanasios et DIMOPOULOS, Nikitas J. *A probabilistic approach for reducing the search cost in binary decision trees.* Systems, Man and Cybernetics, IEEE Transactions on, 1995, vol. 25, no 2, p. 362-370.

[7] SEN, Wu, LING-YU, Wu, YU, Long, et al. *Improved classification algorithm by minsup and minconf based on ID3.* In : Management Science and Engineering, 2006. ICMSE'06. 2006 International Conference on. IEEE, 2006. p. 135-139.

[8] *http://cslibrary.stanford.edu/110/BinaryTrees.html* visited on 01/15/2014.

[9] *http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html* visited on 01/10/2014.

[10] *http://en.wikibooks.org/wiki/Support_Vector_Machines* visited on 01/20/2014.