Using the Multi-Valued Functional Decomposition to Evaluate Error in 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. The increase of data volume is caused greater difficulties to extract useful information for decision support or different analysis tasks. While functional decomposition was originally created for the minimization of logic circuits can be used successfully in the concept of learning in ML, by reducing the complexity of a given data set. The main difference is that machine learning problem have a large number of output don't cares. The purpose of this paper is to demonstrate the applicability functional decomposition in reduced the resulting error of the functional decomposer, where the error is a measure of how well a machine learning algorithm approximated the true function.**

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

1. INTRODUCTION

Many people are inclined to associate artificial intelligence (AI) with the attempt to replicate human intelligence. While this is a valid long-term goal, most work in AI are concerned highly restricted tasks, often of practical importance, with the intention of developing heuristic methods for accomplishing them.

Machine learning is a sub field of AI that addresses the formulation and modification of theories as a result of observations. Learning is diverse, encompassing activities like *explanation-based* learning (where an existing theory is tweaked, often to make it more efficient, in the light of a single new fact), *empirical learning* (which develops a theory from scratch, guided by a substantial number of observations), and learning *theory* (which identifies classes of theories that are easy or difficult to learn and quantities interrelationships among learning variables such as learning time, accuracy, and the number of observations) [11, 12].

Rather than discuss AI and learning in the abstract, this paper illustrates some important ideas in the context of a particular task of machine learning.

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 [18] 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 [19].

Since then decomposition has been applied to many aspects of Boolean and multiple-valued logic synthesis [8, 17].

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 [15]. 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.

2. 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, 13], 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. This system is used *automated inductive inference* (in the machine learning paradigm) to give rules or meaning to the data (known as *classification*). Rules are operations on

[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.bg, krang@abv.bg,

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.

The setting of unknown values to known values is done by creating a network of multi-valued input and multi-valued output blocks by decomposing the original function into a hierarchical network of multi-levelled blocks (intermediate concepts). The 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.

3. MULTI-VALUED FUNCTIONAL DECOMPOSITION

This section considers the basic principles of the decomposition of multi- valued function. Decomposition of multi-valued function is an extension of the decomposition of Boolean functions [1, 2, 4].

Definition 1: Given a multiple-valued variable x_i, the set of values that x_i may assume is $C_j = \{0, 1, \ldots, c_{j-1}\}$. Then an n-input, m-output, multi-valued function is defined as the mapping: $f(x_0, x_1, \ldots, x_{n-1}) = C_0 \times C_1 \times \ldots \times C_{n-1} \to D_0 \times D_1 \times \ldots \times D_{m-1}$, where $D_j = \{0, 1, \ldots, (p-1), -\}$, with p equal to the number of output values represented in the p-valued logic and "-" represented a *don't care* values.

Definition 2: A function $f(x_0, x_1, \ldots, x_{n-1})$ is decomposable under *bound set* $\{x_0, \ldots, x_{i-l}\}$ and *free set* $\{x_{i-1}, \ldots, x_{n-1}\}$, $0<i<n$, $0 \neq l$ if and only if f *can be* represented as the composite function $F(G_0(x_0, \ldots, x_{i-1}), \ldots, G_{j-1}(x_0, \ldots, x_{i-1}), x_{i-1}, \ldots, x_{n-1})$, where $0<j<i-l$. If l equal 0 then f is called *disjunctively decomposable*, otherwise, it is known as *non-disjunctively decomposable*.

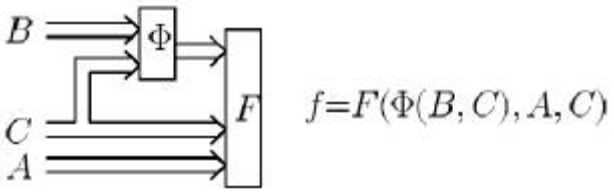

Fig.1:The Decomposition in general

The principle idea of the decomposition using the notation from Definition 2 is shown in Fig. 1. Note that the function is decomposable under a given bound set, the function can be separated into two new functions. This is known as a simple decomposition because one decomposition was done on the function. To find fully decompose the function, is used the iterative process. First the function is decomposed, then the sub-function that are created from the first decomposition are decomposed. This iterative process continues until a given function cannot be decomposed further under given complexity measure.

For an n-input function, the number of simple *disjunctive decomposition* is 2^n, while the number of simple *non-disjunctive decomposition* is 3^n. Thus, evaluating all possible partitions is a NP-complete problem when trying to find fully decompose the function, because of the exponential size of *simple non-disjunctive decomposition*. It is needed to use some heuristics methods to reduce the search complexities finding partitions in decomposition, but this does not exactly means that the problem of finding non-disjunctive decomposition is become trivial.

Definition3: Given a k-valued, completely specified function f, with *bound set* B and *free set* A, then for the partition A B, a *partition matrix representation* of f is defined as a rectangular array, where the *column* correspond to the variables in the *bound set* and the *row* correspond to the variables in the *free set*.

There are the following statement, using the Definition 2 and Definition 3:

1.The array has k^B columns and k^A rows.

2. Given a k-valued function f, with a bound set B, then the corresponding partition matrix has l distinct columns, where l is called the *Column Multiplicity* of a partition.

3. The *Column Multiplicity* [17] for the function can be reduced if the function is incompletely specified, by finding columns that are *compatible* and combing the two column by setting *don't'care* values. By compatible, for every row, the possible output value-sets of the first column (a number or a *don't care*) intersect the non-empty sets of the corresponding output value-sets of the second column.

