The Functional Decomposition and Machine Learning

Katja B. Rangelov[1]

Abstract - **Learning from data is the central theme of Knowledge Discovery in Databases (KDD) and Machine Learning (ML) community. In order to handle large databases, certain assumptions (e.g. Occam's Razor [13]) are necessary to make the problem tractable. The requirement to find solutions of low complexity is both in Machine Learning and in the multiple-valued logic synthesis community. The purpose of this paper is to demonstrate the applicability functional decomposition of switching function paradigm to the Machine Learning.**

Key words **- functional decomposition, multiple–valued function, machine learning, knowledge data discovery, inductive learning.**

I. INTRODUCTION

When dealing with a complex problem, a good strategy is to decompose it to less complex and more manageable sub-problems.

Functional decomposition of binary function was proposed by Ashenhurst [16] in the beginning 1950s, as a method of Boolean logic minimization. While this process has been known for many years, it could not be utilized because of the large complexity computation procedures that are required. In the late 1980s, functional decomposition was re-introduced as an application to Field Programmable Gate Array's (FPGA) design synthesis [17].

Since then decomposition has been applied to many aspects of Boolean and multiple-valued logic synthesis [8, 15].

The relationship between machine learning and logic synthesis is based on the similarities between realising circuits with low complexity (smaller size, simpler description) and natural assumption of Occams'Razor [13]. In logic synthesis, the result of synthesis is a circuit designed with respect to the minimal number of gates, inputs, literals, or product terms. In the case of machine learning, the goal is a reduction of an instance space (compressing sets of examples, attributes and attributes-value triples in a technique called a partition triple). The problems are analogues. In machine learning, we have a database with fields and records. A set of records defines the concept. In logic synthesis, the fields are binary variables (inputs) of the circuit and each record is a specification. The entire set of these records defines the circuits.

Using Occam's Razor, which is a principle in machine learning that states that if several theories explain all the facts that the simplest theory is the best, does this.

[1]Katja B. Rangelov is with the Technical University of Sofia, the Faculty of Computer Systems and Control, 1756 Sofia, Bulgaria, E-mail: krang@tu-sofia.acad.bg, krang@abv.bg,

II. MACHINE LEARNING

In machine learning the idea is to find patterns in the data, such that the data can be partitioned into smaller concepts (data blocks), which correspond to the sub-blocks of the decomposition. The principle of using decomposition in machine learning is to reduce a given function specified by a set of care minterms (samples or examples) to a composition of smaller function (concepts).

The result is a set of expression that describes suitable intermediate concepts. Each of these intermediate concepts can then be decomposed further, leading to expressions that form a more comprehensible description of the learned concepts. The advantage of using decomposition to obtain useful intermediate concepts is that it leads to a result specified as a hierarchy of compositions. The produces the description of the original function as a hierarchy of sub-functions and variables [5, 6, 7, 11], which leads to learning that is faster, involves smaller error and gives better explanation of the learned concepts.

The term *machine learning* is defined as a process by which a machine gains the capability to solve problem by examining *examples* or data. Machine learning is a process called *inductive learning* [3, 5, 7,] which uses empirical evidence, in the form in examples, to derive *rules* for the given data. These system use *automated inductive inference* (in the machine learning paradigm) to give rules or meaning to the data (known as *classification*). Rules are operations on variables, such as the average function, addition, subtraction, etc. Finding rules on variables is a process of discovering patterns and relations that exist between variable combinations. Machine learning is not en exact methodology, but is an attempt of learning based on heuristics and probabilities. This way of learning is becoming more and more important as computers provide relatively inexpensive means to collect and store data. The traditional methods, such as manual data analysis, are insufficient to fully evaluate a given data set. Instead, a new method called Knowledge Discovery in Databases (KDD) [3, 7] is being used to analyse data using analytical tools from statistics, pattern recognition, and artificial intelligence.

In other words, learning is done by using all the known outputs of a given function to help in the determination of value for all don't knows in the output of the given function. By treating examples as cares and considering function with many don't cares, a machine learning problem can be directly converted to a logic synthesis problem.

Definition1: *The components of a function* $y = f(x_0, x_1, \ldots, x_{n-1})$ *where the set* $\{x_0, x_1, \ldots, x_{n-1}\}$ *is defined as the input*

a*ttributes, minterms are defined as* example*s, and the output y is the* target concept *to be learned.*

Definition2: *During decomposition of machine learning functions,* $y = F(x_0, x_1, \ldots, x_{n-1})$ *any* x_i *is known as either an input attribute or an intermediate concept (e.g. a* Φ *composite function in Curtis decomposition [4]).*

The setting of unknown values to known values is done by creating a network of multi-valued input and multivalued output blocks by decomposing the original function into a hierarchical network of multi-leveled blocks (intermediate concepts). A machine learning algorithm is evaluated on its learning effectiveness by how it reduces the *error* of the resulting network. **Error** is how well the algorithm in question sets don't know terms to care terms. A common method for evaluating a machine learning algorithm is to select a *training set*, which is a random sampling of the original known values from the *test* function. The result of learning the *training set* is then compared to the original test function. If the expression has a high error rate then it does not approximate the *test* function well and is not a useful way to describe the function.

Given that induction is a method of extrapolating samples of a function, the extrapolating process is very complex unless some reasonable simplification assumptions are used. For a Boolean function over *n*-attributes, the function has a truth table with 2^n rows. Any truth table with 2^n rows can represent 2^{2n} different functions. Because of the large number of possible functions, it is difficult to find a hypothesis function *g* that approximates *f* given a small set of examples of *f.*

To allow reasonable results in the extrapolation process for finding , assumptions must be used. The assumption that is generally made is the one of low complexity, as in **Occam's Razor** (also known as Ockham's Razor). In Occam's Razor the most likely hypothesis is the simplest one that is consistent with all observations.

The logic circuit and machine learning are similar, but there are some significant differences. The biggest difference is that most circuit – related multi-valued logic problems are nearly completely specified, while functions in machine learning tend to be 99.9% unspecified in their respective learning domain.

III. Missing Attributes

The possibility exists that data collected could have *missing attributes* in the example. A missing attribute is represented by an input don't know to denote the missing data. To associate the concepts of a missing attribute to that of logic synthesis, input doesn't know is akin to input don't care. In logic synthesis, an input A is don't care when, given a set of all other inputs, all possible values of A do not result in a change in the output of the function.

Definition 3. *An* **input don't care** *is defined as covering all possible values for a given variable.*

In machine learning, the possibility exists that data collected could have a missing attribute [1, 9, 10] in the data.

Definition 4. *An* **input don't know** *is defined as representing one (unknown) data value, not all possible values for a given variable.*

For example [1], a questionnaire asks for a person's age. It is possible that the person does not want to state his/her age; in general, instead of guessing at an approximate age, a don't know is placed in the category. In terms of logic synthesis, a don't care implies that the person is all the ages between 0 and 120. While a don't know implies that a person has only one age between 0 and 120.

One possible method of determining missing attributes is to find two input cubes C_1 and C_2 that intersect and have the same output value. Thus, $C_1 \cap C_2 \neq \varnothing$ and the minimal solution for the missing attributes for these two cubes is $C_1 \cap C_2$. This is illustrated in the next example.

Example 1 [1]. *Given the set of input cubes (a dash indicates a don't know, or a don't care):* _bcd*,* a _ cd*,* ab _ d *and .. Figure 1a shows the logic synthesis reduction method, where all the minterms are placed in essential prime implicants. The result is a minimum circuit with four product terms and one sum term. Figure 1b shows the machine learning reduction. The cube:* _bcd*, is thought to have a don't know such that* _bcd *specifies that either the minterm* abcd *exists or the minterm* –bcd *exists. Because the intersection of all the input cubes is* abcd*, the minimal solution is* abcd*. (Note: the minimal solution does not imply that it is the best solution).*

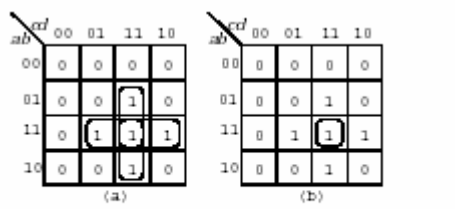

Figure 1. Don't knows in the input variables.