4. Thus, the represent f as a composite function in the form: $f = F (G_0 (), \ldots, G_{j-1} (), x_i, \ldots, x_{n-1})$, where each G function has inputs $(x_0, \ldots, x_{i-1})$ then $j = |log_k l|$, G functions are needed..

This concept of decomposition, which is explained, is general and because is general it can be applied to any type of logic elements or structures. 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.

4. PARTITION SELECTING USING REDUCTING AND EVALUATED ERROR

The biggest difference between logic synthesis and machine learning is the difference in the number of care terms in logic synthesis versus the number of care terms in machine learning problem. The *don't cares* in the machine learning problems should be considered as *don't know*. By *dont' know* it means that the value is not known and the caution must be used in setting *don't know* to a value.

With this concept of a don't known, it is possible to find the method for selecting partitions for machine learning based functional decomposition.

If the function is completely specified, the only way that two column can be compatible is if for every row, the output of the first column is equal to the output value of the second column (in the same row). If the function is p-valued, completely specified, and each value is equally likely, then for every row the probability of matching one column to another is $1/p$.

For a column with B rows, the probability of finding a pair of *compatible* columns is $(1/p)^B = 1/p^B$. The result is that the more row in a column, the less likely it is to find a *compatibility* between two columns. The problem is that it doesn't take *don't cares* into consideration and the fact that machine learning data set are 99% unspecified.

If it assumed the probability of having a *don't care* is P(X) = 0.99, then the probability of having any other value is P(care) = (1-0.99)/ p = 0.01/p, given the function is p-valued and each value is equally likely. The only way to have incompatibility between two columns occurs only when, for a given row, the first column's value is some value and the second column's value is a value that is not equal to the first column's value. Thus, the probability of having an incompatibility between two columns is 0.01 (p-$1/p$).

If it is evaluated small bond set (small number of column and large number of row in partition table) may result in a larger probability of incompatible column. The basic assumption here is that by increasing the probability of incompatible columns, we have decreased the error in the resulting network, because the possibility of combing a don't care with an incorrect care term is reduced. This is based on the probability of combining two columns is much smaller, and thus, the probability of combining a don't care with values is also smaller.

Another complexity that come up is determining column complexity. On of the most using methods for finding column complexity with many don't cares is graph colouring. In general, graph colouring is NP-complete problem because it existed exponential in complexity. It is reasonable to assume that small bound set with small number of columns should have a much faster run time the medium-sized (half the variables in the bound set, the other half in the free set) bound sets. in the case of the medium or large bound sets, the probability of two column being incompatible is much samller than in the case of small bound set. But the number of compatible column can be very large. In fact, because there are so many possible colouring in the graph algorithm, selecting compatible column is done randomly with no basis on probabilities. In this case, having the medium or large bound set will result in great error.

In domain of logic synthesis (the small percent of don't cares) the possibility of finding the good decomposition with small bound sets would probably not work. But, from the previous assumption, the small bound set are the best way to decompose machine learning problems.

5. DECOMPOSITION ALGORITHM

5.1. DATA STRUCTURED THE FUNCTION

The representation of a function is very important and there are the different types of the data structured. One of them is Binary Decision Diagrams (BDD), MDD, decision tree and others [9, 16].

5.2. INESENTIAL VARIABLES

There are tree different classes of variables. A variable is *essential*, *inessential* or *vacuous*. *Essential* or *vacuous* variables are easily defined as variables that have an impact on the output. An *inessential* variable is variable that by setting don't cares in certain way, the variable either may become essential or vacuous.

To determine which variables can be made vacuous is used the compatibility graph algorithm. Each node the graph represented inessential variable, each vertex between two nodes designates that the both variables can be forced vacuous

in the same time. It is used maximum clique algorithm to determine which inessential variable can be forced to become vacuous.

5.3. MACHINE LEARNING COMPLEXITY MEASURES

The complexity measure, called Decomposition Functional Cardinality (DFC) [14] is the sum of the cardinalities of the component functions in a combinational representation, when the sum is minimized. Any function is allowed as a component, but its costs goes up with its cardinality. The cardinality of an n-input, m-output binary function is $2^n * m$.

5.3. THE ALGORITHM

The algorithm is divided into following parts [10]:

1. Find all *inessential* variable (using a maximum clique algorithm [17]) and remove its from the function.

2a. Determine if the current function's DFC is larger than he best DFC found so far. If it is, then return (not all function are decomposed).

2b. Evaluated all two variable bound sets.

2c. Determined the best partition to be used: evaluate the column multiplicity of each partition and selecting those with higher ranks. The partition with the smaller column multiplicity are ranked higher than those with higher column multiplicity.

2d. From the list of possible partitions from 2b, decomposed each function f into sub-function F and Ö. Recursively repeat step 2 with F and Ö, separetly.

4. CONCLUSION

This method for functional decomposition is focused on the highly unspecified multi-valued function. It is allowed to reduce the error in a solution. From the data analysis, this method of functional decomposition did well on reducing and evaluated the error between the original function and the solution.

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] J. R. Quinlan, C4.5: Programs for Machine Learning, San Mateo, California: Morgan Kaufmann, 1993.

[12] J. R. Quinlan, "A Case Study in Machine Learning", *Proceedings 16th Australian Computer Science Conference*, pp., 731-737, Brisbane, Australia, 1993.

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

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

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

[16] 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.

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

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

[19] 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.