Another method of determining missing attributes is to use probabilities. Probabilities are used to determine the value of a missing attribute by determining how probable a value is. A baseline probability is defined as any value occurring more than X-times. If this occurs, then the value is accepted and a don't know can represent all values in the range when the baseline probability is zero or it is possible that no values are in the range. This methodology can be performed by using a preprocessing algorithm that determines (probabilistically) what the don't know values should be changed too. The following example illustrates the concept of using probabilities for determining the value of a don't know.

Example 2. [1] *Figures 2a through 2d show the possible combinations when given the cubes:* a_ *and* _b*. Because*

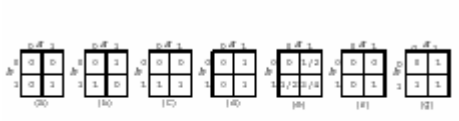

Figure 2. Using probability to find a function from cubes and

there are four combinations, then the probability of finding any of the Karnaugh maps in Figures 2a-d, is ¼ .Figure 2e shows the probabilities of finding an output value with a value of one. By assuming the probability baseline in the interval [0,1], every minterm is calculated to be true or false. By setting the probability baseline to be greater than 0.5 then only the minterm ab *is true, resulting in the Figure 2f. If the probability baseline is greater than 0.4, the function is shown in Figure 2g.*

Another method of determining missing attributes is to use relations [2], but the relations are only used to indefiy overlapping cubes. The relations treat don't know as don't cares, but overlapping cubes of different valued are stored. Example 3 illustrates the concept of a relation.

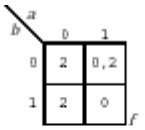

Figure 3. Missing attributes using relations; two values for the minterm .

Example 3 [1]. *Given the cubes:* a_ *and* _b*, the relation recognizes that the two cubes overlap at* ab*. For instance, if has the output value 0, and has the output value 2 then the cube has the output value of 0 or 2. The use of relations only preserves the information that the output of the minterm has two different values. This is shown in Figure 3a.*

For logic synthesis methods to be used on machine learning problems, missing attributes/input don't knows need to be represented in the data structure. A possible representation is to create a new cofactor for each variable that contains don't know information.

Theorem 1. *Given a* Q_f *-valued function* $f(x_0; x_1; : : :; x_{n-1})$*, each variable* x_i *has value* Q_i*, so that* f *may be denoted as* $f : Q_0 \times Q_1 \times \ldots \times Q_{n-1} \rightarrow Q_f$ *. To represent an input don't know, each variable* x_i *has value* $Q_i + 1$*, where the* $(Q_i + 1)^{th}$ *value represents a don't know. The function may now be denoted as* $f: (Q_0 + 1) \times (Q_1 + 1) \times \ldots \times (Q_{n-1} + 1) \rightarrow Q_f$ *. This is called the* **Don't Know** *(DK) representation.*

Proof. $f : Q_0 \times Q_1 \times \ldots \times Q_{n-1} \rightarrow Q_f \subset f (Q_0 + 1) \times (Q_1 + 1) \times \ldots \times (Q_{n-1} + 1) \rightarrow Q_f$ thus, the original function is still represented by the DK representation of the function.

□

For the partition $A|B$, a **DK partition matrix** is defined as the $(Q_0 + 1) \times (Q_1 + 1) \times \ldots \times (Q_{n-1} + 1)$ functional values of f, arranged in $|Q_A|$ rows, and $|Q_B|$ columns.

After creating the DK representation of f, and forcing the input don't knows in the function to values, it is necessary to transform the DK representation of f back to f's original functional representation.

Theorem 2. *Given that* f *is denoted as* $f (Q_0 + 1) \times (Q_1 + 1) \times \ldots \times (Q_{n-1} + 1) \rightarrow Q_f$ *then it is possible to transform* f *back to* $f : Q_0 \times Q_1 \times \ldots \times Q_{n-1} \rightarrow Q_f$ *, iff each* $Q_i + 1$ *value for variable* x_i *is not defined.*

Proof. If each $Q_i + 1$ value is not defined, then all input don't knows in the function are set to a given value. The addition of the Q_i+1 value of each variable is not needed and can be removed. This has no effect on f because the Q_i+1 value was not originally defined in the original function.

□

IV. CONCLUSION

Why use the functional decomposition? Decomposition has the advantage that it is not based on a set of operators or gates (in contrast to all other logic synthesis methods). This is especially notable in the case of multiple-valued logic, where the number of operators can increase as the value of the logic grows. Decomposition is not constrained by a technology or a pre-selected single theory. This has wide applications in both logic synthesis and machine learning.

Machine learning problems also contain continuous data or data that is in the form of real numbers, not only natural numbers. This type of data has shows itself to be very difficult for multi-valued function decomposition. There have been attempt to map real-valued variables onto natural numbers by discretatization.

Another problem that exists is data noise. Noise in machine learning could be caused by many different things – noise that is in the input and noise that is generated on the output.

Handling noise, discretatizion continuous data or data that is in the form of real number onto natural are needs to be looked into further in machine learning community.

REFERENCES

[1] C. M. Files, "New Functional Representation for the Decomposition of Machine Learning Problem", *Third Symposium on Logic Design and Learning, Conference Proceedings*, pp. Oregon, USA, May 2000.

[2] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, J.S. Zhang, "Decomposition of Multiple-Valued Relations", *Proc. IEEE International Symposium on Multiple-Valued Logic*, pp.13 – 18, Nova Scotia, Canada, May 1997.

[3] J. A. Goldman, M. L. Axtell, " On Using Logic Syntesis for Knowledge Discovery", *Tools with AI conference*, 1997

[4] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, Princeton, D. Van Nostrand Co. Inc., NJ, 1962.

[5] B. Zupan, M. Bohanec, I. Bratko, J. Demsar, "Machine Learning by Function Decomposition, *Proceedings of the Fourteenth International Conference Machine Learning* (ICML'97), pp. 421 – 429, Nashville, Tennessee, July 1997.

[6] B. Zupan, M. Bohanec, J. Demsar, I. Bratko, "Feature transformation by function decomposition", *IEEE Intelligent Systems & Their Applications*, vol. 13, pages 38-43, 1998.

[7] B. Zupan, M. Bohanec, I. Bratko, B. Cestnik, "A data set decomposition approach to data mining and machine discovery", *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining,* pp. 229 – 302, Newport Beach, Canada, August 1997.

[8] T. Luba, "Decomposition of Multiple – Valued Functions", *Proc. of 25th IEEE International Symposium on Multiple-Valued Logic*, pp. 256 – 261, Bloomington, Indiana, USA, May 23 – 25, 1995.

[9] C. M. Files, M. A. Perkowski, " Multi-Valued Functional Decomposition as s Machine Learning method", *Proc. on 28th IEEE International Symposium on Multiple-Valued Logic*, pp. 173 – 178, Fukuoko, Japan, May 27 –29, 1998.

[10] C. M. Files, M. A. Perkowski, " An Error Reducing Approach to Machine Learning Using Multi-Valued Functional Decomposition", *Proc. on 28th IEEE International Symposium on Multi-Valued Logic*, pp. 167 – 172, Fukuoko, Japan, May 27 – 29, 1998.

[11] B. Zupan, *Machine Learning Based on Functional Decomposition*, PhD thesis, University of Ljubljana, Slovenia, 1997.

[12] Y. Abu-Mostafa, *Compexity in Information Theory*, Springer - Verlag, New York, 1988.

[13] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth, Occam's razor, *Information Processing Letters*, pp.377 – 380, 1987.

[14] C. M. Files, *A New Functional Decomposition Method As Applied to Machine Learning and VLSI Layout*, Ph.D. Dissertation, Portland State University, Portland Oregon, June 2000.

[15] M. A. Perkowski, S. Grygiel, "A Survey of Literature on Function Decomposition", T*echnical report*, Portland State University, Portland, Oregon, November, 1995.

[16] R. L. Ashenhurst, "The decomposition of switching functions", *International Symposium on Theory Switching Function*, pp. 74 – 116, 1959.

[17] Y. T. Lai, M. Pedram, S. B. K. Vrudhula "BDD based decomposition of logic function with application to FPGA synthesis, Design Automation Conference, pp. 642 – 647, 1993.